

5^a Edición

Este libro proporciona un análisis comprensible y completamente actualizado sobre la organización y la arquitectura de computadores. El texto abarca las novedosas áreas del diseño superescalares, las características del diseño del IA-64 y las tendencias en la organización de procesadores paralelos. Se abordan las necesidades de los estudiantes en cuanto a que se abordan tanto los principios fundamentales como el papel crítico de las prestaciones como guía del diseño de los computadores. El libro también incluye información práctica de ayuda al profesor, con recursos suplementarios e interactivos.

CARACTERÍSTICAS A DESTACAR:

- Utilización de numerosos ejemplos probados, especialmente relacionados con el microprocesador Pentium.
- Enfoque pedagógico utilizado que posibilita al lector evaluar los resultados del diseño del repertorio de instrucciones.
- Presentación ampliada del procesamiento superescalares, incluyendo los nuevos ejemplos del UltraSparc II y del MIPS R10200.
- Tratamiento detallado de la organización de buses, que posibilita al lector evaluar cuestiones clave del diseño.
- Incorpora un capítulo nuevo donde se estudian de forma detallada los procesadores RISC.
- Tratamiento amplio y comprensible de las funciones y estructuras de E/S.

EL SITIO WEB de este libro sirve de gran ayuda a estudiantes, profesores y profesionales, ya que:

- Cuenta con enlaces con importantes servidores que contienen información actualizada relacionada con el material presentado en el texto.
- Proporciona transparencias con las figuras originales del libro en formato PDF (Adobe Acrobat).
- Incluye un conjunto de notas en PDF como apuntes de clase.
- Contiene un conjunto de diapositivas en PowerPoint para impartir clases.

WILLIAM STALLINGS ha realizado una contribución única para la comprensión del amplio campo de conocimiento que abarcan las áreas de la arquitectura y de las redes de computadores. Es autor de 17 libros, que acompañados de sus ediciones revisadas hacen un total de 37 libros, que cubren diversos aspectos de estas áreas. Ha recibido tres veces el premio al mejor libro de texto del año en computación, otorgado por la "Text and Academic Authors Association". Los libros premiados son: Computer Organization and Architecture, Prentice Hall, 1995; Data and Computer Communications, Prentice Hall, 1997; Operating Systems, Prentice Hall, 1998. Todos ellos han sido traducidos al castellano por Prentice Hall.

En más de 20 años dedicado a este campo, el Dr. Stallings ha sido asesor, gestor técnico, y ejecutivo de varias empresas de alta tecnología. Actualmente es un asesor independiente, entre cuyos clientes se encuentran fabricantes y distribuidores de computadores y redes, empresas desarrolladoras de software, e instituciones gubernamentales de investigación en temas de vanguardia. El Dr. Stallings es conferenciante habitual y publica artículos con frecuencia en revistas técnicas y publicaciones comerciales.

William Stallings es Doctor en informática por el MIT, y obtuvo el título B.S. en ingeniería eléctrica en la Universidad de Notre Dame.

Organización y Arquitectura de Computadores

LibroSite es una página web asociada al libro, con una gran variedad de recursos y material adicional al texto para los profesores como para estudiantes. Apoyos a la docencia, ejercicios de autocontrol, enlaces relacionados, material de investigación, etc., hacen de LibroSite el complemento académico perfecto para este libro.

www.LibroSite.net/Stallings/



ISBN 84-205-2993-1



Stallings

Prentice
Hall

AJ Merriam
5^a Edición

Organización y Arquitectura de Computadores

Symmetric Multiprocessing

Superscalar

Multi-level parallelism

Power PC

Instruction

Prentice
Hall

William Stallings

ORGANIZACIÓN Y ARQUITECTURA DE COMPUTADORES

Diseño para optimizar prestaciones

Quinta edición

William Stallings

Traducción:

Antonio Cañas Vargas

Julio Ortega Lopera

Francisco José Pelayo Valle

Beatriz Prieto Campos

Departamento de arquitectura y tecnología de computadores

Universidad de Granada

Coordinación y revisión técnica:

Alberto Prieto Espinosa

Catedrático de arquitectura y tecnología de computadores

Universidad de Granada

PRENTICE HALL

Madrid • México • Santafé de Bogotá • Buenos Aires • Caracas • Lima • Montevideo • San Juan
San José • Santiago • São Paulo • White Plains

datos de catalogación bibliográfica

ORGANIZACIÓN Y ARQUITECTURA
DE COMPUTADORES. Quinta edición
STALLINGS William

PRENTICE HALL IBERIA, Madrid, 2000

ISBN: 84-305-2993-1
Materia: Informática. 681.3

Formato 195 x 250 Páginas: 760

ORGANIZACIÓN Y ARQUITECTURA DE COMPUTADORES
STALLINGS William

No está permitida la reproducción total o parcial de esta obra
ni su tratamiento o transmisión por cualquier medio o método.
sin autorización escrita de la Editorial.

DERECHOS RESERVADOS
© 2000 PEARSON EDUCACIÓN S.A.
Núñez de Balboa, 120
28006 MADRID

SBN: 84-205-2993-1
Depósito legal: M. 41.313-2001
Última reimpresión: 2001

PRENTICE HALL es un sello editorial autorizado de PEARSON EDUCACIÓN, S. A.

ducido de:
COMPUTER ORGANIZATION AND ARCHITECTURE, FIFTH EDITION
2000, 1996 por Prentice Hall Inc.
ISBN 0-13-081294-3

dicción en español:
Editor: Andrés Otero
Asistente editorial: Ana Isabel García
Editor de producción: José Antonio Clares
Diseño de cubierta: DIGRAF, S.A.
Composición: COPIBOOK, S.L.
Impreso por: LAVEL, S.A.

PRESO EN ESPAÑA - PRINTED IN SPAIN

Este libro ha sido impreso con papel y tintas ecológicos

*Para mi valiente esposa
ATS
y sus constantes compañeros Geoffroi
y Princesse Kate Lan Kinetic.
Los Niños del Paraíso*



SITIO WEB PARA EL LIBRO «ORGANIZACIÓN Y ARQUITECTURA DE COMPUTADORES» *Quinta edición.*

En la dirección web <http://www.shore.net/~ws/COASe.html> se encuentra diverso material de ayuda para los profesores y estudiantes que utilicen este libro. Incluye los siguientes elementos¹:



Materiales de ayuda para impartir cursos

Los elementos de ayuda para impartir cursos son:

- Copia de las figuras del libro en formato PDF
- Conjunto detallado de notas en formato PDF, adecuado como apuntes de los alumnos o para usar como esquemas
- Conjunto de transparencias/diapositivas en PowerPoint como ayuda para impartir las clases
- Una sección de erratas del libro, actualizada al menos mensualmente



Cursos sobre organización y arquitectura de computadores

El servidor web COASe incluye enlaces a páginas web que tratan sobre cursos relacionados con el libro. Estos lugares suministran ideas muy útiles sobre la planificación y ordenación de los temas, así como diversos apuntes y otros materiales.

¹ Estas páginas web, que son regularmente actualizadas, se encuentran en el idioma original del libro (inglés).



Sitios web útiles

El servidor COASe incluye enlaces a sitios web relevantes, organizados por capítulos. Los enlaces abarcan un amplio espectro de tópicos y posibilitan al alumno para explorar puntualmente temas en profundidad.



Lista de correo a través de Internet

Se mantiene una lista de direcciones de e-mail para que los profesores que utilizan el libro puedan intercambiar información, sugerencias y dudas entre ellos y con el autor. La información necesaria para ser incluido en esta lista se encuentra en el sitio web.



Herramientas de simulación

El sitio web incluye un enlace a un servidor web que contiene el simulador SimpleScalar. Este simulador es una herramienta que puede usarse para analizar y experimentar con problemas de diseño de procesadores. El servidor incluye software copiable en el equipo del usuario e información básica. El manual del profesor contiene más información sobre cómo copiar el simulador, utilización del software y propuestas de proyectos para realizar por los estudiantes. Véase el Apéndice B para tener más información.

Contenido

Prólogo	XXI
Prólogo a la edición en español	XXV

PARTE I Visión general

Capítulo 1. Introducción	3
1.1. Organización y arquitectura	5
1.2. Estructura y funcionamiento	6
Funcionamiento	7
Estructura	9
1.3. Esquema del libro	12
Evolución y prestaciones del computador	12
Buses del sistema	13
Memoria interna	13
Memoria externa	13
Entrada/salida	13
Apoyo al sistema operativo	13
Aritmética de computadores	14
Repertorios de instrucciones	14
Estructura y funcionamiento de la CPU	14
Computadores de repertorio reducido de instrucciones	14
Procesadores superescalares y paralelismo a nivel de instrucción	14
Funcionamiento de la unidad de control	15
Control microprogramado	15
Procesamiento paralelo	15
Lógica digital	15

Contenido

1.4. Internet y recursos Web	15
Sitios Web de este libro	15
Otros sitios Web	16
Grupos de noticias USENET	16
Capítulo 2. Evolución y prestaciones de los computadores	17
2.1. Una breve historia de los computadores	19
La primera generación: los tubos de vacío	19
La segunda generación: los transistores	26
La tercera generación: los circuitos integrados	28
Últimas generaciones	34
2.2. Diseño para conseguir mejores prestaciones	38
Velocidad del microprocesador	39
Equilibrio de prestaciones	40
2.3. Evolución del Pentium y del PowerPC	42
Pentium	43
PowerPC	44
2.4. Lecturas y sitios Web recomendados	45
2.5. Problemas	46

PARTE II El computador

Capítulo 3. Buses del sistema	49
3.1. Componentes del computador	51
3.2. Funcionamiento del computador	54
Los ciclos de captación y ejecución	54
Interrupciones	58
Funcionamiento de las E/S	65
3.3. Estructuras de interconexión	66
3.4. Interconexión con buses	67
Estructura del bus	68
Jerarquías de buses	70
Elementos de diseño de un bus	72
3.5. PCI	76
Estructura del bus	76
Órdenes del PCI	80
Transferencias de datos	81
Arbitraje	83
3.6. Lecturas y sitios Web recomendados	84
3.7. Problemas	85
Apéndice 3A. Diagramas de tiempo	87
Capítulo 4. Memoria interna	91
4.1. Conceptos básicos sobre sistemas de memoria de computadores	93

Características de los sistemas de memoria	93
Jerarquía de memoria	95
4.2. Memoria principal semiconductor	99
Tipos de memorias semiconductoras de acceso aleatorio	99
Organización	101
Lógica del chip	102
Encapsulado de los chips	104
Organización en módulos	105
Corrección de errores	106
4.3. Memoria cache	111
Principios básicos	111
Elementos de diseño de la cache	113
4.4. Organización de la cache en el Pentium II y el PowerPC	125
Organización de cache en el Pentium II	125
Organización de cache en el PowerPC	128
4.5. Organización avanzada de memorias DRAM	129
DRAM mejorada	129
DRAM cache	130
DRAM síncrona	130
DRAM rambus	131
RamLink	132
4.6. Lecturas y sitios Web recomendados	134
4.7. Problemas	135
Apéndice 4A. Prestaciones de las memorias de dos niveles	138
Localidad	138
Funcionamiento de la memoria de dos niveles	141
Prestaciones	141
Capítulo 5. Memoria externa	145
5.1. Discos magnéticos	147
Organización y formato de los datos	147
Características físicas	148
Parámetros para medir las prestaciones de un disco	152
5.2. RAID	155
Nivel 0 de RAID	156
Nivel 1 de RAID	160
Nivel 2 de RAID	160
Nivel 3 de RAID	161
Nivel 4 de RAID	162
Nivel 5 de RAID	163
Nivel 6 de RAID	163
5.3. Memoria óptica	163
CD-ROM	163
WORM	166
Disco óptico borrable	167
Disco vídeo digital	167
Discos magnético-ópticos	168
5.4. Cinta magnética	168

Contenido

5.5. Lecturas y sitios Web recomendados	169
5.6. Problemas	170
Capítulo 6. Entrada/salida	173
6.1. Dispositivos externos	176
Teclado/monitor	177
Controlador de disco (Disk Drive)	178
6.2. Módulos de E/S	180
Funciones de un módulo	180
Estructura de un módulo de E/S	182
6.3. E/S programada	183
Resumen	183
Órdenes de E/S	183
Instrucciones de E/S	185
6.4. E/S mediante interrupciones	186
Procesamiento de la interrupción	187
Cuestiones de diseño	190
Controlador de interrupciones Intel 82C59A	191
La interfaz programable de periféricos Intel 82C55A	192
6.5. Acceso directo a memoria	195
Inconvenientes de la E/S programada y con interrupciones	195
Funcionamiento del DMA	195
6.6. Canales y procesadores de E/S	198
La evolución del funcionamiento de las E/S	198
Características de los canales de E/S	198
6.7. La interfaz externa: SCSI y FireWire	200
Tipos de interfaces	200
Configuraciones punto-a-punto y multipunto	201
Interfaz SCSI (Small Computer System Interface)	201
Bus serie FireWire	209
6.8. Lecturas y sitios Web recomendados	214
6.9. Problemas	214
Capítulo 7. El soporte del sistema operativo	219
7.1 Conceptos básicos sobre sistemas operativos	221
Objetivos y funciones del sistema operativo	221
Tipos de sistemas operativos	224
7.2 Planificación	232
Planificación a largo plazo	233
Planificación a medio plazo	233
Planificación a corto plazo	233
7.3. Gestión de la memoria	238
Intercambio (Swapping)	238
Definición de particiones	240
Paginación	242
Memoria virtual	243

Buffer de traducción anticipada (Translation Lookaside Buffer)	247
Segmentación	247
7.4. Gestión de memoria en el Pentium II y en el PowerPC	250
Hardware de gestión de memoria en el Pentium II	250
Hardware de gestión de memoria en el PowerPC	255
7.5. Lecturas y sitios Web recomendados	258
7.6. Problemas	258
 PARTE III	
La unidad central de procesamiento	
Capítulo 8. Aritmética del computador	265
8.1. La unidad aritmético-lógica (ALU)	267
8.2. Representación de enteros	267
Representación en signo y magnitud	268
Representación en complemento a dos	269
Conversión entre longitudes de bits diferentes	271
Representación en coma fija	273
8.3. Aritmética con enteros	273
Negación	273
Suma y resta	275
Multiplicación	277
División	284
8.4. Representación en coma flotante	287
Fundamentos	287
Estándar del IEEE para la representación binaria en coma flotante	291
8.5. Aritmética en coma flotante	293
Suma y resta	294
Multiplicación y división	297
Consideraciones sobre precisión	297
Estándar del IEEE para la aritmética binaria en coma flotante	300
8.6. Lecturas y sitios Web recomendados	302
8.7. Problemas	303
Apéndice 8A. Sistemas de numeración	306
Sistema decimal	306
Sistema binario	307
Conversión entre binario y decimal	307
Notación hexadecimal	309
 Capítulo 9. Repertorios de instrucciones: características y funciones	311
9.1. Características de las instrucciones máquina	313
Elementos de una instrucción máquina	313
Representación de las instrucciones	314
Tipos de instrucciones	315
Número de direcciones	316
Diseño del repertorio de instrucciones	318

9.2. Tipos de operandos	319
Números	319
Caracteres	320
Datos lógicos	321
9.3. Tipos de datos en el Pentium II y el PowerPC	321
Tipos de datos en el Pentium II	321
Tipos de datos en el PowerPC	322
9.4. Tipos de operaciones	324
Transferencia de datos	326
Aritméticas	327
Lógicas	328
Conversión	330
Entrada/salida	331
Control del sistema	331
Control de flujo	331
9.5. Tipos de operaciones en el Pentium II y el PowerPC	336
Tipos de operaciones del Pentium II	336
Tipos de operaciones del PowerPC	343
9.6. Lenguaje ensamblador	346
9.7. Lecturas recomendadas	348
9.8. Problemas	348
Apéndice 9A. Pilas	353
Pilas	353
Implementación de la pila	353
Evaluación de expresiones	355
Apéndice 9B. «Little-, Big y Bi-Endian»	357
Orden de los bytes	357
Orden de los bits	361
10. Repertorios de instrucciones: modos de direccionamiento y formatos	363
10.1. Direccionamiento	365
Direccionamiento inmediato	367
Direccionamiento directo	367
Direccionamiento indirecto	367
Direccionamiento de registros	368
Direccionamiento indirecto con registro	369
Direccionamiento con desplazamiento	369
Direccionamiento de pila	371
10.2. Modos de direccionamiento en el Pentium y el PowerPC	371
Modos de direccionamiento del Pentium II	371
Modos de direccionamiento del PowerPC	374
10.3. Formatos de instrucciones	376
Longitud de instrucción	377
Asignación de los bits	378
Instrucciones de longitud variable	381
10.4. Formatos de instrucciones del Pentium y del PowerPC	385
Formatos de instrucción del Pentium II	385

Formatos de instrucción del PowerPC	388
10.5. Lecturas recomendadas	389
10.6. Problemas	390
Capítulo 11. Estructura y función de la CPU	395
11.1. Organización del procesador	395
11.2. Organización de los registros	396
Registros visibles para el usuario	397
Registros de control y de estado	398
Ejemplos de organizaciones de registros de microprocesadores	400
11.3. El ciclo de instrucción	401
El ciclo indirecto	402
Flujo de datos	403
11.4. Segmentación de instrucciones	405
Estrategia de segmentación	405
Prestaciones de un cauce segmentado	410
Tratamiento de saltos	417
Segmentación del Intel 80486	417
11.5. El procesador Pentium	419
Organización de los registros	419
Procesamiento de interrupciones	424
11.6. El procesador PowerPC	427
Organización de los registros	427
Procesamiento de interrupciones	430
11.7. Lecturas recomendadas	433
11.8. Problemas	434
Capítulo 12. Computadores de repertorio reducido de instrucciones	437
12.1. Características de la ejecución de instrucciones	439
Operaciones	441
Operandos	442
Llamadas a procedimientos	443
Consecuencias	444
12.2. Utilización de un amplio banco de registros	444
Ventanas de registros	445
Variables globales	447
Un amplio banco de registros frente a una cache	447
12.3. Optimización de registros basada en el compilador	448
12.4. Arquitectura de repertorio reducido de instrucciones	450
¿Por qué CISC?	451
Características de las arquitecturas de repertorio reducido de instrucciones	452
Características CISC frente a RISC	455
12.5. Segmentación de cauce en RISC	457
Segmentación con instrucciones regulares	457
Optimización de la segmentación	459

Contenido

12.6. MIPS R4000	460
Repertorio de instrucciones	461
Cauce de instrucciones	464
12.7. SPARC	468
Conjunto de registros del SPARC	468
Repertorio de instrucciones	468
Formato de instrucción	472
12.8. La controversia entre RISC y CISC	473
12.9. Lecturas recomendadas	474
12.10. Problemas	475
Capítulo 13. Paralelismo a nivel de instrucciones, y procesadores superescalares	479
13.1. Visión de conjunto	481
Superescalar frente a supersegmentado	482
Limitaciones	483
13.2. Cuestiones relacionadas con el diseño	486
Paralelismo a nivel de instrucciones y paralelismo de la máquina	486
Políticas de emisión de instrucciones	487
Renombramiento de registros	490
Paralelismo de la máquina	491
Predicción de saltos	492
Ejecución superescalar	493
Implementación superescalar	493
13.3. Pentium II	494
Unidad de captación y decodificación de instrucciones	495
Buffer de reordenación	497
Unidad de envío/ejecución	497
Unidad de retiro	499
Predicción de saltos	499
13.4. PowerPC	500
PowerPC 601	500
Procesamiento de saltos	504
PowerPC 620	507
13.5. MIPS R10000	507
13.6. UltraSPARC-II	509
Organización interna	509
Cauce segmentado	510
13.7. IA-64/MERCED	511
Motivación	512
Organización	513
Formato de instrucción	514
Ejecución con predicados	515
Carga especulativa	519
13.8. Lecturas y sitios Web recomendados	522
13.9. Problemas	523

PARTE IV	
La unidad de control	
Capítulo 14. Funcionamiento de la unidad de control	531
14.1. Microoperaciones	533
El ciclo de captación	534
El ciclo indirecto	536
El ciclo de interrupción	536
El ciclo de ejecución	537
El ciclo de instrucción	538
14.2. Control del procesador	539
Requisitos funcionales	539
Señales de control	540
Un ejemplo de señales de control	542
Organización interna del procesador	544
El Intel 8085	545
14.3. Implementación cableada	550
Entradas de la unidad de control	550
Lógica de la unidad de control	551
14.4. Lecturas recomendadas	553
14.5. Problemas	553
Capítulo 15. Control microprogramado	555
15.1. Conceptos básicos	557
Microinstrucciones	557
Unidad de control microprogramada	559
Control de Wilkes	562
Ventajas e inconvenientes	565
15.2. Secuenciamiento de microinstrucciones	566
Consideraciones respecto al diseño	566
Técnicas de secuenciamiento	566
Generación de direcciones	569
Secuenciamiento de microinstrucciones en el LSI-11	571
15.3. Ejecución de microinstrucciones	571
Una taxonomía de las microinstrucciones	572
Codificación de las microinstrucciones	575
Ejecución de microinstrucciones en el LSI-11	578
Ejecución de microinstrucciones en el IBM 3033	581
15.4. TI 8800	583
Formato de microinstrucción	584
Microsecuenciador	584
ALU con registros	589
15.5. Aplicaciones de la microprogramación	594
15.6. Lecturas recomendadas	595
15.7. Problemas	595

PARTE V
Organización paralela

Capítulo 16. Procesamiento paralelo	599
16.1. Organizaciones con varios procesadores	601
Tipos de sistemas de paralelos	601
Organizaciones paralelas	603
16.2. Multiprocesadores simétricos	604
Organización	605
Consideraciones de diseño de un sistema operativo de multiprocesador	609
Un SMP como gran computador	609
16.3. Coherencia de cache y protocolo MESI	613
Soluciones software	613
Soluciones hardware	614
El protocolo MESI	616
16.4. «Clusters»	619
Configuraciones de «clusters»	619
Consideraciones en el diseño del sistema operativo	622
«Clusters» frente a SMP	623
16.5. Acceso no uniforme a memoria	623
Motivación	624
Organización	624
Pros y contras de un computador NUMA	626
16.6. Computación vectorial	627
Aproximaciones a la computación vectorial	627
Unidad vectorial IBM 3090	633
16.7. Lecturas recomendadas	639
16.8. Problemas	640
Apéndice A. Lógica digital	645
A.1. Álgebra de Boole	646
A.2. Puertas	648
A.3. Circuitos combinacionales	650
Implementación de las funciones booleanas	650
Multiplexores	661
Decodificadores	662
Array lógico programable (PLA, Programmable Logic Array)	664
Memoria de solo lectura (ROM, Read Only Memory)	667
Sumadores	668
A.4. Circuitos secuenciales	671
Biestables	671
Registros	674
Contadores	676
A.5. Problemas	680
Apéndice B. Proyectos para enseñar arquitectura y organización de computadores	683

B.1. Proyectos de investigación	684
B.2. Proyectos de simulación	684
B.3. Lecturas/informes	685
Apéndice C. Pentium III	687
C.1. Incidencia de las instrucciones SIMD en las aplicaciones	688
C.2. El repertorio de instrucciones SSE	690
El repertorio SSE	693
C.3. Interconexión del Pentium III a su entorno	700
C.4. Lecturas y sitios Web recomendados	700
Glosario	703
Bibliografía	713
Índice alfabético	723

Prólogo

OBJETIVOS

Este libro trata sobre la estructura y funcionamiento de los computadores. Su objetivo es presentar, tan clara y completamente como sea posible, la naturaleza y características de los computadores de hoy en día.

Este objetivo es todo un reto por varias razones: En primer lugar, hay una gran variedad de sistemas que pueden recibir correctamente el nombre de computador, desde microprocesadores de un solo chip que cuestan unos pocos dólares, a supercomputadores que cuestan decenas de millones de dólares. Esta variedad es patente, no sólo en costes, sino también en tamaño, prestaciones y aplicaciones. En segundo lugar, la rapidez de los cambios que ha caracterizado siempre a la tecnología de computadores continúa sin pausa. Estos cambios cubren todos los aspectos de la tecnología de computadores, desde la tecnología subyacente de los circuitos integrados, usados para construir componentes de computadores, hasta el creciente uso de conceptos de organización paralela para combinar dichos componentes.

A pesar de la variedad y rapidez de los cambios en el campo de los computadores, se aplican sistemáticamente ciertos conceptos fundamentales. La aplicación de estos conceptos depende del estado actual de la tecnología y de los objetivos del diseñador en cuanto a precio y prestaciones. La intención de este libro es proporcionar una discusión concienzuda de los fundamentos de la organización y arquitectura de computadores, y relacionar éstos con problemas de diseño actuales.

El subtítulo del libro indica el tema y el modo de enfoque de este libro. Para diseñar computadores, siempre ha sido importante lograr altas prestaciones, pero este requisito nunca ha sido más difícil de satisfacer que hoy en día. Todas las características básicas de funcionamiento, incluyendo velocidad del procesador, velocidad y capacidad de la memoria y velocidad de interconexión de los datos, están creciendo rápidamente; además, están creciendo a velocidades diferentes. Esto hace que sea difícil diseñar un sistema equilibrado que maximice las prestaciones y utilice todos los elementos. Así, el diseño de computadores se convierte cada vez más en un juego de cambiar la estructura o la función de un área para compensar la desigualdad de prestaciones en otras áreas. Veremos este juego en numerosas áreas de diseño a lo largo del libro.

de captar estos cambios manteniendo a la vez una visión amplia y exhaustiva de la materia. Para empezar este proceso de revisión, la cuarta edición de este libro fue revisada exhaustivamente por numerosos profesores que imparten esta materia. El resultado es que, en muchos casos, la redacción se ha clarificado y ajustado, y las figuras se han mejorado. Además, han añadido una serie de nuevos problemas probados en la realidad.

Más allá de estos detalles para mejorar la pedagogía y la comodidad del usuario, se han hecho a cabo cambios relevantes a lo largo del libro. En líneas generales, se ha mantenido la iniciación de los capítulos, pero gran parte del material se ha revisado, y se han añadido otros conceptos. Algunos de los cambios más notables son los siguientes:

• **Memoria óptica:** La materia sobre memorias ópticas se ha ampliado a dispositivos de memoria magneto-ópticos.

Diseños superescalares: El capítulo sobre diseño superescalar se ha ampliado para añadir una discusión más detallada y dos nuevos ejemplos, el UltraSparc II y el MIPS R10000.

Repertorio de instrucciones multimedia: Se examina el conjunto de instrucciones MMX, usado en el Pentium II y Pentium III.

Ejecución anticipada y carga especulativa: Esta edición trae una discusión de estos conceptos recientes, que se centran en el diseño de la nueva arquitectura IA-64 de Intel y Hewlett-Packard.

Sistemas SMP, «clusters» y sistemas NUMA: El capítulo sobre organización paralela se ha reescrito por completo. El nuevo incluye descripciones detalladas y comparaciones entre multiprocesadores simétricos (SMP), «clusters», y sistemas de acceso a memoria no uniformes (NUMA).

Apoyo añadido al profesor: Como se mencionó anteriormente, el libro proporciona ahora una extensa ayuda para proyectos, ayuda suministrada por la página Web del libro, que también se ha ampliado.

MIENTOS

Nueva edición se ha beneficiado de la revisión de una serie de personas, que han aportado numerosamente su tiempo y conocimientos. Las siguientes personas revisaron la segunda edición y hicieron muchas sugerencias útiles: Yew Pen-Chung, de la Universidad de Minnesota; Yuval Tamir, de UCLA; Arthur Werbner y Bina Ramamurthy, de SUNY, Buffalo; Kitiles, de la Universidad de Minnesota; y Marcus Goncalves, de «Automation Research». David Lambert, de Intel, revisó el material sobre Pentium. Las siguientes personas fueron partes del manuscrito de la quinta edición: Jay Kubicky y Mike Albaugh, de Juegari; Tom Callaway, de Silicon Graphics; James Stine, de la Universidad de Lehigh; Gados Reis, de la «Ecole Normale Supérieure de Cachan»; y Rick Thomas, de Rutgers. Leppla, de IBM Alemania, me ayudó en la comprensión de la estrategia SMP del gran fabricador IBM. El profesor Cindy Norris, de la Universidad de Appalachian State, contribuyó algunos ejercicios.

Prólogo a la edición en español

El estudio de la estructura y arquitectura de los computadores se incluye en diversos currículos de ingeniería y ciencias. No abundan los buenos textos, como el presente, que cubran los programas correspondientes de forma amplia y rigurosa.

La elaboración de un texto de las características indicadas (al igual que sucede con otros libros de ingeniería) es de gran complejidad, dado que el autor debe realizar un laborioso trabajo de generalización de las diversas técnicas utilizadas en computadores concretos, y no sólo debe limitarse a recopilar información detallada sobre ellas. El texto debe presentar al lector abstracciones de equipos reales, de forma que le capaciten no sólo a entender los computadores actuales sino también los futuros, cuando éstos vean la luz. Este concepto es especialmente relevante en un área tan cambiante y en explosión como es la de los computadores. Considero que ésta es una de las principales cualidades del libro de Stallings, donde se da mayor relevancia a los conceptos que a la información (siempre en evolución). En casi todos los capítulos el autor utiliza este enfoque: primero presenta los conceptos clave, y luego los aplica a procesadores concretos. En la presente edición utiliza fundamentalmente las familias de procesadores Pentium y PowerPC, que prácticamente cubren la mayor parte de las tendencias de diseño de los computadores actuales (CISC y RISC, respectivamente), sin que por ello olvide describir ideas relevantes introducidas o usadas en otros procesadores (UltraSparc II, MIPS R10000, IA64, etc.).

También es destacable, como corresponde a un buen libro de ingeniería, la búsqueda que en todo momento hace el autor del análisis de prestaciones, y la presentación (dentro de este contexto) de técnicas específicas (fundamentalmente paralelismo) para equilibrar las prestaciones de los distintos elementos que pueden integrar un computador.

En la presente edición, además de las innovaciones indicadas, se ha efectuado una revisión completa de todo el material del libro, pudiendo destacar la actualización, o nueva introducción, de contenidos tales como memoria óptica, diseño superescalar, repertorio de instrucciones multimedia, ejecución anticipada y carga especulativa, sistemas SMP, clusters, y sistemas NUMA. El libro es complementado con una página web (<http://www.prenhall.com/stallings>) que contiene abundante ayuda tanto para los lectores como para los profesores de la materia.

Esta edición del libro en español contiene además, como valor añadido, un Apéndice C dedicado al microprocesador Pentium III, que hemos realizado profesores del departamento de arquitectura y tecnología de computadores de la universidad de Granada y que no aparece en la versión original del libro en inglés. Con este apéndice se trata de completar más aún

Un computador, como cualquier sistema, se compone de un conjunto de componentes interrelacionados. El sistema se caracteriza mejor en términos de estructura (cómo están interconectados los componentes) y de función (el funcionamiento de cada componente). Además, la organización de un computador es jerárquica. Cada componente principal puede ser descrito descomponiéndolo en sus subcomponentes principales y describiendo su estructura y funcionamiento. Esta organización jerárquica, para garantizar la claridad y facilidad de comprensión, se describe en este libro en orden descendente:

- **Computador:** los componentes principales son: el procesador, la memoria y los módulos de E/S.
- **Procesador:** los componentes principales son: la unidad de control, los registros, la ALU y la unidad de ejecución de instrucciones.
- **Unidad de control:** los componentes principales son: la memoria de control, la lógica de secuenciación de microinstrucciones y los registros.

El objetivo es presentar el material de forma que los conceptos nuevos queden en un contexto claro. Esto debería minimizar el riesgo de que el lector se pierda, y motivarle más que con una aproximación ascendente.

A través de esta discusión se ven los aspectos del sistema, tanto desde el punto de vista de la arquitectura (los atributos de un sistema visibles al programador de lenguaje máquina), como de la organización (las unidades funcionales y sus interconexiones, que dan lugar a la arquitectura).

SISTEMAS TOMADOS COMO EJEMPLO

A lo largo de este libro se usan ejemplos de docenas de máquinas diferentes para clarificar y enfatizar los ejemplos que se presentan. Muchos, pero no todos, de los ejemplos que se usan, se obtienen de dos familias de computadores: el Pentium II de Intel y el PowerPC. (El recientemente introducido Pentium III es esencialmente igual al Pentium II, añadiéndole un conjunto de instrucciones multimedia.) Estos dos tipos de computadores abarcan la mayor parte de las tendencias de diseño de los computadores actuales. El Pentium II es esencialmente un computador con un repertorio de instrucciones complejo (CISC), mientras que el PowerPC es esencialmente un computador con un repertorio reducido de instrucciones (RISC). Ambos sistemas hacen uso de principios de diseño superescalares, y ambos soportan configuraciones multiprocesador.

ORGANIZACIÓN DEL TEXTO

El libro está organizado en cinco partes:

Parte I. Visión general: esta parte contiene una introducción y marca el contexto para el resto del libro.

Parte II. El computador: un computador está compuesto por un procesador, una memoria y módulos de E/S, así como por las interconexiones entre estos componentes principales. En esta parte se examinan cada uno de estos aspectos, a excepción del procesador, que es lo bastante complejo como para ser tratado en la parte III.

Parte III. La unidad de procesamiento central: la CPU contiene una unidad de control, registros, la unidad aritmético lógica, la unidad de ejecución de instrucciones y las interconexiones entre estos componentes. Esta parte cubre materias arquitectónicas, como el

diseño del repertorio de instrucciones y los tipos de datos. También se presta atención a materias de organización, como la segmentación de cauce.

Parte IV. La unidad de control: es la parte del procesador que activa sus distintos componentes. Esta parte trata del funcionamiento de la unidad de control y su implementación usando microprogramación.

Parte V. Organización paralela: esta última parte trata de materias implicadas en la organización de multiprocesadores y procesadores vectoriales.

Al final del Capítulo 1 aparece un resumen más detallado, capítulo a capítulo.

SERVICIOS DE INTERNET PARA PROFESORES Y ESTUDIANTES

Existe una página Web de este libro que proporciona ayuda a estudiantes y profesores. Esta página incluye enlaces a otros sitios importantes, transparencias originales de figuras del libro en formato PDF (Adobe Acrobat), e información de la lista de correo de Internet del libro. La página Web está en <http://www.prenhall.com/stallings>; para más información, véase la Sección «Sitio Web del libro», que precede a este Prólogo. Se ha creado una lista de correo en Internet para que los profesores que utilicen este libro puedan intercambiar información, sugerencias y preguntas entre ellos y con el autor. Estará disponible una lista de erratas del libro, tan pronto como se vaya descubriendo cualquier tipo de error, en la dirección: <http://www.prenhall.com/stallings>.

PROYECTOS PARA LA ENSEÑANZA DE LA ARQUITECTURA Y ORGANIZACIÓN DE COMPUTADORES

Para muchos profesores, un componente importante de un curso de arquitectura y organización de computadores es un proyecto o conjuntos de proyectos con los que el estudiante vaya adquiriendo experiencia práctica para reforzar los conceptos del texto. Este libro proporciona un grado incomparable de apoyo, ya que incluye una sección de proyectos en el curso. El manual del profesor no sólo incluye una guía de cómo asignar y estructurar los proyectos, sino también un conjunto de proyectos propuestos que abarcan un amplio rango de la materia de este texto:

- **Proyectos de investigación:** El manual incluye una serie de tareas de investigación que instruyen al estudiante en la búsqueda de una materia concreta en la Web, en la bibliografía o en la escritura de informes.
- **Proyectos de simulación:** El manual proporciona apoyo para el uso del paquete de simulación SimpleScalar, que se puede utilizar para explorar las cuestiones de diseño en la organización y arquitectura de computadores.
- **Tareas de lectura/redacción:** El manual incluye una lista de artículos, uno o más por capítulo, que se pueden asignar a un alumno para que los lea y redacte un informe corto.

Véase el Apéndice B para más detalles.

NOVEDADES DE LA QUINTA EDICIÓN

En los cuatro años que han pasado desde la publicación de la cuarta edición de este libro han surgido continuas innovaciones y mejoras en este campo. En esta nueva edición he trata-

el texto con el estudio de procesadores o técnicas de última hora. Este apéndice será actualizado conforme se vayan realizando reimpresiones del presente libro, sin necesidad de esperar a ediciones nuevas.

Deseo destacar el esmerado trabajo de los traductores y la profesionalidad de Andrés Otero, editor de Informática del grupo editorial Pearson Educación, S. A. y responsable de la edición en español.

*Alberto Prieto
Coordinador de la traducción
Granada, 1 de mayo de 2000*

APÍTULO 1

Introducción

1.1. Organización y arquitectura

1.2. Estructura y funcionamiento

Fucionamiento

Estructura

1.3. Esquema del libro

Evolución y prestaciones de los computadores

Buses del sistema

Memoria interna

Memoria externa

Entrada/salida

Apoyo al sistema operativo

Aritmética de computadores

Repertorios de instrucciones

Estructura y funcionamiento de la CPU

Computadores de repertorio reducido de instrucciones

Procesadores superescalares y paralelismo a nivel de instrucción

Funcionamiento de la unidad de control

Control microprogramado

Procesamiento paralelo

Lógica digital

1.4. Internet y recursos Web

Sitios Web de este libro

Otros sitios Web

Grupos de noticias USENET

PARTE I

VISIÓN GENERAL

CUESTIONES A TRATAR EN LA PRIMERA PARTE

El objetivo de la Parte I es proporcionar una base y un contexto para el resto del libro. Se presentan los conceptos fundamentales sobre arquitectura y organización de computadores.

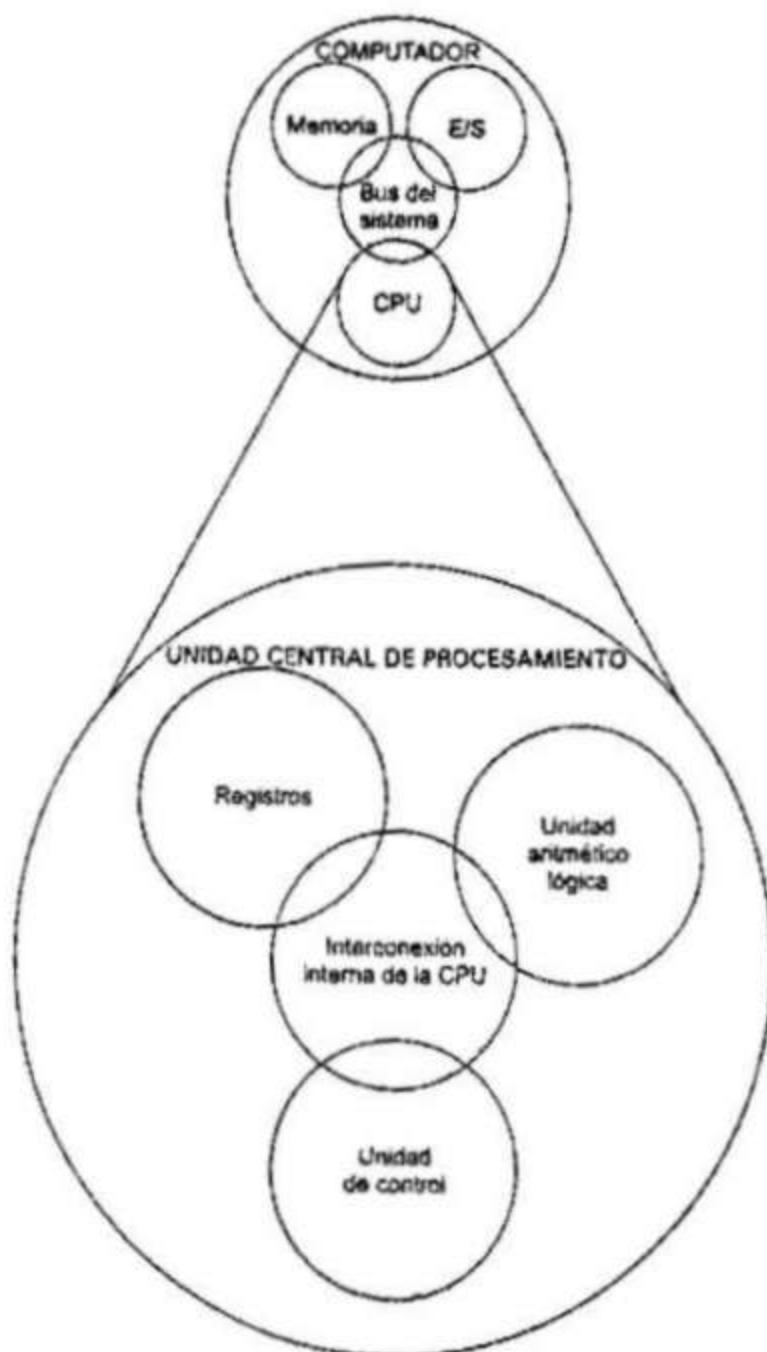
ESQUEMA DE LA PARTE I

CAPÍTULO 1. INTRODUCCIÓN

El Capítulo 1 introduce el concepto de computador como sistema jerárquico. Un computador puede ser visto como una estructura de componentes, y su funcionamiento puede ser descrito en términos del funcionamiento colectivo de sus componentes cooperantes. Cada componente puede ser descrito, a su vez, según su estructura interna y funcionamiento. Se introducen los niveles principales de esta visión jerárquica. El resto del libro está organizado comenzando por el nivel superior y bajando hasta los inferiores, según estos niveles.

CAPÍTULO 2. EVOLUCIÓN Y PRESTACIONES DE LOS COMPUTADORES

El Capítulo 2 contiene una breve historia del desarrollo de los computadores, desde sus antepasados mecánicos a los sistemas de hoy en día. Esta historia sirve para destacar algunos aspectos importantes del diseño de computadores, y para dar una visión de alto nivel de la estructura de un computador. El capítulo presenta luego un tema clave de este libro: diseño para prestaciones. También se señala la importancia de lograr un equilibrio en la utilización de componentes con características de funcionamiento muy diferentes.



Los elementos principales de un computador son: la unidad de procesamiento central (CPU), la memoria principal, el subsistema de entrada/salida y algunos medios de interconexión de todos estos componentes. La CPU, por su parte, consta de una unidad de control, una unidad aritmético-lógica (ALU, Arithmetic and Logic Unit), registros internos e interconexiones.

Este libro se complementa con una página Web que proporciona enlaces a otras páginas Web relevantes y a otras informaciones útiles. Véase la Sección sobre la página Web al principio del libro para más detalles.

Este libro trata sobre la estructura y funcionamiento de los computadores. Su objetivo es presentar, tan clara y completamente como sea posible, la naturaleza y las características de los computadores de hoy día. Este objetivo es todo un reto por dos razones. En primer lugar, hay una gran variedad de sistemas que pueden recibir correctamente el nombre de *computador*, desde microprocesadores de un solo chip, que cuestan unos pocos dólares, a supercomputadores que cuestan decenas de millones de dólares. Esta variedad es patente no sólo en costes, sino también en tamaño, prestaciones y aplicaciones. En segundo lugar, la rapidez de los cambios, que ha caracterizado siempre a la tecnología de computadores, continúa sin pausa. Estos cambios cubren todos los aspectos de la tecnología de computadores, desde la tecnología subyacente de circuitos integrados, usados para construir componentes de computadores, hasta el creciente uso de conceptos de organización paralela para combinar esos componentes.

A pesar de la variedad y rapidez de los cambios en el campo de los computadores, se aplican sistemáticamente ciertos conceptos fundamentales. La aplicación de estos conceptos depende del desarrollo actual de la tecnología y de los objetivos, en cuanto a precio y aplicaciones, del diseñador. La intención de este libro es ofrecer un concienzudo análisis de los fundamentos de la arquitectura y organización de los computadores y relacionar éstos con materias de diseño actuales. Este capítulo introductorio analiza brevemente el enfoque descriptivo que se va a considerar y ofrece una visión global del resto del libro.

ORGANIZACIÓN Y ARQUITECTURA

Cuando se describe un computador, frecuentemente se distingue entre *arquitectura* y *organización del computador*. Aunque es difícil dar una definición precisa para estos términos, existe un consenso sobre las áreas generales cubiertas por cada uno (véase [VRAN80], [SIEW82] y [BELL78a]).

La arquitectura de computadores se refiere a los atributos de un sistema que son visibles para un programador, o, para decirlo de otra manera, a aquellos atributos que tienen un impacto directo en la ejecución lógica de un programa. La organización de computadores se refiere a las unidades funcionales y sus interconexiones, que dan lugar a especificaciones arquitectónicas. Entre los ejemplos de atributos arquitectónicos se encuentran el conjunto de instrucciones, el número de bits usados para representar varios tipos de datos (por ejemplo, números y caracteres), los mecanismos de E/S y las técnicas para direccionamiento de memoria. Entre los atributos de organización se incluyen aquellos detalles de hardware transparentes al programador, tales como señales de control, interfaces entre el computador y los periféricos y la tecnología de memoria usada.

Para poner un ejemplo, una cuestión de diseño arquitectónico es si el computador tendrá la instrucción de multiplicar. Una cuestión de organización es si esa instrucción será implementada por una unidad especializada en multiplicar, o por un mecanismo que haga un uso iterativo de la unidad de suma del sistema. La decisión de organización puede estar basada en la frecuencia prevista del uso de la instrucción de multiplicar, la velocidad relativa de las dos aproximaciones, y el coste y el tamaño físico de una unidad especializada en multiplicar.

Históricamente, y aún hoy día, la distinción entre arquitectura y organización ha sido importante. Muchos fabricantes de computadores ofrecen una familia de modelos, todos con la misma arquitectura, pero con diferencias en cuanto a la organización. Consecuentemente, los diferentes modelos de la familia tienen precios y prestaciones distintas. Más aún, una arquitectura puede sobrevivir muchos años, pero su organización cambia con la evolución de la tecnología. Un ejemplo destacado de ambos fenómenos es la arquitectura IBM Sistema/370.

Esta arquitectura apareció por primera vez en 1970 e incluía varios modelos. Un cliente con necesidades modestas podía comprar un modelo más barato y lento, y si la demanda se incrementaba, cambiarse más tarde a un modelo más caro y rápido sin tener que abandonar el software que ya había sido desarrollado. A través de los años, IBM ha introducido muchos modelos nuevos con tecnología mejorada para reemplazar a modelos más viejos, ofreciendo al consumidor mayor velocidad, precios más bajos o ambas cosas a la vez. Estos modelos más nuevos conservaban la misma arquitectura, para proteger así la inversión en software del consumidor. Podemos destacar que la arquitectura del Sistema/370, con unas pocas mejoras, ha sobrevivido hasta hoy día como la arquitectura de la línea de grandes productos de computación IBM.

En una clase de sistemas, los llamados microcomputadores, la relación entre arquitectura y organización es muy estrecha. Los cambios en la tecnología no sólo influyen en la organización, sino que también dan lugar a la introducción de arquitecturas más ricas y potentes. Generalmente, hay menos requisitos de compatibilidad, generación a generación, para estas pequeñas máquinas. Así, hay más interacción entre las decisiones de diseño arquitectónicas y de organización. Un ejemplo interesante de esto son los computadores de repertorio reducido de instrucciones (RISC, Reduced Instruction Set Computer), que veremos en el Capítulo 12.

En este libro se examina, tanto la organización, como la arquitectura de un computador. Se da, quizás, más énfasis a la parte de organización. Sin embargo, como la organización de un computador debe ser diseñada para implementar una especificación de una arquitectura particular, un estudio exhaustivo de la organización requiere también un examen detallado de la arquitectura.

ESTRUCTURA Y FUNCIONAMIENTO

Un computador es un sistema complejo; los computadores de hoy en día contienen millones de componentes electrónicos básicos. ¿Cómo podríamos describirlos claramente? La clave está en reconocer la naturaleza jerárquica de la mayoría de los sistemas complejos, incluyendo el computador [SIMO69]. Un sistema jerárquico es un conjunto de subsistemas interrelacionados, cada uno de los cuales, a su vez, se organiza en una estructura jerárquica, hasta que se alcanza el nivel más bajo del subsistema elemental.

La naturaleza jerárquica de los sistemas complejos es esencial, tanto para su diseño, como para su descripción. El diseñador necesita tratar sólamente con un nivel particular del sistema a la vez. En cada nivel, el sistema consta de un conjunto de componentes y sus interrelaciones. El comportamiento en cada nivel depende sólo de una caracterización abstracta y simplificada del sistema que hay en el siguiente nivel más bajo. De cada nivel, al diseñador le importa la estructura y el funcionamiento [KOES78]:

- Estructura: el modo en que los componentes están interrelacionados.
- Funcionamiento: la operación de cada componente individual como parte de la estructura.

En términos de descripción tenemos dos opciones: empezar por lo más bajo y construir una descripción completa, o comenzar con una visión desde arriba y descomponer el sistema en sus subpartes. La experiencia a partir de muchos campos nos ha enseñado que la descripción de arriba a abajo (*«top-down»*) es la más clara y efectiva [WEIN75].

El enfoque seguido en este libro considera este punto de vista. El computador será descrito de arriba a abajo. Comenzamos con los componentes principales del sistema, describiendo su estructura y funcionamiento, y seguimos, sucesivamente, hacia capas más bajas de la jerarquía. Lo que queda de esta sección ofrece una breve visión global de este plan de ataque.

FUNCIONAMIENTO

Tanto la estructura como el funcionamiento de un computador son, en esencia, sencillos. La Figura 1.1 señala las funciones básicas que un computador puede llevar a cabo. En términos generales hay sólo cuatro:

- Procesamiento de datos
- Almacenamiento de datos
- Transferencias de datos
- Control

El computador, por supuesto, tiene que ser capaz de *procesar datos*. Los datos pueden adoptar una gran variedad de formas, y el rango de los requisitos de procesado es amplio. Sin embargo, veremos que hay sólo unos pocos métodos o tipos fundamentales de procesado de datos.

También es esencial que un computador *almacene datos*. Incluso si el computador está procesando datos al vuelo (es decir, si los datos se introducen y se procesan, y los resultados se obtienen inmediatamente), el computador tiene que guardar temporalmente, al menos aquellos datos con los que está trabajando en un momento dado. Así, hay al menos una función de almacenamiento de datos a corto plazo. Con igual importancia, el computador lleva a cabo una función de almacenamiento de datos a largo plazo. El computador almacena ficheros de datos para que se recuperen y actualicen en un futuro.

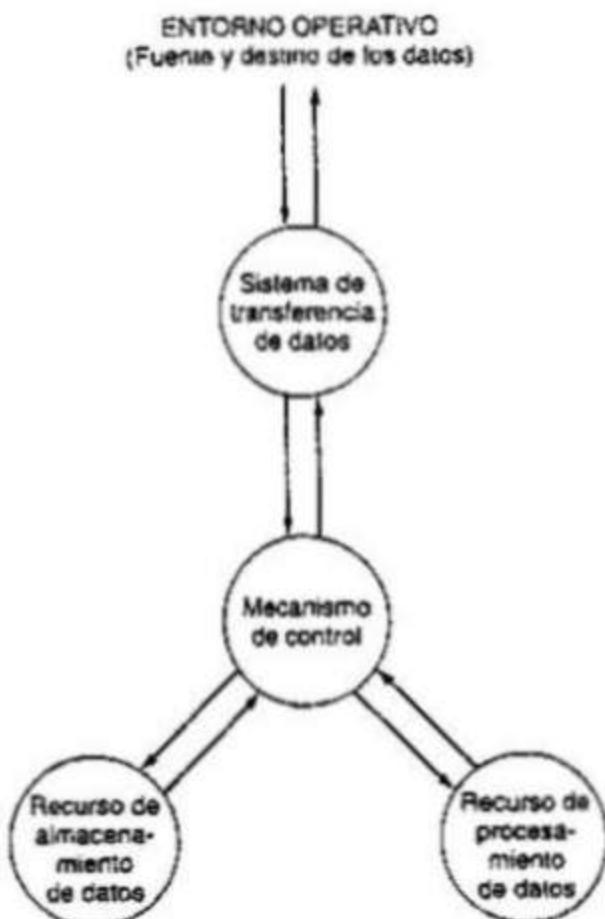


Figura 1.1. Una visión funcional de un computador.

El computador tiene que ser capaz de *transferir datos* entre él mismo y el mundo exterior. El entorno de operación del computador se compone de dispositivos que sirven bien como fuente o como destino de datos. Cuando se reciben o se llevan datos a un dispositivo que está directamente conectado con el computador, el proceso se conoce como *entrada-salida* (E/S), y este dispositivo recibe el nombre de *periférico*. El proceso de transferir datos a largas distancias, desde o hacia un dispositivo remoto, recibe el nombre de *comunicación de datos*.

Finalmente, debe haber un *control* de estas tres funciones. Este control es ejercido por el/los ente(s) que proporciona(n) al computador instrucciones. Dentro del computador, una unidad de control gestiona los recursos del computador y dirige las prestaciones de sus partes funcionales en respuesta a estas instrucciones.

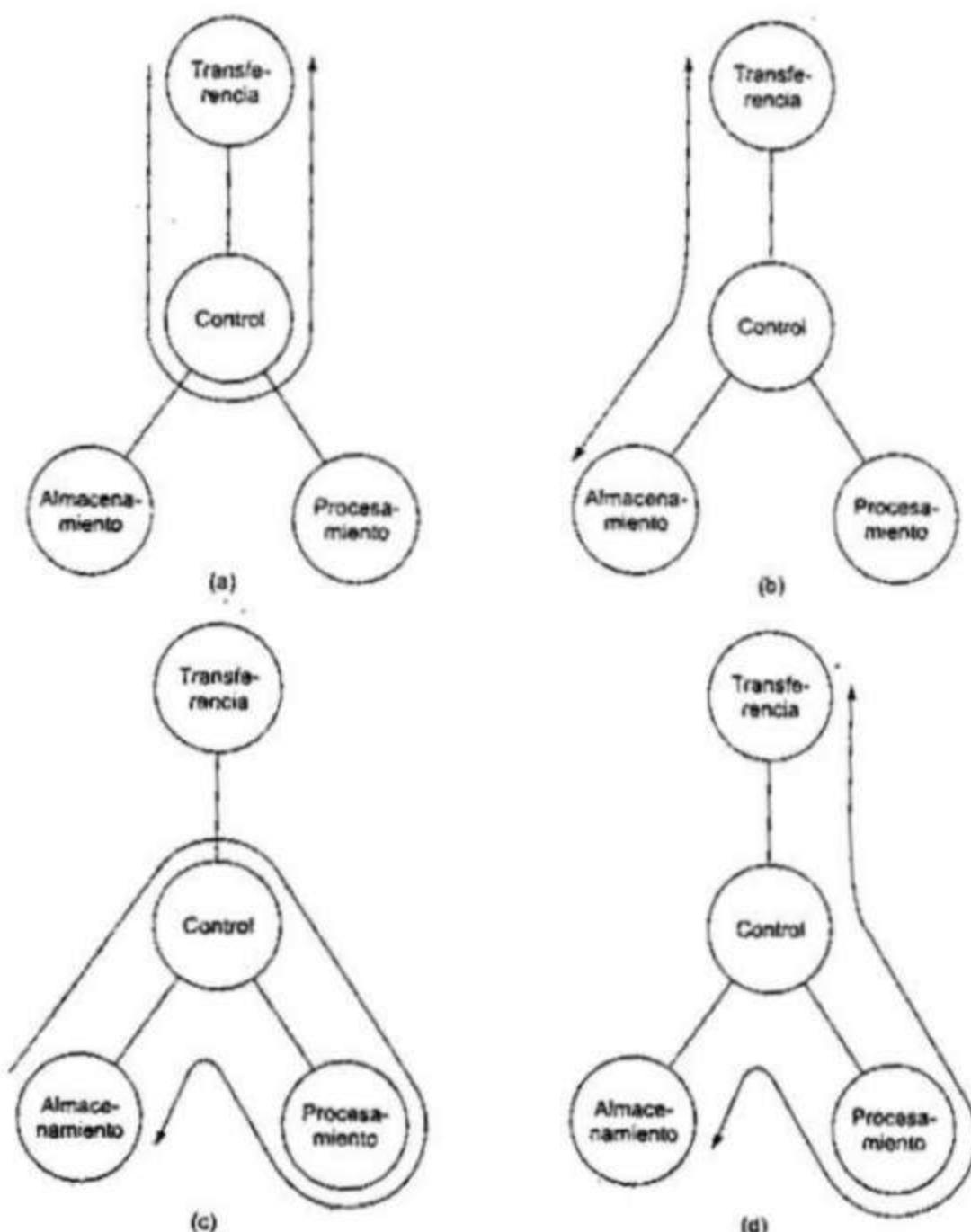


Figura 1.2. Posibles operaciones de un computador.

A este nivel general de discusión, el número de operaciones posibles que pueden ser realizadas es pequeño. La Figura 1.2 muestra los cuatro posibles tipos de operaciones. El computador puede funcionar como un dispositivo de transferencia de datos (Figura 1.2a), simplemente transfiriendo datos de un periférico o línea de comunicaciones a otro. También puede funcionar como un dispositivo de almacenamiento de datos (Figura 1.2b), con datos transferidos desde un entorno externo al almacén de datos del computador (leer) y viceversa (escribir). Los dos diagramas siguientes muestran operaciones que implican procesamiento de datos en datos almacenados (Figura 1.2c), o bien en tránsito entre el almacén y el entorno externo.

La exposición precedente puede parecer absurdamente generalizada. Es posible, incluso en el nivel más alto de la estructura de un computador, diferenciar varias funciones, pero, citando [SIEW82]:

Hay, sorprendentemente, muy pocas formas de estructuras de computadores que se ajusten a la función que va a ser llevada a cabo. En la raíz de esto subyace el problema de la naturaleza de uso general de los computadores, en la cual toda la especialización funcional ocurre cuando se programa y no cuando se diseña.

ESTRUCTURA

La Figura 1.3 es la representación más sencilla posible de un computador. El computador es una entidad que interactúa de alguna manera con su entorno externo. En general, todas sus conexiones con el entorno externo pueden ser clasificadas como dispositivos periféricos o líneas de comunicación. Diremos algo sobre ambos tipos de conexiones.

Pero es de mayor interés en este libro la estructura interna del computador mismo, la cual mostramos, en su nivel más alto, en la Figura 1.4. Hay cuatro componentes estructurales principales:

Unidad central de procesamiento (CPU, Central Processing Unit): controla el funcionamiento del computador y lleva a cabo sus funciones de procesamiento de datos. Frecuentemente se le llama simplemente *procesador*.

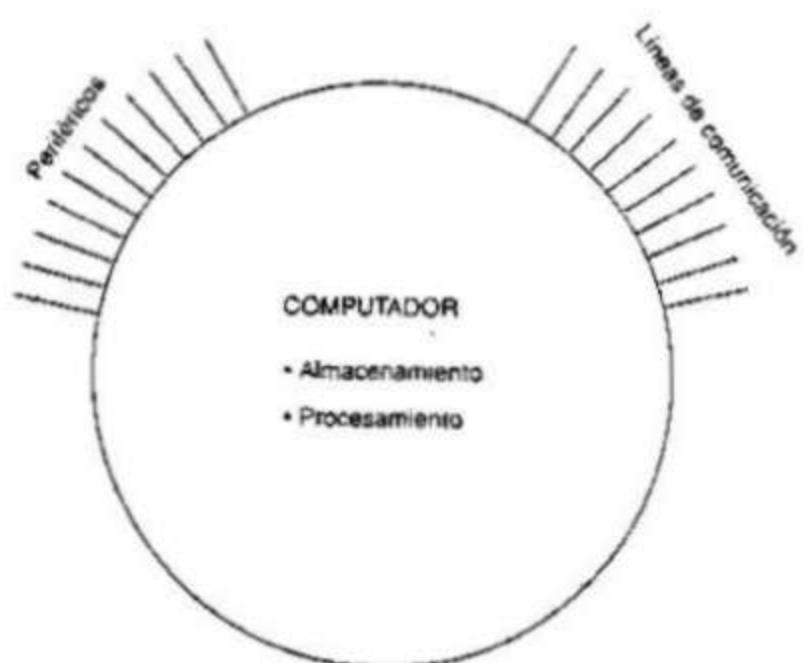


Figura 1.3. El computador.

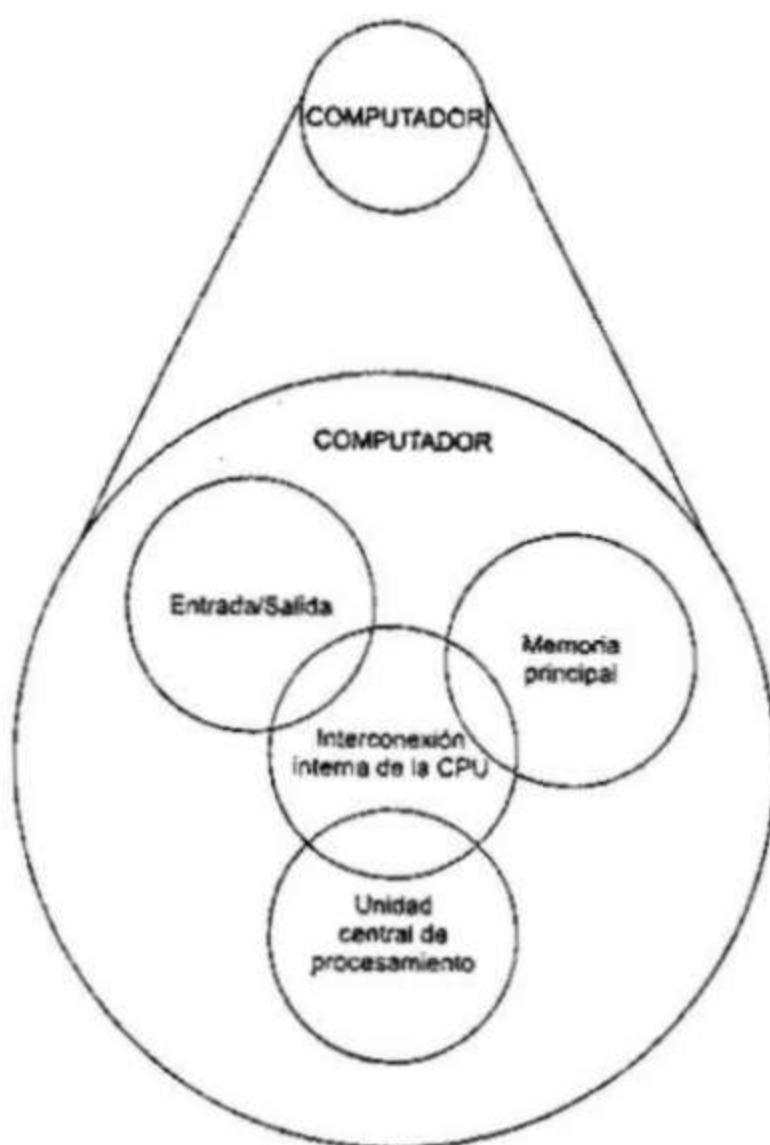


Figura 1.4. El computador: estructura del nivel superior.

- **Memoria principal:** almacena datos.
- **E/S:** transfiere datos entre el computador y el entorno externo.
- **Sistema de interconexión:** es un mecanismo que proporciona la comunicación entre la CPU, la memoria principal y la E/S.

Puede que haya uno o más de cada uno de estos componentes. Tradicionalmente, ha habido sólo una CPU. En los últimos años, ha habido un uso creciente de varios procesadores en un solo sistema. Surgen algunas cuestiones relativas a multiprocesadores y se discuten conforme el texto avanza; el Capítulo 16 se centra en tales sistemas.

Cada uno de estos componentes será examinado con cierto detalle en la Parte II. Sin embargo, para nuestros objetivos, el componente más interesante y, de algún modo, el más complejo es la CPU; su estructura se muestra en la Figura 1.5. Sus principales componentes estructurales son:

- **Unidad de control:** controla el funcionamiento de la CPU y, por tanto, del computador.
- **Unidad aritmético-lógica (ALU, Arithmetic Logic Unit):** lleva a cabo las funciones de procesamiento de datos del computador.

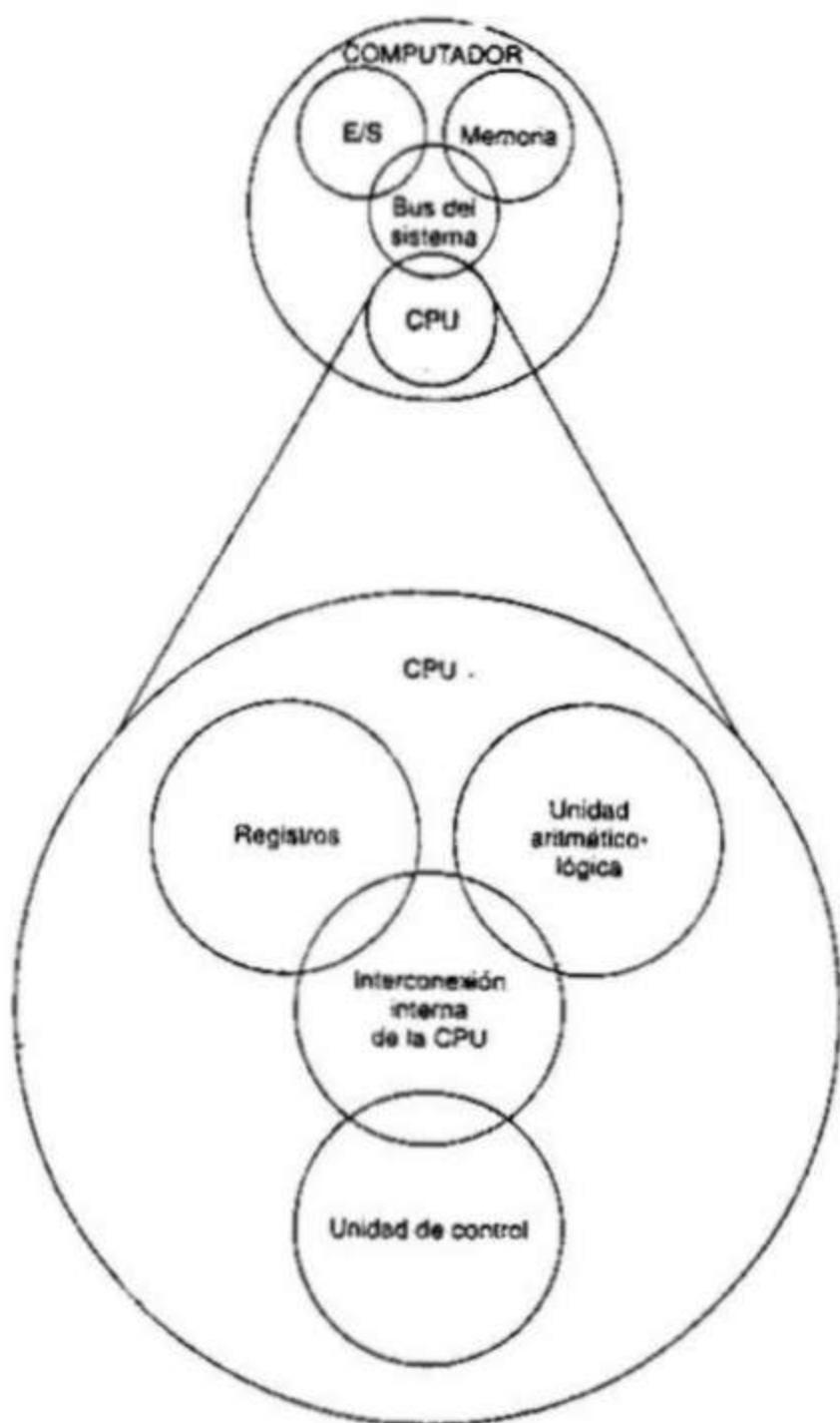


Figura 1.5. La unidad central de procesamiento (CPU).

- **Registros:** proporcionan almacenamiento interno a la CPU.
- **Interconexiones CPU:** son mecanismos que proporcionan comunicación entre la unidad de control, la ALU y los registros.

Cada uno de estos componentes será examinado con detalle en la Parte III, donde veremos que la complejidad aumenta con el uso de técnicas de organización paralelas y de segmentación de cauce. Finalmente, hay varias aproximaciones para la implementación de la unidad de control, pero sin duda la más común es la implementación *microprogramada*. Con esta aproximación, la estructura de la unidad de control puede ser como la mostrada en la Figura 1.6. Esta estructura será examinada en la Parte IV.

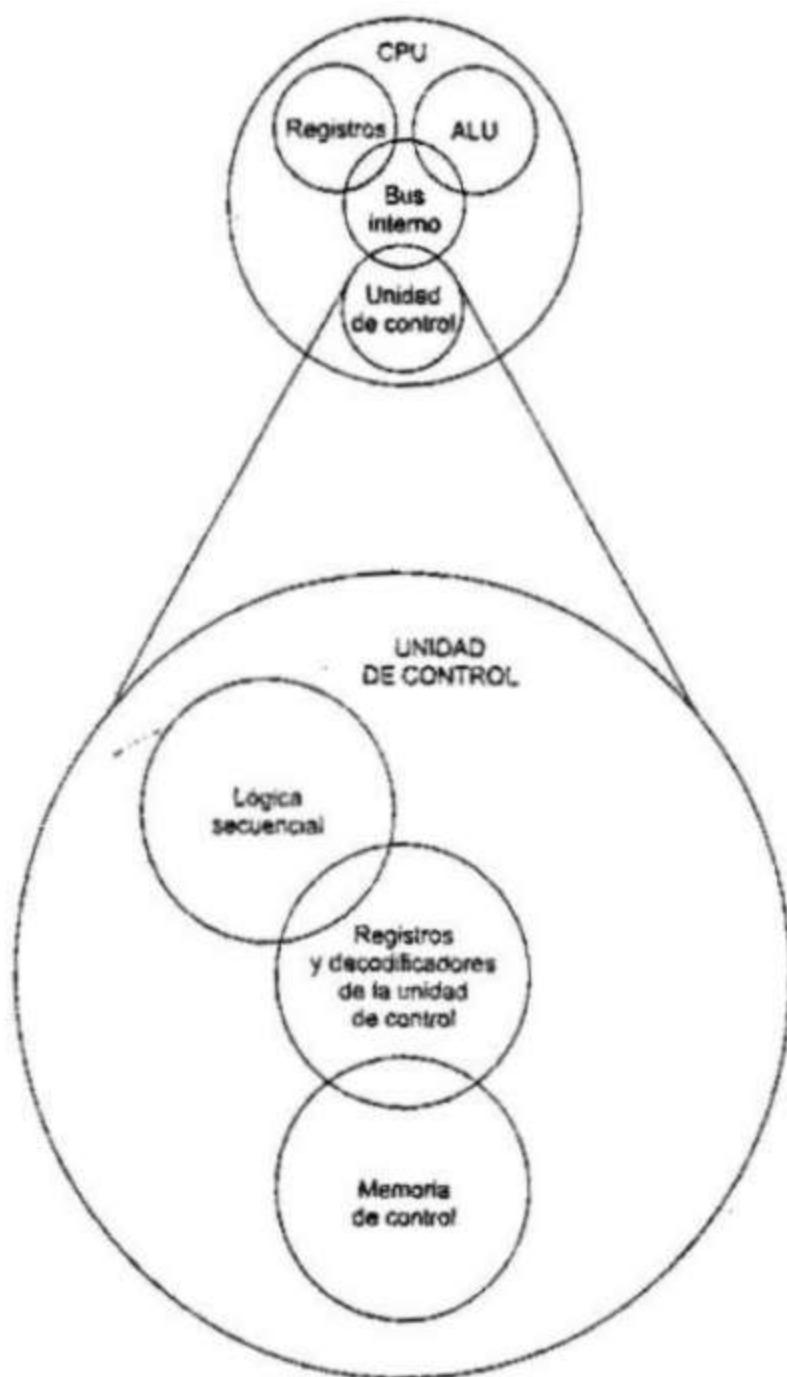


Figura 1.6. La unidad de control.

ESQUEMA DEL LIBRO

Este capítulo sirve como introducción al libro entero. Aquí se ofrece una breve sinopsis del resto de los capítulos.

EVOLUCIÓN Y PRESTACIONES DEL COMPUTADOR

El Capítulo 2 tiene dos objetivos. El primero es hacer una exposición de la historia de la tecnología de computadores, como un modo fácil e interesante de familiarizarnos con los

conceptos básicos de organización y arquitectura de computadores. El capítulo también muestra las tendencias tecnológicas que han hecho de las prestaciones el centro del diseño de computadores y da una primera visión de las distintas técnicas y estrategias que se usan para lograr unas prestaciones equilibradas y eficientes.

BUSES DEL SISTEMA

En su nivel superior, un computador se compone de procesador, memoria y componentes de E/S. El funcionamiento del sistema consiste en el intercambio de datos y señales de control entre estos componentes. Para realizar este intercambio, estos componentes deben estar interconectados. El Capítulo 3 comienza con un breve examen de los componentes del computador y sus requisitos de entrada/salida. El capítulo trata luego de temas clave, que afectan al diseño de la interconexión, especialmente la necesidad de soportar interrupciones. El grueso del capítulo se dedica al estudio del tipo de interconexiones más común: la estructura de buses.

MEMORIA INTERNA

En memoria de computadores hay una amplia gama de tipos, tecnologías, organizaciones, prestaciones y precios. El computador típico está equipado con una jerarquía de subsistemas de memoria, algunos de ellos internos (directamente accesibles por el procesador) y otros externos (accesibles por el procesador vía un módulo de E/S). El Capítulo 4 comienza con una visión general de esta jerarquía, centrándose luego en materias de diseño relacionadas con la memoria interna. Primero, se examina la naturaleza y organización de la memoria principal semiconductora. Más tarde, el capítulo trata con detalle del diseño de la memoria caché, incluyendo cachés de datos y de códigos separadamente, y cachés de dos niveles. Finalmente, se exploran organizaciones de memoria DRAM, recientes y avanzadas.

MEMORIA EXTERNA

En el Capítulo 5 se examinan los diferentes parámetros de diseño y prestaciones asociados a la memoria de disco. Luego, los esquemas RAID, que se están volviendo cada vez más comunes. Finalmente, se examinan los sistemas de memoria óptica y de cinta magnética.

ENTRADA/SALIDA

Los módulos de E/S están conectados con el procesador y la memoria principal, y cada uno controla uno o más dispositivos externos. En el Capítulo 6 se examinan los mecanismos por los que un módulo de E/S interactúa con el resto del computador, usando las técnicas de E/S programada, E/S por interrupción y acceso directo a memoria (DMA, Direct Memory Access). También se describe la interfaz entre un módulo de E/S y los dispositivos externos.

APOYO AL SISTEMA OPERATIVO

En este momento es apropiado estudiar el sistema operativo, para explicar cómo los componentes básicos del computador son gestionados para llevar a cabo trabajos útiles, y cómo el hardware del computador se organiza para dar apoyo al sistema operativo. El Capítulo 7 comienza con una breve historia, que sirve para identificar los tipos principales de sistemas

operativos y para motivar las razones de su uso. Luego, se explica la multiprogramación, examinando las funciones de planificación a corto y a largo plazo. Finalmente, un examen de la gestión de memoria incluye un análisis de la segmentación, paginación y memoria virtual.

ARITMÉTICA DE COMPUTADORES

El Capítulo 8 comienza con un análisis detallado del procesador y un estudio de la aritmética de los computadores. Los procesadores, tradicionalmente, implementan dos tipos de aritmética: entera o de punto fijo, y de punto flotante. Para ambos casos, el capítulo examina primero la representación de los números, y discute luego las operaciones aritméticas. Se estudia con detalle el importante estándar de punto flotante IEEE 754.

REPERTORIOS DE INSTRUCCIONES

Desde el punto de vista del programador, el mejor modo para comprender el funcionamiento de un procesador es aprender el conjunto de instrucciones máquina que se ejecutan. El Capítulo 9 examina las características clave de los repertorios de instrucciones máquina. Cubre varios tipos de datos y de operaciones que se encuentran normalmente en un conjunto de instrucciones. Luego, se explica brevemente la relación entre las instrucciones del procesador y el lenguaje ensamblador. En el Capítulo 10 se examinan los posibles modos de direccionamiento. Finalmente, se trata el tema del formato de instrucción, incluyendo un examen de compromisos.

ESTRUCTURA Y FUNCIONAMIENTO DE LA CPU

El Capítulo 11 está dedicado a la discusión de la estructura interna y el funcionamiento del procesador. Se revisa la organización general (ALU, unidad de control y banco de registros). Luego, se discute la organización del banco de registros. El resto del capítulo describe el funcionamiento del procesador en la ejecución de instrucciones máquina. Se examina el ciclo de instrucción, para mostrar la interrelación de los ciclos de captación de instrucciones, ciclos indirectos, ciclos de ejecución y ciclos de interrupción. Por último, se explora en profundidad el uso de segmentación de cauce para la mejora de las prestaciones.

COMPUTADORES DE REPERTORIO REDUCIDO DE INSTRUCCIONES

Una de las innovaciones más importantes en la organización y arquitectura de computadores en los últimos años, es el computador de conjunto reducido de instrucciones (RISC). La arquitectura RISC es una drástica desviación de las tendencias históricas en arquitectura de procesadores. Un análisis de esta aproximación saca a la luz muchos de los temas importantes en la organización y arquitectura de computadores. El Capítulo 12 presenta la aproximación RISC y la compara con la del computador de repertorio de instrucciones complejo (CISC).

PROCESADORES SUPERESCALARES Y PARALELISMO A NIVEL DE INSTRUCCIÓN

En el Capítulo 13 se examina una innovación aún más reciente e igualmente importante: el procesador superescalar. Aunque la tecnología superescalar puede ser usada en cualquier procesador, es especialmente adecuada en la arquitectura RISC. En el capítulo también se ven cuestiones generales sobre paralelismo a nivel de instrucción.

FUNCIONAMIENTO DE LA UNIDAD DE CONTROL

El Capítulo 14 vuelve al tema de cómo se llevan a cabo las funciones del procesador, o, más específicamente, cómo se controlan los distintos elementos del procesador para efectuar esas funciones por medio de la unidad de control. Se muestra que cada ciclo de instrucción está formado por un conjunto de microoperaciones que generan señales de control. La ejecución es llevada a cabo por el efecto de estas señales de control, que parten de la unidad de control hacia la ALU, los registros y la estructura de interconexión de los sistemas. Finalmente, se presenta una aproximación de la realización de la unidad de control, denominada «implementación cableada».

CONTROL MICROPROGRAMADO

En el Capítulo 15 se muestra cómo se puede implementar la unidad de control usando la técnica de la microprogramación. Primero, las microoperaciones se agrupan en microinstrucciones. Luego, se describe el formato de la memoria de control que contiene un microprograma para cada instrucción máquina. Entonces, podremos explicar la estructura y funcionamiento de la unidad de control microprogramada.

PROCESAMIENTO PARALELO

Tradicionalmente, el computador ha sido visto como una máquina secuencial. Conforme la tecnología de computadores evolucionaba y el coste del hardware caía, los diseñadores de computadores han empezado a buscar cada vez más oportunidades para el paralelismo, normalmente para mejorar las prestaciones, y en algunos casos para mejorar la fiabilidad. El Capítulo 16 considera una serie de aplicaciones de la organización paralela. Con respecto a los multiprocesadores, el capítulo examina el tema clave de la coherencia de caché.

LÓGICA DIGITAL

La mayor parte del texto trata los elementos de almacenamiento binario y las funciones digitales como bloques fundamentales que constituyen los computadores. En el Apéndice del texto, se describe cómo estos elementos de almacenamiento y de síntesis de funciones se pueden implementar con lógica digital. El Apéndice comienza con una breve revisión del álgebra de Boole. Luego, se introduce el concepto de puerta. Finalmente, se discuten los circuitos combinacionales y secuenciales que se pueden construir a partir de las puertas.

EN INTERNET Y RECURSOS WEB

Hay muchos recursos disponibles en Internet y en la Web que dan apoyo a este libro y que ayudan a seguir el desarrollo de este campo.

SITIOS WEB DE ESTE LIBRO

Se ha creado recientemente una Companion Web Site especial para este libro:

<http://www.prenhall.com/stallings>

Véase la Sección al principio del libro para tener una descripción más detallada de la página.

OTROS SITIOS WEB

Hay numerosos sitios Web que proporcionan algún tipo de información relacionada con la materia de este libro. En la sección de «Lecturas recomendadas» de cada capítulo, se pueden encontrar punteros a sitios Web específicos. Como los URL de las páginas Web tienden a cambiar frecuentemente, no los he incluido en este libro. En la página Web del libro, se puede encontrar el enlace adecuado de cada sitio Web mencionado en el libro.



Las siguientes páginas Web son direcciones de interés general relacionadas con la arquitectura y organización de computadores:

- **Página principal de WWW Computer Architecture:** índice exhaustivo de información relacionada con investigadores de arquitectura de computadores, incluyendo grupos de arquitectura y proyectos, organizaciones técnicas, bibliografía, empleo e información comercial.
- **CPU Info Center:** información sobre procesadores específicos, incluyendo artículos técnicos, información sobre productos y últimas novedades.
- **ACM Special Interest Group on Computer Architecture:** información sobre actividades y publicaciones SIGARCH.
- **IEEE Technical Committee on Computer Architecture:** copias de las cartas TCAA.
- **Intel Technology Journal:** publicaciones de Intel.

GRUPOS DE NOTICIAS USENET

Una serie de grupos de noticias USENET se dedica a algunos aspectos de la arquitectura y organización de computadores. Como con casi todos los grupos USENET, hay una relación señal ruido alta, pero merece la pena ver experimentalmente si alguno satisface sus necesidades. Los más importantes son los siguientes:

- **comp.arch:** grupo de noticias general para discutir sobre arquitectura de computadores. A menudo bastante bueno.
- **comp.arch.arithmetic:** tratan sobre algoritmos aritméticos de computadores y normalizaciones.
- **comp.arch.storage:** discuten desde productos hasta tecnología, en asuntos de uso práctico.

CAPÍTULO 2

Evolución y prestaciones de los computadores

2.1. Una breve historia de los computadores

La primera generación: los tubos de vacío

La segunda generación: los transistores

La tercera generación: los circuitos integrados

Últimas generaciones

2.2. Diseño para conseguir mejores prestaciones

Velocidad del microprocesador

Equilibrio de prestaciones

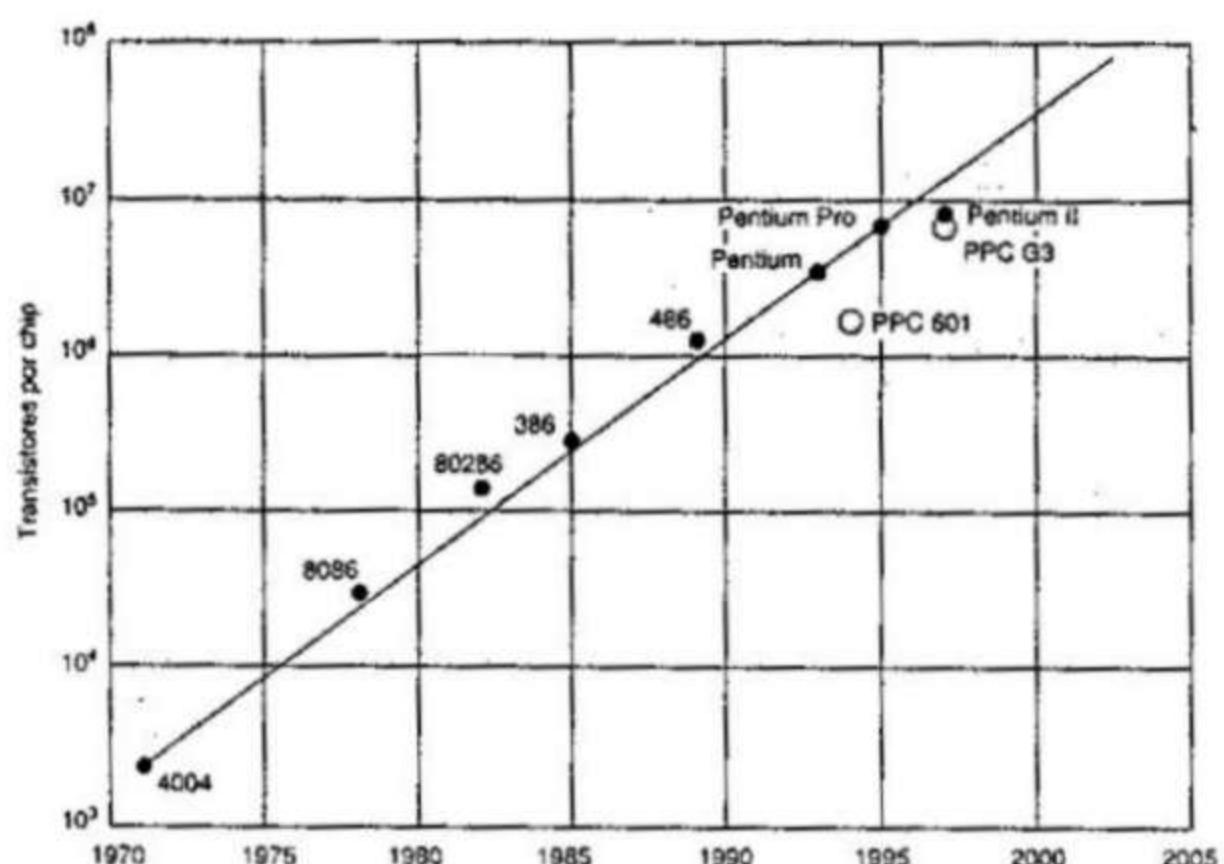
2.3. Evolución del Pentium y del PowerPC

Pentium

PowerPC

2.4. Lecturas y sitios Web recomendados

2.5. Problemas



- La evolución de los computadores se ha caracterizado por un incremento de la velocidad del procesador, una disminución del tamaño de los componentes, un aumento del tamaño de memoria y un aumento de la capacidad de E/S y de la velocidad.
- Otro factor responsable del gran aumento de la velocidad del procesador es la disminución del tamaño de los componentes del microprocesador; esto reduce la distancia entre componentes y, por tanto, aumenta la velocidad. Sin embargo, la verdadera ganancia en velocidad en los últimos años se debe a la organización del procesador, incluyendo un fuerte uso del encauzamiento y de las técnicas de ejecución paralela, y del uso de técnicas de ejecución especulativa, que conducen a la ejecución tentativa de instrucciones futuras que se puedan necesitar. Todas estas técnicas se diseñan para mantener al procesador ocupado el mayor tiempo posible.
- Otro asunto crítico en el diseño de computadores, es hacer un balance de las prestaciones de los distintos elementos, de forma que esta ganancia en prestaciones en un área no perjudique a otras áreas. En particular, la velocidad del procesador ha aumentado más rápidamente que el tiempo de acceso a memoria. Se han usado distintas técnicas para compensar este desacople, incluyendo memorias cache, caminos de datos más anchos de la memoria al procesador, y más circuitos de memoria inteligentes.

Empezamos nuestro estudio de los computadores con una breve historia. Esta historia es interesante por sí misma y, además, sirve para proporcionar una visión general de la estructura y funcionamiento de los computadores. Luego, se trata el tema de las prestaciones. Hay que considerar la necesidad de equilibrar los recursos de un computador, lo que nos da un contexto útil en todo el libro. Finalmente, veremos brevemente la evolución de dos sistemas que sirven como ejemplos claves en todo el libro: Pentium y PowerPC.

ZP. UNA BREVE HISTORIA DE LOS COMPUTADORES

LA PRIMERA GENERACIÓN: LOS TUBOS DE VACÍO

ENIAC

El ENIAC (Electronic Numerical Integrator And Computer), diseñado y construido bajo la supervisión de John Mauchly y John Presper Eckert en la Universidad de Pennsylvania, fue el primer computador electrónico de propósito general del mundo.

El proyecto fue una respuesta a necesidades militares en tiempo de guerra de los Estados Unidos. El Laboratorio de Investigación de Balística (BRL, Ballistics Research Laboratory) del Ejército, una agencia responsable del desarrollo de tablas de tiro y de trayectoria para nuevas armas, tenía dificultades para elaborarlas con exactitud y dentro de un plazo de tiempo razonable. Sin estas tablas de tiro, las nuevas armas y piezas de artillería eran inútiles para los artilleros. El BRL empleó a más de 200 personas, la mayoría mujeres, las cuales, utilizando calculadoras de mesa, resolvían las ecuaciones balísticas necesarias. La preparación de las tablas para una sola arma le habría llevado a una persona muchas horas, incluso días.

Mauchly, un catedrático de ingeniería eléctrica de la Universidad de Pennsylvania, y Eckert, uno de sus alumnos de licenciatura, propusieron construir un computador de propósito general, usando tubos de vacío, para utilizarlo en las aplicaciones de la BRL. En 1943, esta proposición fue aceptada por el Ejército, y se comenzó a trabajar en el ENIAC. La máquina que construyeron era enorme, pesaba 30 toneladas, ocupaba 15.000 pies cuadrados y contenía más de 18.000 tubos de vacío. Cuando funcionaba, consumía 140 Kilowatios de potencia. También era bastante más rápida que cualquier computador electromecánico, ya que era capaz de efectuar 5.000 sumas por segundo.

El ENIAC era una máquina decimal, y no binaria. Es decir, los números estaban representados en forma decimal y la aritmética se realizaba también en el sistema decimal. Su memoria consistía en 20 «acumuladores», cada uno capaz de contener un número decimal de 10 dígitos. Cada dígito estaba representado por un anillo de 10 tubos de vacío. En un momento dado, sólo uno de los tubos de vacío estaba en estado ON, representando uno de los diez dígitos. Uno de los mayores inconvenientes del ENIAC era que tenía que ser programado manualmente mediante conmutadores y conectando y desconectando cables.

El ENIAC se terminó en 1946, demasiado tarde para ser utilizado durante la guerra. En lugar de eso, su primera misión fue realizar una serie de cálculos complejos, que se usaron para ayudar a determinar la viabilidad de la bomba H. El uso del ENIAC para una función distinta de aquella para la que fue construido, demostró su naturaleza de propósito general. Así, 1946 marcó el comienzo de la nueva era de los computadores electrónicos, culminando años de esfuerzo. El ENIAC siguió funcionando bajo la dirección del BRL hasta 1955, cuando fue desmontado.

La máquina de von Neumann

~~La tarea de cargar y modificar programas para el ENIAC era extremadamente tediosa. El proceso de programación podría ser más fácil si el programa se representara en una forma adecuada para ser guardado en la memoria junto con los datos.~~ Entonces, un ordenador podría conseguir sus instrucciones leyéndolas de la memoria, y se podría hacer o modificar un programa colocando los valores en una zona de memoria.

Esta idea, conocida como *concepto del programa-almacenado*, se atribuye a los diseñadores del ENIAC, sobre todo al matemático John von Neumann, que era asesor del proyecto ENIAC. La idea fue también desarrollada, aproximadamente al mismo tiempo, por Turing. La primera publicación de la idea fue en una propuesta de von Neumann para un nuevo computador en 1945, el EDVAC (Electronic Discrete Variable Computer).

En 1946, von Neumann y sus colegas empezaron, en el Instituto para Estudios Avanzados de Princeton, el diseño de un nuevo computador de *programa-almacenado*, que llamaron IAS. El computador IAS, no completado hasta 1952, es el prototipo de toda una serie de computadores de propósito general.

La Figura 2.1 muestra la estructura general del computador IAS. Esta consta de:

- Una memoria principal, que almacena tanto datos como instrucciones.
- Una unidad aritmético-lógica (ALU), capaz de hacer operaciones con datos binarios.
- Una unidad de control, que interpreta las instrucciones en memoria y provoca su ejecución.
- Un equipo de entrada-salida (E/S) dirigido por la unidad de control.

Esta estructura fue esbozada en la primera proposición de von Neumann, que merece la pena mencionar en este momento [VONM45]:

2.2. **Primero:** Como el dispositivo es, principalmente, un computador, tendrá que realizar las operaciones aritméticas elementales muy frecuentemente. Estas son: la suma, la resta, la multiplicación y la división: +, -, ×, ÷. Es, por tanto, razonable, que contenga elementos especializados sólo en estas operaciones.

Debe observarse, sin embargo, que, aunque este principio parece consistente, la manera específica de cómo se aplica requiere un examen cuidadoso [...] En cualquier caso, tendrá que existir la parte de *aritmética central* que constituirá la *primera parte específica: CA* (Central Arithmetical).

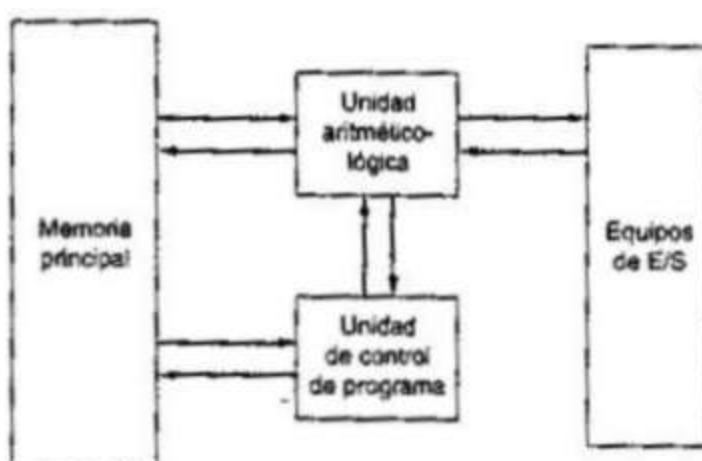


Figura 2.1. Estructura del computador IAS.

2.3. Segundo: El control lógico del dispositivo, es decir, la secuenciación adecuada de las operaciones, debe ser realizado eficientemente por un órgano central de control. Si este dispositivo de control tiene que ser versátil, es decir, servir en lo posible para todo uso, hay que diferenciar entre las instrucciones específicas que definen un problema particular, y los órganos de control general que se ocupan de que se lleven a cabo dichas instrucciones, sean cuáles sean. Las primeras deben almacenarse en algún lugar; las otras deben representarse definiendo partes operativas del dispositivo. Con el control central nos referimos sólo a esta última función, y los órganos que la realizan forman la *segunda parte específica: CC* (Central Control).

2.4. Tercero: Cualquier dispositivo que realice secuencias largas y complicadas de operaciones (concretamente de cálculo), debe tener una memoria considerable [...].

(b) Las instrucciones que gobiernan un problema complicado pueden constituir un material considerable, sobre todo si el código es circunstancial (lo cual ocurre en la mayoría de las situaciones). Este material debe tenerse en cuenta [...].

En cualquier caso, la memoria total es la *tercera parte específica del dispositivo: M* (Memoria).

2.6. Las tres partes específicas, CA, CC (juntas C) y M, corresponden a las neuronas *asociativas* del sistema nervioso humano. Queda por discutir los equivalentes a las neuronas *sensoriales o aferentes* y las *motoras o eferentes*. Estos son los órganos del dispositivo de *entrada y salida* [...].

El dispositivo tiene que estar dotado con la habilidad de mantener contacto de entrada y salida (sensorial y motor) con medios específicos de este tipo (cf. I.2); el medio será llamado el *medio de grabación exterior del dispositivo: R* (Recording) [...].

2.7. Cuarto: El dispositivo tiene que tener órganos para transferir... información a partir de R a sus partes específicas C y M. Estos órganos forman su *entrada, la cuarta parte específica: I* (Input). Veremos que lo mejor es hacer todas las transferencias a partir de R (mediante I) hasta M, y nunca directamente a partir de C [...].

2.8. Quinto: El dispositivo tiene que tener órganos para transferir [...] información a partir de sus partes específicas C y M hacia R. Estos órganos forman su *salida, la quinta parte específica: O* (Output). Veremos que es mejor, de nuevo, hacer todas las transferencias a partir de M (mediante O) a R, y nunca directamente a partir de C [...].

Salvo raras excepciones, todos los computadores de hoy en día tienen la misma estructura general y funcionamiento que la indicada en las máquinas de von Neumann. Por tanto, merece la pena en este momento describir brevemente la manera de operar del computador IAS [BURK46]. Siguiendo [HAYE88], la terminología y la notación de von Neumann han cambiado para ajustarse más a las necesidades actuales: los ejemplos e ilustraciones que acompañan a esta exposición están basados en el último texto.

La memoria del IAS consiste en 1.000 posiciones de almacenamiento, llamadas *palabras*, de 40 dígitos binarios (bits) cada una. Tanto los datos como las instrucciones se almacenan ahí. Por tanto, los números se pueden representar en forma binaria, y cada instrucción tiene también un código binario. La Figura 2.2 muestra estos formatos. Cada número se representa con un bit de signo y 39 bits de valor. Una palabra puede contener también dos instrucciones de 20 bits, donde cada instrucción consiste en un código de operación de 8 bits (codop), que especifica la operación que se va a realizar, y una dirección de 12 bits, que indica una de las palabras de la memoria (numeradas de 0 a 999).

La unidad de control dirige el IAS captando instrucciones de la memoria y ejecutándolas una a una. Para explicar esto, se necesita un diagrama de estructura más detallado, como se indica en la Figura 2.3. Esta figura muestra que, tanto la unidad de control como la ALU, contienen posiciones de almacenamiento, llamadas *registros*, definidos de la siguiente manera:

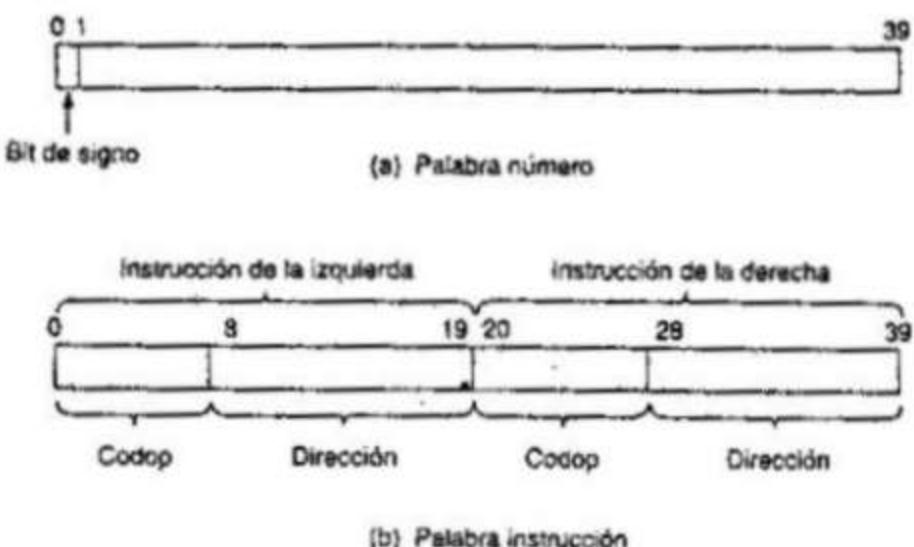


Figura 2.2. Formatos de la memoria IAS.

- **Registro temporal de memoria (MBR, Memory Buffer Register):** contiene una palabra que debe ser almacenada en la memoria, o es usado para recibir una palabra procedente de la memoria.
- **Registro de dirección de memoria (MAR, Memory Address Register):** especifica la dirección en memoria de la palabra que va a ser escrita o leída en MBR.

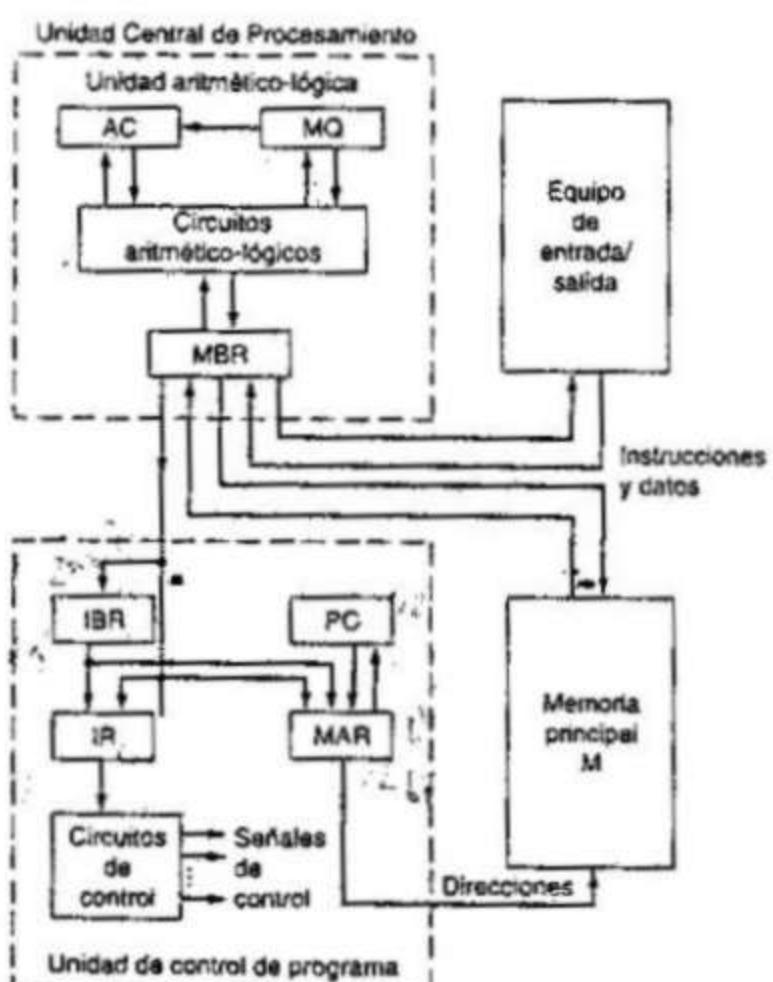
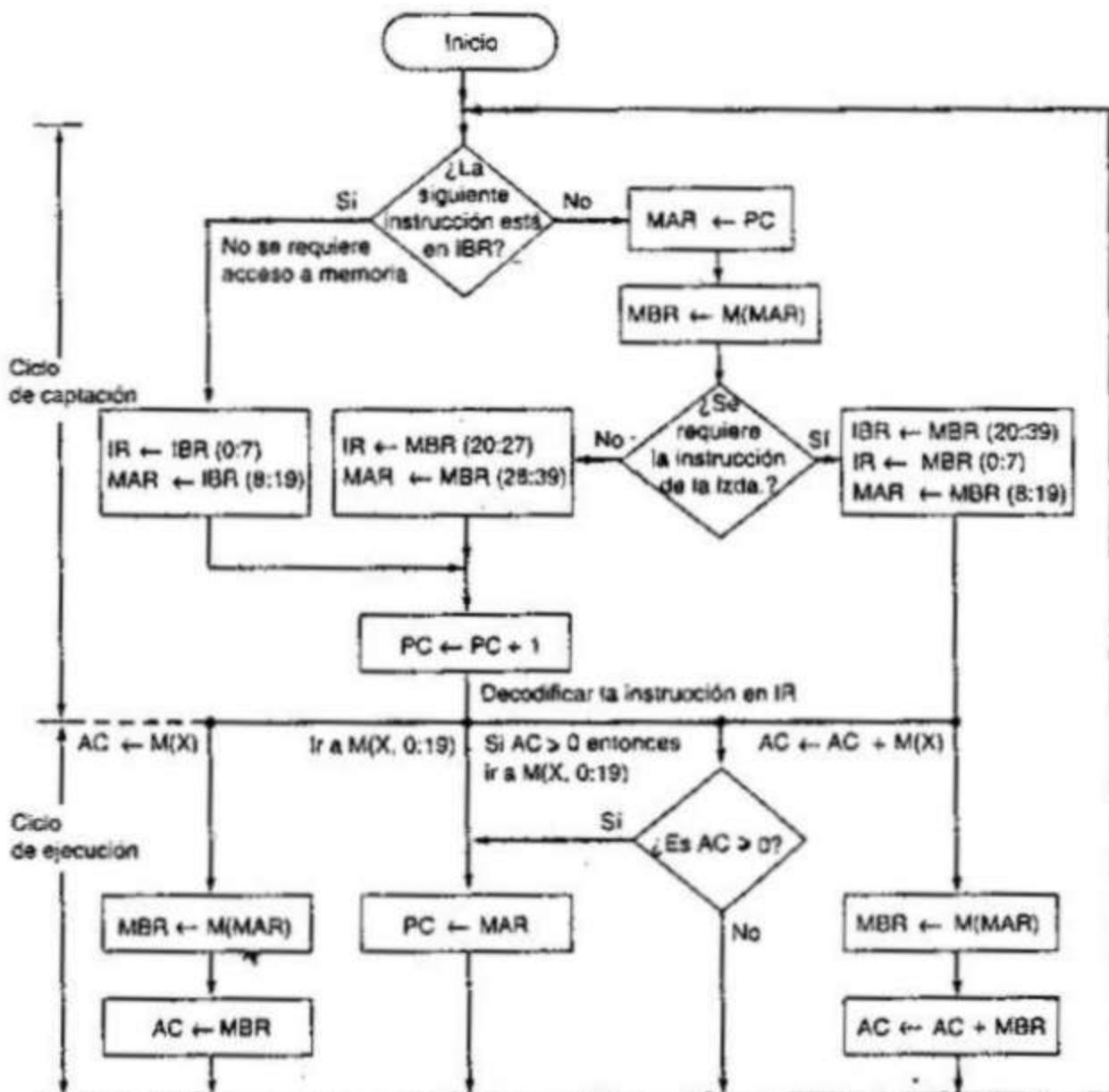


Figura 2.3. Estructura expandida del computador IAS.

- **Registro de instrucción (IR, Instruction Register):** contiene los 8 bits del código de operación de la instrucción que se va a ejecutar.
- **Registro temporal de instrucción (IBR, Instruction Buffer Register):** empleado para almacenar temporalmente la instrucción contenida en la parte derecha de una palabra en memoria.
- **Contador de programa (PC, Program Counter):** contiene la dirección de la próxima pareja de instrucciones que van a ser captadas de la memoria.
- **Acumulador (AC) y multiplicador cociente (MQ, Multiplier Quotient):** se emplean para almacenar operandos y resultados de operaciones de la ALU temporalmente. Por ejemplo, el resultado de multiplicar dos números de 40 bits es un número de 80 bits; los 40 bits más significativos se almacenan en el AC y los menos significativos en el MQ.

El IAS opera ejecutando repetidamente un *ciclo instrucción*, como se puede ver en la Figura 2.4. Cada ciclo instrucción consta de dos subciclos. Durante el *ciclo de captación*, el codop



$M(X)$ = contenido de la posición de memoria cuya dirección está en X

$(X : Y)$ = bits de X a Y

Figura 2.4. Diagrama de flujo parcial de las operaciones del IAS.

de la siguiente instrucción es cargado en el IR, y la parte que contiene la dirección es almacenada en el MAR. Esta instrucción puede ser captada desde el IBR, o puede ser obtenida de la memoria cargando una palabra en el MBR, y luego en IBR, IR y MAR.

¿Por qué la indirección? Todas estas operaciones están controladas por circuitos electrónicos, y dan lugar al uso de caminos de datos. Para simplificar la electrónica, se usa un solo registro para especificar la dirección en memoria para lectura o escritura, y un solo registro para la fuente o el destino. MAR

Una vez que el codop está en el IR, se lleva a cabo el ciclo de ejecución. Los circuitos de control interpretan el codop y ejecutan la instrucción, enviando las señales de control adecuadas para provocar que los datos se transfieran o que la ALU realice una operación.

El computador IAS tiene un total de 21 instrucciones, que se indican en la Tabla 2.1. Estas se pueden agrupar de la siguiente manera:

- Transferencia de datos: transferir datos entre la memoria y los registros de la ALU o entre dos registros de la ALU.
- Salto incondicional: normalmente, la unidad de control ejecuta instrucciones secuencialmente en la memoria. Las instrucciones de salto pueden cambiar esta secuencialidad. Esto facilita las operaciones repetitivas.
- Salto condicional: el salto depende de una condición, lo que permite puntos de decisión.
- Aritmética: operaciones realizadas por la ALU.
- Modificación de direcciones: permite que la ALU haga operaciones con las direcciones y las inserte en instrucciones almacenadas en memoria. Esto permite una considerable flexibilidad de direccionamiento en un programa.

La Tabla 2.1 presenta las instrucciones en una forma simbólica y fácil de leer. En realidad, cada instrucción debe tener el formato de la Figura 2.2b. La parte de codop (los 8 primeros bits) especifican cuál de las 21 instrucciones va a ser ejecutada. La parte de la dirección (los 12 bits restantes) especifican cuál de las 1.000 posiciones de memoria está implicada en la ejecución de la instrucción.

La Figura 2.4 muestra varios ejemplos de la ejecución de una instrucción por la unidad de control. Hay que destacar que cada operación requiere varios pasos. Algunas son bastante complejas. (La operación de multiplicación requiere 39 suboperaciones, una para cada bit excepto para el bit de signo)

Computadores comerciales

Los años 50 contemplaron el nacimiento de la industria de computadores con dos compañías, Sperry e IBM, dominando el mercado.

En 1947, Eckert y Mauchly formaron la Eckert-Mauchly Computer Corporation para fabricar computadores con fines comerciales. Su primera máquina de éxito fue el UNIVAC I (Universal Automatic Computer), que fue empleada por la oficina del censo para sus cálculos en 1950. La Eckert-Mauchly Computer Corporation formó luego parte de la división UNIVAC de la Sperry-Rand Corporation, que siguió construyendo una serie de máquinas sucesoras de la primera.

El UNIVAC I fue el primer computador comercial de éxito. Estaba diseñado, como su nombre indica, tanto para aplicaciones científicas como comerciales. El primer documento que describía el sistema mencionaba, como ejemplos de tareas que podía realizar, operaciones algebraicas con matrices, problemas de estadística, reparto de primas para las compañías de seguros de vida y problemas logísticos.

Tabla 2.1. El conjunto de instrucciones del IAS

Tipo de instrucción	Codop	Representación simbólica	Descripción
Transferencia de datos	00001010	LOAD MQ	Transferir el contenido del registro MQ al acumulador AC
	00001001	LOAD MQ,M(X)	Transferir el contenido de la posición de memoria X a MQ
	00100001	STOR M(X)	Transferir el contenido del acumulador a la posición de memoria X
	00000001	LOAD M(X)	Transferir M(X) al acumulador
	00000010	LOAD -M(X)	Transferir -M(X) al acumulador
	00000011	LOAD M(X)	Transferir el valor absoluto de M(X) al acumulador
Salto incondicional	00001101	JUMP M(X,0:19)	Captar la siguiente instrucción de la mitad izquierda de M(X)
	00001110	JUMP M(X,20:39)	Captar la siguiente instrucción de la mitad derecha de M(X)
Salto condicional	00001111	JUMP + M(X,0:19)	Si el número en el acumulador es no negativo, captar la siguiente instrucción de la mitad izquierda de M(X)
	00010000	JUMP + M(X,20:39)	Si el número en el acumulador es no negativo, captar la siguiente instrucción de la mitad derecha de M(X)
Aritmética	00000101	ADD M(X)	Sumar M(X) a AC; colocar el resultado en AC
	00000111	ADD M(X)	Sumar M(X) a AC; colocar el resultado en AC
	00000110	SUB M(X)	Restar M(X) a AC; colocar el resultado en AC
	00001000	SUB M(X)	Restar M(X) a AC; colocar el resultado en AC
	00001011	MUL M(X)	Multiplicar M(X) por MQ; colocar los bits más significativos del resultado de AC, y los menos significativos en MQ
	00001100	DIV M(X)	Dividir AC por M(X); colocar el cociente en MQ y el resto en AC
	00010100	LSH	Multiplicar el acumulador por 2; esto es, desplazar su contenido una posición a la izquierda
	00010101	RSH	Dividir el acumulador por 2; esto es, desplazar su contenido una posición a la derecha
Modificación de direcciones	00010010	STOR M(X,8:19)	Reemplazar el campo de dirección de la izquierda de M(X) por los 12 bits de la derecha de AC
	00010011	STOR M(X,28:39)	Reemplazar el campo de dirección de la derecha de M(X) por los 12 bits de la derecha de AC

El UNIVAC II, que tenía una capacidad de memoria mayor y más aplicaciones que el UNIVAC I, salió al mercado a finales de los 50, e ilustra varias tendencias, que han permanecido como características de la industria de computadores. Primera: los avances en la tecnología permiten a las compañías seguir construyendo computadores más grandes y más potentes. Segunda: cada compañía intenta hacer sus nuevas máquinas superiores y compatibles con las anteriores. Esto significa que los programas escritos para las viejas máquinas pueden ejecutarse en las nuevas máquinas. Esta estrategia se adopta para retener la base de clientes; es decir que, cuando un cliente decide comprar una máquina nueva, probablemente la comprará a la misma compañía para evitar perder su inversión en programas.

La división UNIVAC comenzó también el desarrollo de la serie de computadores 1100, que fue su producto principal. Esta serie ilustra una distinción que existió en aquella época. El primer modelo, el UNIVAC 1103, y sus sucesores durante muchos años, estaban diseñados principalmente para aplicaciones científicas que implicaban cálculos largos y complejos. Otras compañías se centraron en el campo de la gestión, lo que conllevaba el procesamiento de grandes cantidades de textos. Esta separación desapareció hace tiempo, pero fue patente durante algunos años.

IBM, que había ayudado a construir el Mark I y era entonces el principal fabricante de equipos de procesamiento con tarjetas perforadas, sacó su primer computador con programas almacenados electrónicamente, el 701, en 1953. El 701 fue diseñado principalmente para aplicaciones científicas [BASH81]. En 1955, IBM presentó los productos 702, que tenían varias características hardware que los hacían adecuados para aplicaciones de gestión. Estos fueron los primeros de una larga serie de computadores 700/7000, que situaron a IBM como el fabricante de computadores dominante, con gran diferencia.

LA SEGUNDA GENERACIÓN: LOS TRANSISTORES

El primer cambio importante en los computadores electrónicos vino con la sustitución de los tubos de vacío por transistores. El transistor es más pequeño, más barato, disipa menos calor y puede ser usado de la misma forma que un tubo de vacío en la construcción de computadores. Mientras que un tubo de vacío requiere cables, placas de metal, una cápsula de cristal y vacío, el transistor es un dispositivo de estado sólido, hecho con silicio.

El transistor fue inventado en los Laboratorios Bell en 1947, y en los años 50 y provocó una revolución electrónica. Sin embargo, los computadores completamente transistorizados no estuvieron disponibles comercialmente hasta finales de los 50. IBM no fue la primera compañía que lanzó esta nueva tecnología. NCR y, con más éxito, RCA fueron los primeros en sacar pequeñas máquinas de transistores. IBM los siguió pronto con la serie 7000.

El uso del transistor define la segunda generación de computadores. La clasificación de los computadores en generaciones basándose en la tecnología hardware empleada, fue amplia-

Tabla 2.2. Generaciones de computadores

Generación	Fechas aproximadas	Tecnología	Velocidad típica (operaciones por segundo)
1	1946-1957	Válvulas	40.000
2	1958-1964	Transistores	200.000
3	1965-1971	Pequeña y media integración	1.000.000
4	1972-1977	Gran integración	10.000.000
5	1978-	Alta integración	100.000.000

mente aceptada (Tabla 2.2). Cada nueva generación se caracteriza por la mayor velocidad, mayor capacidad de memoria y menor tamaño que la generación anterior.

También hay otros cambios. En la segunda generación se introdujeron unidades lógicas y aritméticas y unidades de control más complejas, así como el uso de lenguajes de programación de alto nivel, y se proporcionó un software del sistema con el computador.

La segunda generación es destacable también por la aparición de la empresa Digital Equipment Corporation (DEC). DEC fue fundada en 1957, y en este año sacó su primer computador, el PDP-1. Este computador y esta compañía iniciaron el desarrollo de los minicomputadores, que fue de gran importancia en la tercera generación.

El IBM 7094

Desde la introducción en 1952 de la serie 700 y la introducción, en 1964, del último miembro en 1964 de la serie 7000, esta línea de productos sufrió la evolución típica de los computadores. Los productos sucesivos de la línea presentaron un aumento de prestaciones y capacidad y/o la disminución de precios.

Esta tendencia se puede ver en la Tabla 2.3. El tamaño de la memoria principal, en múltiplos de 2^{10} palabras de 36 bits, creció de 2K ($1K = 2^{10}$) a 32K palabras, mientras que el tiempo de acceso a una palabra de memoria, el tiempo de ciclo de memoria, cayó de 30μs a 1,4 μs. El número de códigos de operación creció de un modesto 24 a 185.

La columna final indica la velocidad de ejecución relativa de la CPU. El incremento de velocidad se logró mejorando la electrónica (por ejemplo, una implementación con transistores es más rápida que con tubos de vacío) y con una circuitería más compleja. Por ejemplo, el IBM 7094 incluye un registro de respaldo de instrucciones, usado como buffer de la siguiente instrucción. La unidad de control capta las dos palabras adyacentes de la memoria para captar una instrucción. Excepto en una instrucción de salto, que suele ser poco frecuente, esto significa que la unidad de control tiene que acceder a la memoria en busca de una instrucción en sólo la mitad de los ciclos de instrucción. Esta precaptación reduce considerablemente el tiempo medio de ciclo de instrucción.

El significado del resto de las columnas de la Tabla 2.3 es claro tras la explicación anterior.

La Figura 2.5 muestra una configuración (con muchos periféricos) del IBM 7094, que es representativo de los computadores de la segunda generación [BELL71a]. Merece la pena señalar varias diferencias con el computador IAS. La más importante es el uso de canales de datos. Un canal de datos es un módulo de E/S independiente, con su propio procesador y su propio conjunto de instrucciones. En un computador con tales dispositivos, la CPU no ejecuta instrucciones detalladas de E/S. Tales instrucciones son almacenadas en una memoria principal, para ser ejecutadas con un procesador de uso específico para el canal de datos mismo. La CPU inicia una transferencia de E/S enviando señales de control al canal de datos, instruyéndolo para ejecutar una secuencia de instrucciones en memoria. El canal de datos realiza esta tarea independientemente de la CPU y de las señales de la CPU hasta que la operación se completa. Esta disposición libera a la CPU de una carga de procesamiento considerable.

Otra característica es el multiplexor, que es el punto de conexión central de los canales de datos, la CPU y la memoria. El multiplexor organiza los accesos a la memoria desde la CPU y los canales de datos, permitiendo a estos dispositivos actuar de forma independiente.

Tabla 2.3. Ejemplos de miembros de la serie IBM 700/7000

Año de los	Primera entrega	Tecnología de la CPU	Tecnología de la memoria	Tiempo de ciclo (ns)	Tamaño de memoria (K)	Número de codops	Número de registros índice	Punto flotante por hardware	Solapamiento de E/S (Canales)	Solapamiento de captación instrucciones	Velocidad de captación relativa a 7011
701	1952	Tubos de vacío	Tubos electroestática	30	2-4	24	0	no	no	no	1
'04	1955	Tubos de vacío	Núcleos	12	4-32	90	3	sí	no	no	2.5
09	1958	Tubos de vacío	Núcleos	12	32	140	3	sí	sí	no	4
90	1960	Transistor	Núcleos	2.18	32	169	3	sí	sí	no	25
341	1962	Transistor	Núcleos	2	32	185	7	sí (doble precisión)	sí	sí	30
411	1964	Transistor	Núcleos	1.4	32	185	7	sí (doble precisión)	sí	sí	50



Figura 2.5. Configuración de un IBM 7094.

LA TERCERA GENERACIÓN: LOS CIRCUITOS INTEGRADOS

A un transistor simple y autocontenido se le llama *componente discreto*. A través de los años 50 y principios de los 60, los equipos electrónicos estaban compuestos en su mayoría por componentes discretos (transistores, resistencias, capacidades, etc.). Los componentes discretos se fabricaban separadamente, encapsulados en sus propios contenedores, y soldados o cableados juntos en tarjetas de circuitos en forma de panel, que eran instalados en computadoras.

dores, osciloscopios y otros equipos electrónicos. Cuando un dispositivo necesitaba un transistor, había que soldar éste, que tenía una forma de un pequeño tubo de metal y contenía una pieza de silicio del tamaño de la cabeza de un alfiler, en una tarjeta de circuitos. Todo el proceso de fabricación, desde el transistor hasta el panel de circuitos, era caro y engorroso.

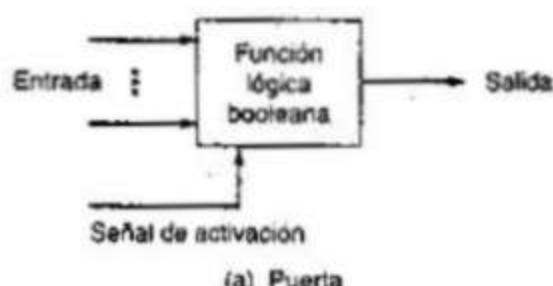
Estos hechos fueron el comienzo del surgimiento de problemas en la industria de computadores. Los primeros computadores de la segunda generación contenían alrededor de 10.000 transistores. Esta cantidad creció a cientos de miles, haciendo cada vez más difícil la fabricación de máquinas nuevas y más potentes.

En 1958 ocurrió algo que revolucionó la electrónica e inauguró la era de la microelectrónica: la invención del circuito integrado. El circuito integrado define la tercera generación de computadores. En esta sección haremos una breve introducción a la tecnología de circuitos integrados. Después veremos los que, quizás, sean los dos miembros más importantes de la tercera generación, que surgieron al principio de la era: el IBM Sistema 360 y el DEC PDP-8.

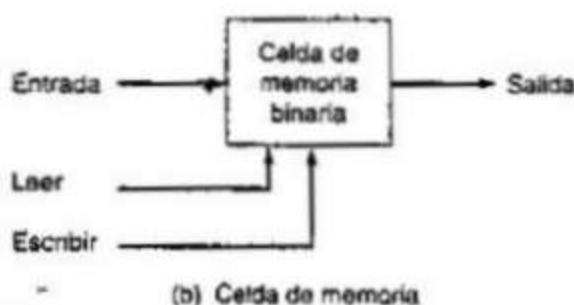
Microelectrónica

Microelectrónica significa literalmente «pequeña electrónica». Desde los comienzos de la electrónica digital y la industria de computadores, ha habido una tendencia, persistente y consistente, a la reducción del tamaño de los circuitos electrónicos digitales. Antes de examinar las implicaciones y beneficios de esta tendencia, necesitamos decir algo sobre la naturaleza de la electrónica digital. En el Apéndice A se encuentra una discusión más detallada.

Los elementos básicos de un computador digital, como ya sabemos, deben ofrecer almacenamiento, procesamiento y control de funciones. Sólo se requieren dos tipos fundamentales de componentes (Figura 2.6): puertas y celdas de memoria. Una puerta es un dispositivo que implementa una función lógica o booleana simple, como: SI A AND B ES CIERTO, ENTONCES C ES CIERTO (puerta AND). A tales dispositivos se les llama puertas, porque controlan el flujo en cierta manera, como lo hacen las puertas de un canal. La celda de memoria es un dispositivo que puede almacenar un dato de un bit; es decir, el dispositivo puede



(a) Puerta



(b) Celda de memoria

Figura 2.6. Elementos de un computador básico.

estar, en un instante dado, en uno de dos estados estables. Interconectando muchos de estos dispositivos fundamentales, podemos construir un computador. Podemos relacionar esto con nuestras cuatro funciones básicas de la siguiente forma:

- **Almacenamiento de datos:** proporcionado por las celdas de memoria.
- **Procesamiento de datos:** proporcionado por las puertas.
- **Transferencias de datos:** los caminos entre componentes se usan para llevar datos de memoria a memoria y de memoria a través de las puertas, a memoria.
- **Control:** los caminos entre componentes pueden llevar las señales de control. Por ejemplo, una puerta tendrá dos entradas de datos más una entrada de control que activará la puerta. Cuando la señal de control está en ON, la puerta realiza su función con los datos de entrada, y produce un dato de salida. De manera similar, las celdas de memoria almacenarán el bit en su entrada si la señal de control WRITE está ON, y situarán el bit en la salida cuando la señal de control READ esté ON.

Por tanto, un computador consta de puertas, celdas de memoria e interconexiones entre estos elementos. Las puertas y las celdas de memoria están constituidas por componentes electrónicos simples.

Aunque la tecnología de transistores introducida en la segunda generación de computadores fue una gran mejora respecto a los tubos de vacío, aún quedaban problemas. Los transistores se montaban individualmente con encapsulados independientes interconectados con cables separados, en tarjetas de circuito impreso. Este proceso era complejo, consumía tiempo y era propenso a errores.

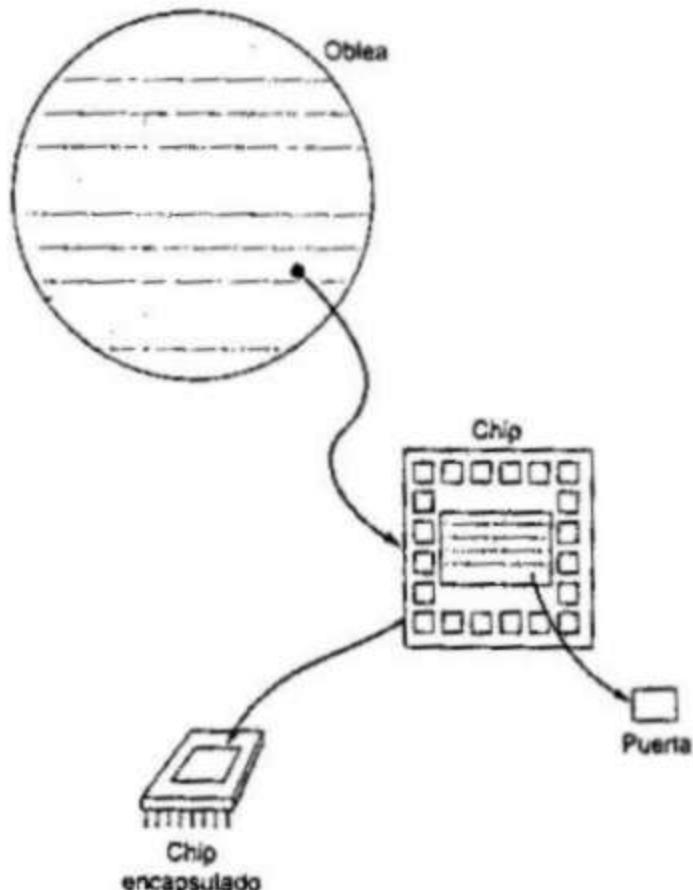


Figura 2.7. Relación entre oblea, chip y puerta.

Los circuitos integrados utilizaron el hecho de que tales componentes (como transistores, resistencias y conductores) podían ser fabricados a partir de un semiconductor como el silicio. Es simplemente un avance del arte del estado sólido consistente en fabricar un circuito entero en un pequeño trozo de silicio, en vez de ensamblar componentes discretos hechos a partir de trozos de silicio separados, en el mismo circuito. Se pueden construir cientos, e incluso miles, de transistores al mismo tiempo en una sola oblea de silicio. Igualmente importante es que estos transistores pueden ser conectados con un proceso de metalización para formar circuitos.

La Figura 2.7 muestra los conceptos clave de un circuito integrado. Se divide una fina oblea de silicio en una matriz de pequeñas áreas, cada una de unos pocos milímetros cuadrados. Se fabrica el mismo patrón de circuito en cada área, y la oblea se divide en chips. Cada chip consiste en muchas puertas, más una serie de puntos para conexiones de entrada y salida. El chip es encapsulado en una carcasa que lo protege y proporciona patas para conectar dispositivos fuera del chip. Varios de estos elementos pueden ser interconectados en una tarjeta de circuito impreso para producir circuitos más complejos y mayores.

Inicialmente sólo podían fabricarse y encapsularse juntas, con fiabilidad, unas pocas puertas o celdas de memoria. A estos primeros circuitos integrados se les llama *de pequeña escala de integración* (SSI, Small-Scale Integration). A medida que el tiempo pasó, fue posible encapsular más y más componentes en un mismo chip. Este crecimiento en densidad se puede ver en la Figura 2.8; ésta es una de las tendencias tecnológicas más importantes que nunca se ha visto. Esta figura refleja la famosa ley de Moore, que fue propuesta por Gordon Moore, co-fundador de Intel, en 1965 [MOOR65]. Moore observó que el número de transistores que se podrían integrar en un solo chip se duplicaba cada año y se predecía, correctamente, que esto continuaría en un futuro cercano. Para sorpresa de muchos, incluido Moore, este ritmo continuaría año tras año y década tras década. El ritmo disminuyó, duplicándose cada 18 meses en los 70, pero ha mantenido esta velocidad desde entonces.

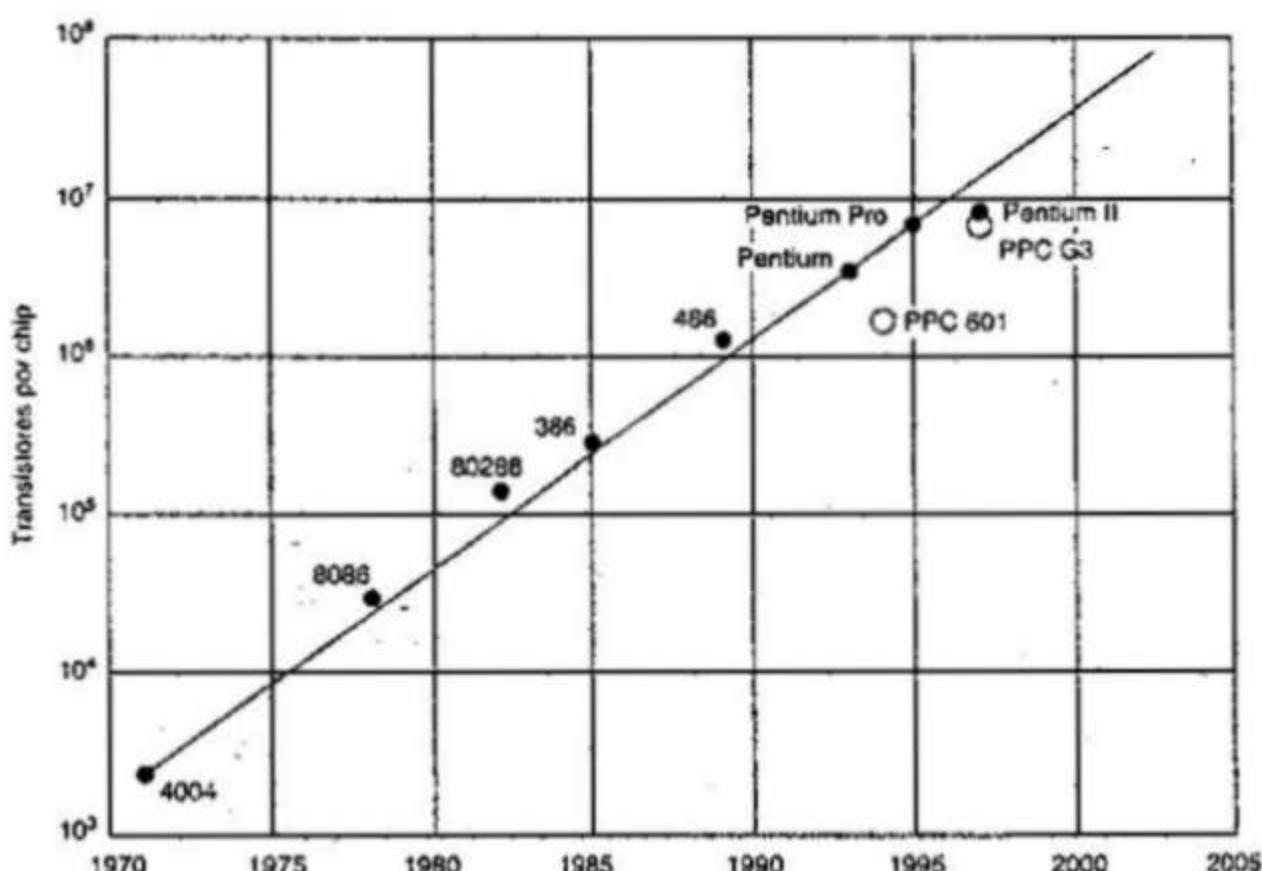


Figura 2.8. Evolución del número de transistores en los procesadores.

Las consecuencias de la ley de Moore son profundas:

1. El precio de un chip ha permanecido prácticamente invariable a través de este periodo de rápido crecimiento en densidad. Esto significa que el costo de la lógica del computador y de la circuitería de la memoria han caído a una velocidad drástica.
2. Ya que los elementos de la lógica y la memoria están más próximos en chips más densamente encapsulados, la longitud de las interconexiones eléctricas ha disminuido, incrementándose así la velocidad operativa.
3. El computador es ahora más pequeño, lo que lo hace más adecuado para más entornos.
4. Hay una reducción de las necesidades de potencia y refrigeración.
5. Las interconexiones de los circuitos integrados son mucho más fiables que las conexiones soldadas. Con más circuitos en cada chip, hay menos conexiones entre chips.

IBM Sistema/360

En 1964, IBM tenía un firme dominio del mercado con sus máquinas de la serie 7000. Aquel año, IBM anunció el Sistema/360, una nueva familia de productos de computadores. Aunque el anuncio mismo no fue ninguna sorpresa, contenía algunas noticias desagradables para los clientes habituales de IBM: la línea de productos 360 era incompatible con las máquinas IBM anteriores. Por ello, la transición al 360 sería difícil para los clientes de IBM. Este fue un paso audaz de IBM, pero sentían que era necesario romper con algunas de las limitaciones de la arquitectura 7000 y producir un sistema capaz de evolucionar junto con la nueva tecnología de circuitos integrados [PADE81, GIFF87]. La estrategia resultó provechosa, tanto técnica como financieramente. El 360 fue el éxito de la década, y consolidó a IBM como el dominante absoluto en las ventas de computadores, con una cuota de mercado por encima del 70 %, y, con algunas modificaciones y ampliaciones, la arquitectura del 360 permanece hasta hoy en día en la arquitectura de los grandes computadores de IBM. A lo largo del texto, se pueden encontrar ejemplos que utilizan esta arquitectura.

El Sistema/360 fue la primera familia de computadores de la historia que se planeó. La familia abarcaba un amplio rango de prestaciones y precios. La Tabla 2.4 indica alguna de las características claves de los distintos modelos en 1965 (cada miembro de la familia se distingue por un número de modelo). Los distintos modelos eran compatibles en el sentido de que, un programa escrito para un modelo, tenía que ser capaz de ser ejecutado por otro modelo de la serie, con la única diferencia del tiempo de ejecución.

Tabla 2.4. Características clave de la familia Sistema/360

Características	Modelo 30	Modelo 40	Modelo 50	Modelo 65	Modelo - 75
Tamaño máximo de memoria (bytes)	~64K	256K	256K	512K	512K
Velocidad de transferencia de datos procedentes de la memoria (Mbytes/segundo)	0,5	0,8	2,0	8,0	16,0
Tiempo de ciclo del procesador (μ segundos)	1,0	0,625	0,5	0,25	0,2
Velocidad relativa	1	3,5	10	21	50
Número máximo de canales de datos	3	3	4	5	6
Máxima velocidad de transferencia de datos en un canal (Kbytes/segundo)	250	400	800	1250	1250

El concepto de familia de computadores compatibles era a la vez novedoso y extremadamente exitoso. Un cliente con necesidades modestas y un presupuesto limitado podía empezar con el modelo 30, relativamente barato. Más tarde, si las necesidades del cliente crecían, era posible pasarse a una máquina más rápida y con más memoria, sin sacrificar la inversión ya realizada en software. Las características de una familia son:

- Conjunto de instrucciones similar o idéntico: en muchos casos, se encuentran exactamente el mismo conjunto de instrucciones máquina en todos los miembros de la familia. Así, un programa que se ejecuta en una máquina, se podrá ejecutar en cualquier otra. En algunos casos, el computador más bajo de la familia tiene un conjunto de instrucciones que es un subconjunto del computador más alto de la familia. Esto quiere decir que los programas se pueden mover hacia arriba, pero no hacia abajo.
- Sistemas operativos similares o idénticos: el mismo sistema operativo básico está disponible para todos los miembros de la familia. En algunos casos, se añaden características complementarias a los miembros más altos.
- Velocidad creciente: la velocidad de ejecución de las instrucciones se incrementa conforme se sube desde los miembros más bajos a los más altos de la familia.
- Número creciente de puertos de E/S: conforme se va desde lo más bajo a los más alto de la familia.
- Tamaño de memoria creciente: conforme se va de lo más bajo a lo más alto de la familia.
- Costo creciente: conforme se va de lo más bajo a lo más alto de la familia.

Cómo podría implementarse tal concepto de familia? Las diferencias entre los modelos se basaron en tres factores: la velocidad básica, el tamaño y el grado de simultaneidad [STEV64]. Por ejemplo, podría lograrse mayor velocidad en la ejecución de una instrucción dada usando una circuitería más compleja en la ALU, permitiendo que las suboperaciones se llevaran a cabo en paralelo. Otro modo de incrementar la velocidad era incrementar la amplitud del camino de los datos entre la memoria principal y la CPU. En el Modelo 30, sólo se podía captar un byte (8 bits) a la vez de la memoria principal, mientras que en el Modelo 70 se podían captar 8 bytes a la vez.

El Sistema/360 no solamente dictó la carrera hacia el futuro de IBM, sino que también tuvo un profundo impacto en toda la industria. Muchas de sus características se han convertido en un estándar para otros grandes computadores.

DEC PDP-8

En el mismo año que IBM lanzó su primer Sistema/360, tuvo lugar otro lanzamiento trascendental: el PDP-8 de DEC. En aquella época, cuando la mayoría de los computadores requerían una habitación con aire acondicionado, el PDP-8 (llamado por la industria «minicomputador» en honor a la minifalda de aquellos tiempos) era lo bastante pequeño para ser colocado en lo alto de una mesa de laboratorio o embutido en otro equipo. No podía hacer todo lo que hacían los grandes computadores, pero a 16.000 dólares era suficientemente barato para que cada técnico de laboratorio tuviera uno. Por contra, los computadores de la serie Sistema/360, presentados sólo unos meses antes, costaban cientos de miles de dólares.

El bajo costo y pequeño tamaño del PDP-8 permitía a otros fabricantes comprarse un PDP-8 e integrarlo en un sistema global para revenderlo. Estos otros fabricantes se conocían como fabricantes de equipos originales (OEM), y el mercado de OEM llegó a tener, y aún tiene, la mayor cuota del mercado de computadores.

El PDP-8 fue un éxito inmediato, y logró el enriquecimiento de DEC. Esta máquina, y los otros miembros de la familia PDP-8 que la siguieron (véase la Tabla 2.5), lograron un estatus

Tabla 2.5. Evolución del PDP-8 (VOEL.88)

Modelo	Primeras ventas	Costo del procesador + 4K palabras de 12-bit de memoria (miles de \$)	Veloc. transfer. de memoria (Pal./μseg.)	Vol. (pies cúbicos)	Innovaciones y mejoras
PDP-8	4/65	16,2	1,26	8,0	Producción automática con conductores enrollados («wire-wrapping»)
PDP-8/5	9/66	8,79	0,08	3,2	Implementación de instrucciones serie
PDP-8/1	4/68	11,5	1,34	8,0	Circuitos integrados de media escala
PDP-8/L	11/68	7,0	1,26	2,0	Chasis menor
PDP-8/E	3/71	4,99	1,52	2,2	Omnibus
PDP-8/M	6/72	3,69	1,52	1,8	Chasis de la mitad de tamaño y menos ranuras que el 8/E
PDP-8/A	1/75	2,6	1,34	1,2	Memoria de semiconductor; procesador de punto flotante

de producción antes reservado a los computadores IBM, con alrededor de 50.000 máquinas vendidas en los siguientes doce años. Como se dice en la historia oficial de DEC, el PDP-8 «estableció el concepto de minicomputador, abriendo el camino a una industria de miles de millones de dólares». También estableció a DEC como el vendedor de miniordenadores número uno, y cuando el PDP-8 alcanzó el fin de su vida útil, DEC era el segundo fabricante de computadores detrás de IBM.

En contraste con la arquitectura de conmutador central (Figura 2.5) usada por IBM en sus sistemas 700/7000 y 360, los últimos modelos del PDP-8 usaban una estructura que ahora es prácticamente universal para minicomputadores y microcomputadores: la estructura de bus. Esto se muestra en la Figura 2.9. El bus PDP-8, llamado Omnibus, consiste en 96 hilos conductores separados, usados para control, direccionamiento y datos. Como todos los componentes del sistema comparten un conjunto de caminos, su uso debe estar controlado por la CPU. Esta arquitectura es altamente flexible, permitiendo conectar módulos al bus para crear varias configuraciones. En la Figura 2.10 se puede ver la configuración del PDP-8/E.

ÚLTIMAS GENERACIONES

Más allá de la tercera generación, hay menos acuerdo general en la definición de las generaciones de computadores. En la Tabla 2.2 se sugieren las que serían la cuarta y la quinta generación, basadas en los avances de la tecnología de los circuitos integrados. Con la introducción de la integración a gran escala (LSI, Large-Scale Integration), podía haber más de 1.000 componentes en un simple chip de circuito integrado. Con la integración a muy gran escala (VLSI, Very-Large Scale Integration), se lograron más de 10.000 componentes por chip, y los chips VLSI actuales pueden contener más de 100.000 componentes.

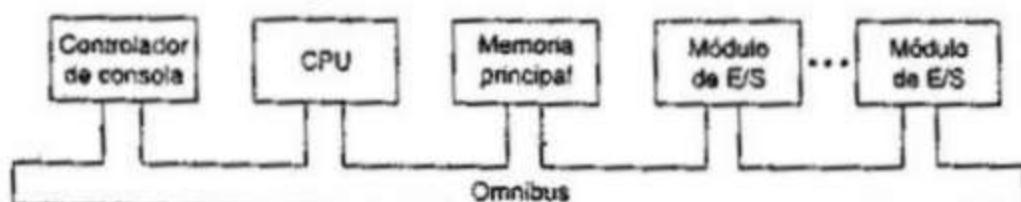


Figura 2.9. Estructura del bus PDP-8.

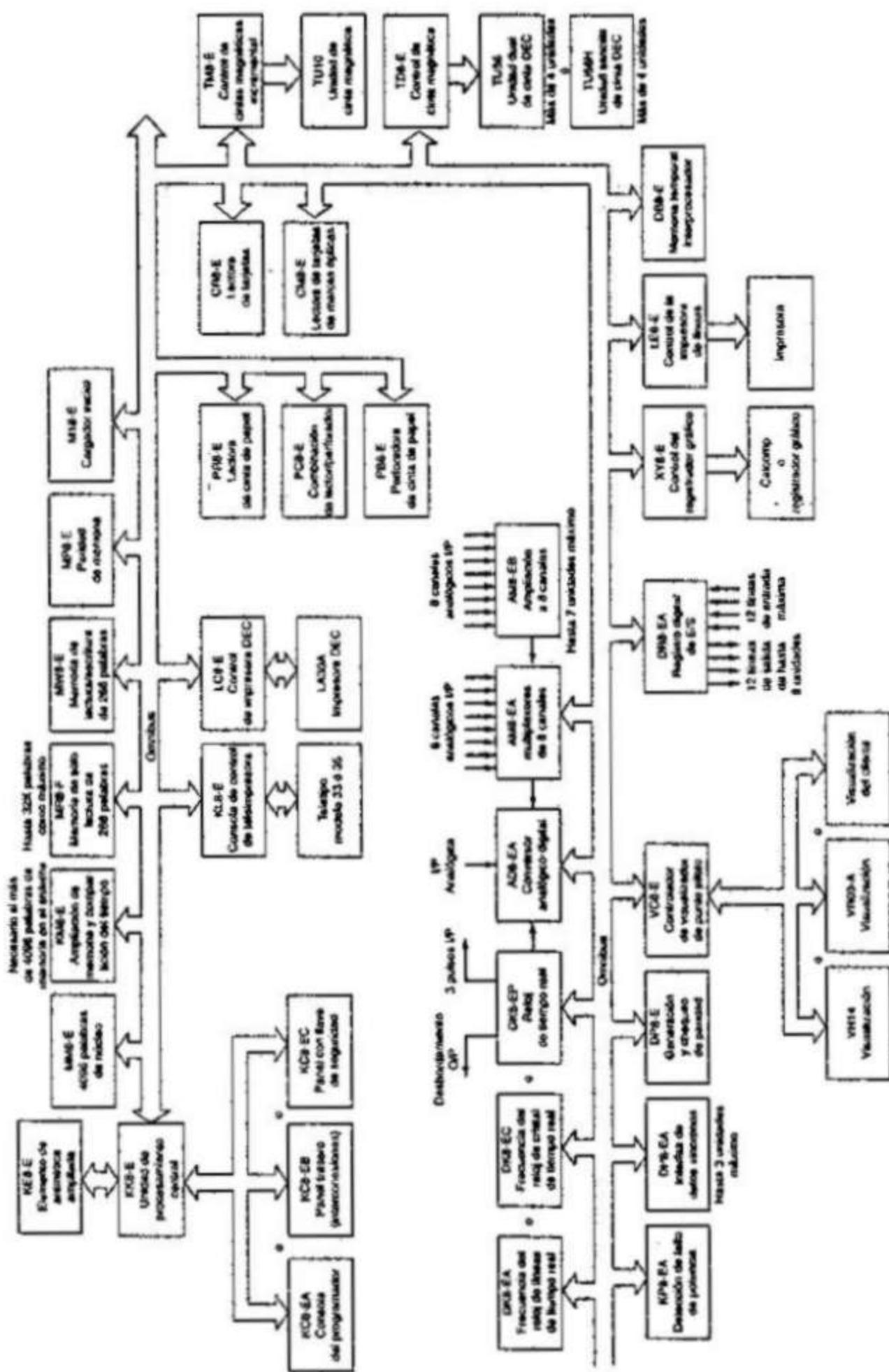


Figura 2.10. Diagrama de bloques del sistema PDP-8/E.

Con el gran avance de la tecnología, la rápida introducción de nuevos productos, y la importancia del software y las comunicaciones, así como del hardware, la clasificación en generaciones se vuelve cada vez menos clara y menos significativa. Se podría decir que la aplicación comercial de nuevos desarrollos resultó uno de los principales cambios de principios de los años 70, y los resultados de estos cambios duran todavía. En esta sección, mencionaremos dos de los más importantes.

Memoria semiconductor

La primera aplicación de la tecnología de circuitos integrados en computadores, dejó a un lado la construcción del procesador (la unidad de control y la unidad aritmético-lógica) con chips de circuitos integrados. Sin embargo, se encontró con que esta misma tecnología podía usarse para construir memorias.

En los años 50 y 60, la mayoría de las memorias de los computadores se hacían con pequeños anillos de material ferromagnético, cada uno de un dieciseisavo de pulgada de diámetro. Estos anillos de ferrita se insertaban en mallas de finos cables engarzados en pequeños marcos, dentro del computador. Se magnetizaba en un sentido el anillo (llamado *núcleo*) y representaba un uno; magnetizado en el otro sentido, representaba un cero. La memoria de núcleo magnético era más bien rápida: tardaba tan poco como una milésima de segundo en leer un bit almacenado en memoria. Pero era cara, voluminosa y usaba lectura destructiva: el simple hecho de leer un núcleo borraba los datos almacenados en él. Era, por consiguiente, necesario hacer circuitos que recuperaran el dato tan pronto como se extraía.

Entonces, en 1970, Fairchild produjo la primera memoria semiconductor con relativa capacidad. Este chip, del tamaño de un sencillo núcleo de ferrita, podía tener 256 bits de memoria. Era no destructiva, y mucho más barata que un núcleo. Tardaba solamente 70 mil millonésimas de segundo en leer un bit. Sin embargo, el costo por bit era mayor que el de un núcleo.

En 1974, ocurrió un hecho sorprendente: el precio por bit de memoria semiconductor cayó por debajo del precio por bit de memoria de núcleo. Siguiendo esto, ha habido una continua disminución del precio de la memoria, acompañado de un correspondiente aumento de la densidad de memoria. Esto ha llevado, en pocos años, a hacer máquinas más pequeñas y más rápidas con el mismo tamaño de memoria que máquinas más grandes y más caras. El desarrollo de la tecnología de memorias, junto con el desarrollo de la tecnología de procesadores, del que hablaremos después, cambiaron la naturaleza de los computadores en menos de una década. Aunque los computadores caros y voluminosos permanecieron dentro del panorama, el computador se ha llevado también al «consumidor final», en forma de máquinas de oficina y de computadores personales.

A partir de 1970, la memoria semiconductor ha tenido ocho generaciones: 1K, 4K, 16K, 64K, 256K, 1M, 4M, y ahora 16M bits en un solo chip ($1K = 2^{10}$, $1M = 2^{20}$). Cada generación ha proporcionado cuatro veces más densidad de almacenamiento que la generación previa, junto con un menor costo por bit y una mayor velocidad de acceso.

Microprocesadores

Igual que la densidad de elementos en los chips de memoria ha continuado creciendo, también lo ha hecho la densidad de elementos de procesamiento. Conforme el tiempo pasaba, en cada chip había más y más elementos, así que cada vez se necesitaban menos y menos chips para construir un procesador de un computador.

En 1971, se hizo una innovación sensacional, cuando Intel desarrolló su 4004. El 4004 fue el primer chip que contenía todos los componentes de la CPU en un solo chip: el microprocesador había nacido.

El 4004 podía sumar dos números de 4 bits y multiplicar sólo con sumas sucesivas. Según los estándares de hoy en día, el 4004 es muy primitivo, pero marcó el comienzo de la evolución continua en capacidad y potencia de los microprocesadores.

Esta evolución se puede ver más fácilmente, considerando el número de bits que el procesador trata a la vez. No hay una medida clara de esto, pero quizás la mejor medida es la anchura del bus de datos: el número de bits de un dato que puede venir o ir al procesador a la vez. Otra medida es el número de bits del acumulador o del conjunto de registros de uso general. A menudo, estas medidas coinciden, pero no siempre. Por ejemplo, hay una serie de microprocesadores que operan con números de 16 bits en los registros, pero que sólo pueden leer y escribir 8 bits a la vez.

El siguiente paso importante en la evolución de los microprocesadores fue la introducción en 1972 del Intel 8008. Este fue el primer microprocesador de 8 bits, y era casi dos veces más complejo que el 4004.

Ninguno de estos pasos tuvo el impacto del siguiente acontecimiento importante: la introducción del Intel 8080 en 1974. Este fue el primer microprocesador de uso general. Mientras que el 4004 y el 8008 habían sido diseñados para aplicaciones específicas, el 8080 fue diseñado para ser la CPU de un microcomputador de propósito general. Al igual que el 8008, el 8080 es un microprocesador de 8 bits. El 8080, sin embargo, es más rápido, tiene un conjunto de instrucciones más rico, y tiene una capacidad de direccionamiento mayor.

Sobre la misma época empezaron a desarrollarse los microprocesadores de 16 bits. Sin embargo, hasta el final de los 70 no aparecieron estos potentes microprocesadores de 16 bits de propósito general. Uno de estos fue el 8086. El siguiente paso en esta tendencia ocurrió en 1981, cuando los Laboratorios Bell y Hewlett-Packard desarrollaron microprocesadores de un solo chip de 32 bits. Intel introdujo su microprocesador de 32 bits, el 80386, en 1985 (Tabla 2.6).

Tabla 2.6. Evolución de los microprocesadores de Intel

(a) Procesadores de la década de los 70

	4004	8008	8080	8086	8088
Fecha de introducción	15/11/71	1/4/72	1/4/74	8/6/78	1/6/79
Frecuencia de reloj	108KHz	108KHz	2MHz	5MHz, 8MHz, 10MHz	5MHz, 8MHz
Anchura del bus	4 bits	8 bits	8 bits	16 bits	8 bits
Número de transistores (micrones)	2.300 (10)	3.500	6.000 (6)	29.000 (3)	29.000 (3)
Memoria direccionable	640 bytes	16 kbytes	64 kbytes	1 Mbyte	1 Mbyte
Memoria virtual	—	—	—	—	—

Tabla 2.6. Continuación

(b) Procesadores de la década de los 80

	80286	Intel386TM DX Microprocesador	Intel386TM SX Microprocesador	Intel486TM DX CPU Microprocesador
Fecha de introducción	1/2/82	17/10/85	16/6/88	10/4/89
Velocidad de reloj	6 MHz- 12,5 MHz	15 MHz-33 MHz	16 MHz-33 MHz	25 MHz-50 MHz
Anchura del bus	16 bits	32 bits	16 bits	32 bits
Número de transistores (micrones)	134.000 (1,5)	275.000 (1)	275.000 (1)	1,2 millones (0,8-1)
Memoria direccionable	16 megabytes	4 gigabytes	4 gigabytes	4 gigabytes
Memoria virtual	1 gigabyte	64 terabytes	64 terabytes	64 terabytes

(c) Procesadores de la década de los 90

	Intel486TM SX Microprocesador	Pentium® Procesador	Pentium® Pro procesador	Pentium® II Procesador
Fecha de introducción	22/4/91	22/3/93	1/11/95	7/5/97
Velocidad de reloj	16 MHz-33 MHz	80 MHz-166 MHz	150 MHz-200 MHz	200 MHz-300 MHz
Anchura del bus	32 bits	32 bits	64 bits	64 bits
Número de transistores (micrones)	1.185 millones (1)	3,1 millones (,8)	5,5 millones (0,6)	7,5 millones
Memoria direccionable	4 megabytes	4 gigabytes	64 gigabytes	84 gigabytes
Memoria virtual	64 gigabytes	64 terabytes	64 terabytes	64 terabytes

fuente: Intel Corp. <http://www.intel.com/intel/museum/25anniv/hof/tspecs.htm>

2. DISEÑO PARA CONSEGUIR MEJORES PRESTACIONES

Año tras año, el precio de los computadores continúa cayendo dramáticamente, mientras que las prestaciones y la capacidad de estos sistemas sigue creciendo. En una tienda se puede conseguir un computador, por menos de 1.000 dólares, con prestaciones similares a las de un IBM de hace 10 años. Dentro de un computador personal, incluyendo el microprocesador y la memoria y otros chips, se pueden conseguir unos 100 millones de transistores. No se pueden comprar 100 millones de nada por tan poco dinero. Esa cantidad de papel higiénico costaría más de 100.000 dólares.

Por tanto, tenemos la potencia del computador prácticamente gratis. Esta continua revolución tecnológica ha habilitado el desarrollo de una sorprendente complejidad y potencia. Por ejemplo, las aplicaciones de oficina que requieren la mayor potencia de los sistemas de hoy en día basados en microprocesadores incluyen:

- Procesamiento de imágenes
- Reconocimiento del habla
- Vídeo-conferencias
- Aplicaciones multimedia
- Almacenamiento de ficheros de voz y vídeo

Las estaciones de trabajo soportan ahora aplicaciones de ingeniería y ciencia altamente sofisticadas, así como simulaciones, y pueden aplicar los principios de trabajo en grupo a aplicaciones de imagen y vídeo. Además, en los negocios se está confiando en la creciente potencia de los servidores para manejar transacciones y procesamiento de bases de datos, y para soportar redes cliente-servidor masivas, que han reemplazado a los gigantescos centros de computadores de antaño.

Lo más fascinante de todo esto, desde la perspectiva de la organización y arquitectura de computadores, es que, por una parte, los bloques básicos de los portentosos computadores de hoy en día son prácticamente los mismos que los del computador IAS de hace casi 50 años, mientras que, por otra parte, las técnicas para sacar hasta la última gota del rendimiento de los elementos disponibles se han vuelto cada vez más sofisticadas.

Esta observación sirve de guía principal para la presentación de este libro. A medida que avanzamos en los distintos elementos y componentes de un computador, se persiguen dos objetivos. Primero, el libro explica la funcionalidad fundamental en cada área que se considera, y segundo, el libro explora las técnicas requeridas para conseguir el máximo de prestaciones. En el resto de esta sección, destacamos algunos de los factores que hay tras la necesidad de diseñar para obtener mejores prestaciones.

VELOCIDAD DEL MICROPROCESADOR

Lo que le da al Pentium o al PowerPC esa increíble potencia es la persecución sin descanso de la velocidad por parte de los fabricantes del procesador. La evolución de estas máquinas continúa confirmando lo que se conoce como ley de Moore. El presidente de Intel Gordon Moore observó a mediados de los 60 que, reduciendo el tamaño de las delgadas líneas que formaban los circuitos del transistor en silicio alrededor de un 10% al año, los fabricantes de chips podrían crear una nueva generación de chips cada tres años (con el cuádruple de transistores). En chips de memoria esto ha cuadruplicado, cada tres años, la capacidad de las memorias dinámicas de acceso aleatorio (DRAM), que son aún la tecnología básica de la memoria principal de un computador. En microprocesadores, la adición de nuevos circuitos, y la potenciación de la velocidad, que proviene de la reducción de las distancias entre ellos, ha conseguido cuadruplicar o quintuplicar las prestaciones cada tres años desde que Intel lanzó su familia X86 en 1979.

Pero la velocidad bruta del procesador no alcanzará su potencial a menos que se le alimente con una corriente constante de trabajo en forma de instrucciones del computador. Cualquier cosa que se interponga en el camino de esta corriente limita la potencia del procesador. Conforme a esto, mientras que los fabricantes de chips han estado ocupados aprendiendo cómo se fabrican chips de densidad cada vez mayor, los diseñadores del procesador tienen que producir técnicas cada vez más elaboradas para «alimentar al monstruo». Entre las técnicas incorporadas a los procesadores de hoy en día están:

- **Predicción de ramificación:** el procesador se anticipa al software y predice qué ramas, o grupos de instrucciones, se van a procesar después con mayor probabilidad. Si el procesador acierta la mayoría de las veces, puede precaptar las instrucciones correctas y al-

macenarlas para mantener al procesador ocupado. Los ejemplos más sofisticados de esta estrategia predicen no sólo la siguiente rama, sino también varias ramas. Por tanto, la predicción de ramificación incrementa la cantidad de trabajo disponible que el procesador tiene que ejecutar.

- **Análisis de flujo de datos:** el procesador analiza qué instrucciones dependen de los resultados de otras instrucciones o datos, para crear una organización optimizada de instrucciones. De hecho, las instrucciones se regulan para ser ejecutadas cuando estén listas, independientemente del orden original del programa. Esto evita retrasos innecesarios.
- **Ejecución especulativa:** utilizando la predicción de ramificación y el análisis de flujo de datos, algunos procesadores ejecutan especulativamente instrucciones antes de que aparezcan en la ejecución del programa, manteniendo los resultados en posiciones temporales. Esto permite al procesador mantener sus máquinas de ejecución tan ocupadas como sea posible, ejecutando instrucciones que es probable que se necesiten.

Estas y otras sofisticadas técnicas se hacen necesarias simplemente porque el procesador es muy potente. Estas técnicas hacen posible explotar la potencia bruta del procesador.

EQUILIBRIO DE PRESTACIONES

Mientras que la velocidad del procesador ha crecido con increíble rapidez, otros componentes esenciales del computador no lo han hecho tan rápido. El resultado de esto es que ahora hace falta prestar atención al equilibrio de las prestaciones: ajustar la organización y la arquitectura para compensar las desigualdades de capacidad entre los distintos componentes.

El problema creado por tales desigualdades no es en ningún lugar tan grave como en la interfaz entre el procesador y la memoria principal. Observemos la historia representada en la Figura 2.11. Mientras la velocidad del procesador y la capacidad de la memoria han crecido rápidamente, la velocidad con la que los datos pueden ser transferidos entre la memoria

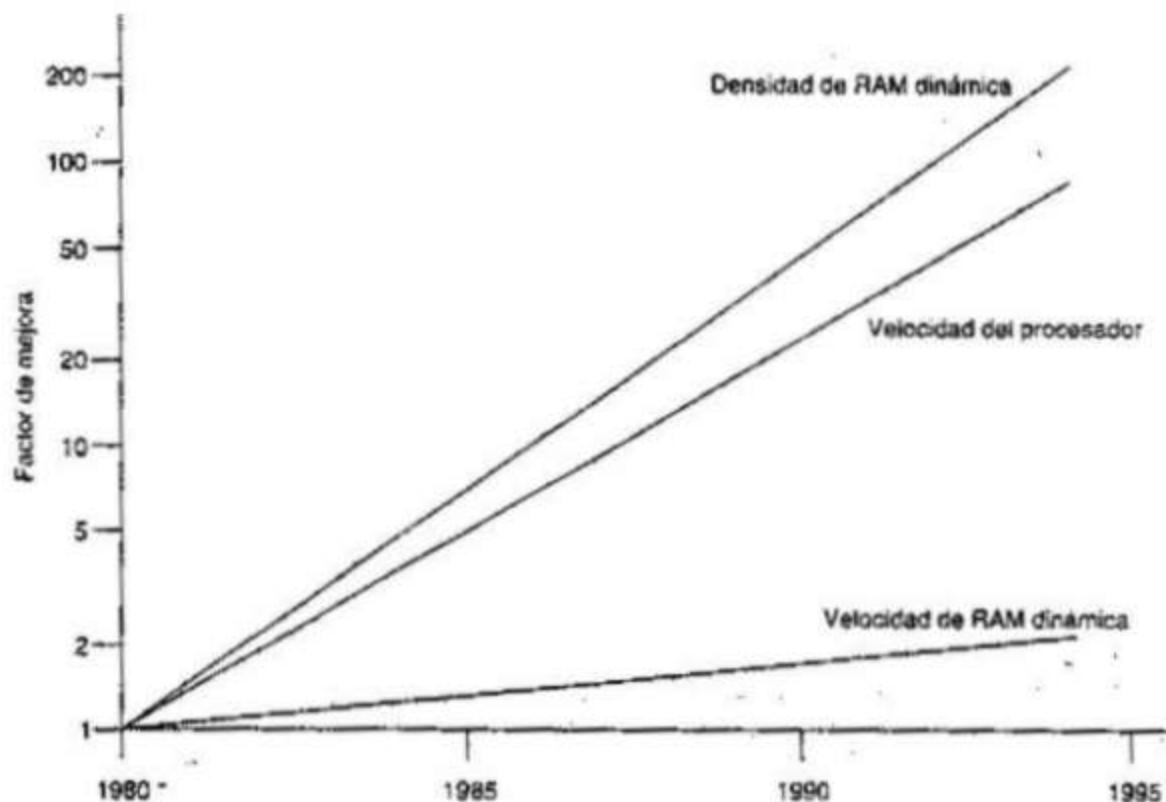


Figura 2.11. Evolución de las características de DRAM y del procesador.

principal y el procesador se ha quedado dramáticamente retrasada. La interfaz entre el procesador y la memoria principal es el camino más importante de todo el computador, ya que es el responsable de llevar el constante flujo de instrucciones y datos entre los chips de la memoria y el procesador. Si la memoria o la interfaz no logran mantener el ritmo de las insistentes demandas del procesador, éste se estanca en una posición de espera, y se pierde así tiempo de procesamiento valioso.

Los efectos de estas tendencias se muestran claramente en la Figura 2.12. La cantidad de memoria que se necesita está creciendo, pero la densidad de las DRAM está creciendo más rápido. El resultado es que, de media, el número de DRAM por sistema está bajando. Las líneas negras gruesas muestran que, para una memoria de tamaño constante, el número de DRAM que se necesitan está bajando. Pero esto tiene una consecuencia en la transferencia de datos, porque con menos DRAM hay menos oportunidad de transferencias paralelas de datos. Las bandas sombreadas muestran que, para un tipo de sistema determinado, el tamaño de la memoria principal ha aumentado lentamente, mientras que el número de DRAM ha bajado.

Hay varias maneras de que una arquitectura pueda atacar este problema, y todas se reflejan en los diseños de computadores contemporáneos. He aquí algunos ejemplos:

- Incrementar el número de bits que se recuperan de una sola vez haciendo las DRAM más «anchas» en lugar de más «profundas» utilizando buses de datos más anchos.
- Cambiar la interfaz DRAM para hacerla más eficiente, incluyendo una cache u otro esquema de almacenamiento temporal en el chip DRAM.
- Reducir la frecuencia del acceso a memoria incorporando, entre el procesador y la memoria principal, caches cada vez más complejas y eficientes. Esto incluye la incorporación de una o más caches en el chip del procesador, así como una cache fuera del chip cerca del procesador.

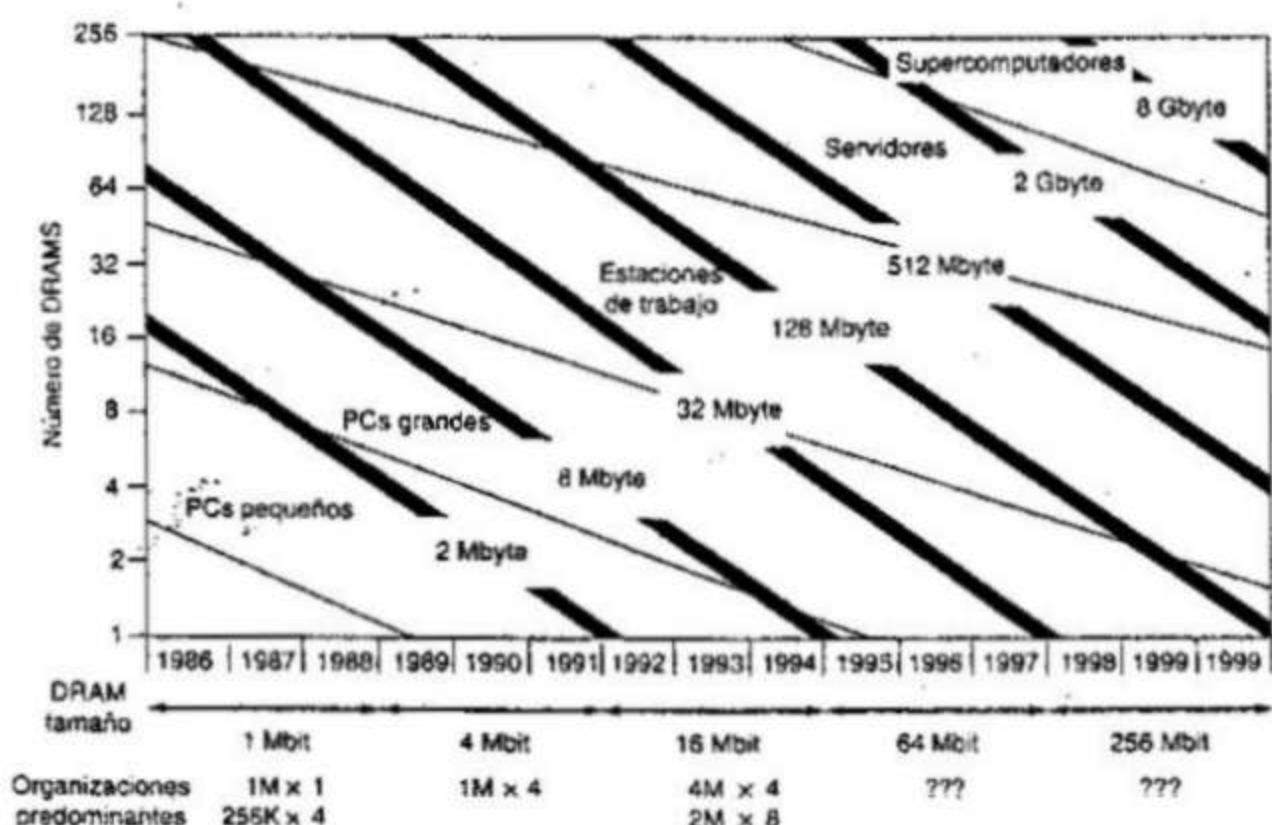


Figura 2.12. Tendencias en el uso de DRAM (PRZY94).

Tabla 2.7. Requerimientos típicos de ancho de banda para distintas tecnologías de periféricos

Periférico	Tecnología	Ancho de banda requerido
Gráficos	color de 24 bits	30 MBytes/seg
Red de área local	100BASEX o FDDI	12 MBytes/seg
Controlador de disco	SCSI o P1394	10 MBytes/seg
Vídeo de movimiento completo	1024 × 768@30fps	67 + MBytes/seg
Periférico de E/S	Otras varias	5 + MBytes/seg

- Incrementar el ancho de banda entre el procesador y la memoria usando buses de más alta velocidad y una jerarquía de buses para almacenar y estructurar el flujo de datos.

Otra área de diseño se centra en el manejo de dispositivos de E/S. Conforme los computadores se hacen más rápidos y potentes, se desarrollan aplicaciones más sofisticadas, que se apoyan en el uso de periféricos con demandas intensivas de E/S. La Tabla 2.7 muestra algunos ejemplos de los dispositivos periféricos típicos que se usan en ordenadores personales y estaciones de trabajo. Estos dispositivos crean una tremenda demanda de procesamiento de datos. La generación actual de procesadores puede manejar los datos producidos por esos dispositivos, pero aún queda el problema de mover esos datos entre el procesador y los periféricos. Las estrategias en relación con esto incluyen esquemas de caches y almacenamiento, más el uso de buses de interconexión de más alta velocidad y con estructuras más elaboradas. Además, el uso de configuraciones multiprocesador puede ayudar a satisfacer las demandas de E/S.

La clave en todo esto es el equilibrio. Los diseñadores luchan constantemente por alcanzar el equilibrio en la demanda de rendimiento y procesamiento por parte de los componentes del procesador, la memoria principal, los dispositivos de E/S y de las estructuras de interconexión. Y este diseño tiene que ser constantemente replanteado para hacer frente a dos factores en continua evolución:

- La velocidad a la que el rendimiento está cambiando en las distintas áreas tecnológicas (procesador, buses, memoria y periféricos) difiere enormemente de un tipo de elemento a otro.
- Las nuevas aplicaciones y nuevos dispositivos periféricos cambian constantemente la naturaleza de la demanda en el sistema en cuanto al perfil de instrucción típico y el modelo de acceso de datos.

Así, el diseño de computadores es una forma de arte en constante evolución. Este libro intenta plantear los fundamentos en los que se basa esta forma de arte y dar una visión general de su estado actual.

2.3. EVOLUCIÓN DEL PENTIUM Y DEL PowerPC

A lo largo de este libro nos basamos en muchos ejemplos concretos de diseño e implementación de computadores para ilustrar conceptos y aclarar los compromisos. El libro se basa, la mayor parte de las veces, en ejemplos de dos familias de computadores: el Pentium de Intel y el PowerPC. El Pentium representa el resultado de décadas de esfuerzo de diseño en computadores de repertorio complejo de instrucciones (CISC). Incorpora los sofisticados principios de diseño que antes se encontraban sólo en ordenadores grandes y supercomputadores, y es un excelente ejemplo de diseño CISC. El PowerPC es descendiente directo del primer sistema

RISC, el IBM 801, y es uno de los sistemas basados en RISC más potentes y mejor diseñados del mercado.

En esta sección damos una breve visión general de ambos sistemas.

PENTIUM

Intel ha sido el número uno de los fabricantes de microprocesadores durante décadas, una posición que no parece probable que abandone. La evolución de su microprocesador más representativo es un buen indicador de la evolución de la tecnología de computadores en general.

La Tabla 2.6 muestra esta evolución. Aunque el Pentium es ahora la estrella de la línea de productos Intel, ya tienen dos procesadores en marcha: el P6, presentado en 1995 y el P7, aún en desarrollo. Conforme los microprocesadores se han hecho mucho más rápidos y complejos, Intel ha mejorado el ritmo. Antes, Intel solía desarrollar los microprocesadores, uno tras otro, cada cuatro años. Pero para el Pentium, el intervalo generacional se redujo a tres años. Y para sus chips P6 y P7, Intel espera mantener a sus rivales a raya reduciéndolo otro año más.

Merece la pena enumerar algunos de los rasgos más destacados de la evolución de los productos Intel:

- 8080: es el primer microprocesador de propósito general del mundo. Era una máquina de 8 bits, con datos de memoria de 8 bits.
- 8086: una máquina de 16 bits, mucho más potente. Además de un camino de datos más ancho y registros más grandes, el 8086 tenía una cache de instrucción, o cola, que pre-captaba algunas instrucciones antes de ser ejecutadas.
- 80286: esta ampliación del 8086 permitía direccionar una memoria de 16 MBytes en lugar de sólo 1 MByte.
- 80386: fue la primera máquina de Intel con 32 bits, y constituyó una gran revisión del modelo anterior. Con una arquitectura de 32 bits, el 80386 rivalizaba en complejidad y potencia con los minicomputadores y grandes computadores introducidos en el mercado pocos años antes.
- 80486: el 80486 introduce el uso de tecnología de cache mucho más sofisticada y potente, e instrucciones de segmentación de cauce sofisticadas.
- Pentium: con el Pentium, Intel introduce el uso de técnicas superescalares, que permiten que varias instrucciones se ejecuten en paralelo.
- Pentium Pro: El Pentium Pro continuó la tendencia iniciada con el Pentium hacia la organización superescalar, con el uso agresivo del renombrado de registros, predicción de ramificaciones, análisis del flujo de datos y ejecución especulativa.
- Pentium II: en el Pentium II se incorporó la tecnología Intel MMX, que se diseñó específicamente para procesar de forma eficiente datos de video, audio y gráficos.
- Pentium III: el Pentium III incorporó instrucciones adicionales en punto flotante para procesar software de gráficos 3D.
- Merced: esta nueva generación de procesadores Intel usa una organización de 64 bits.

PowerPC

En 1975, el proyecto de minicomputador 801 de IBM fue el primero en muchos de los conceptos de arquitectura usados en sistemas RISC. El 801, junto con el procesador RISC I de Berkeley, comenzó con el movimiento RISC. El 801, sin embargo, era simplemente un prototipo que intentaba demostrar los conceptos de diseño. El éxito del proyecto 801 permitió a IBM desarrollar una estación de trabajo RISC comercial, el RT PC. El RT PC, introducido en 1986, adaptaba los conceptos arquitectónicos del 801 a un producto real. El RT PC no tuvo éxito comercial, y tuvo muchos rivales con prestaciones comparables o mejores. En 1990, IBM produjo un tercer sistema, que incorporaba las lecciones aprendidas con el 801 y el RT PC. El IBM RISC System/6000 era una máquina superescalar RISC comercializada como una estación de trabajo de altas prestaciones; poco después de su introducción, IBM comenzó a llamarla «arquitectura POWER».

Como siguiente paso, IBM se alió con Motorola, que había desarrollado las series 68000 de microprocesadores, y Apple, que usaba el chip de Motorola en sus computadoras Macintosh. El resultado es una serie de máquinas que implementan la arquitectura PowerPC. Esta arquitectura deriva de la arquitectura Power (Figura 2.13). Se hicieron cambios para añadir características claves que no estaban y para permitir una implementación más eficiente, eliminando algunas instrucciones y relajando la especificación, para eliminar casos especialmente problemáticos. La arquitectura PowerPC resultante es un sistema RISC superescalar. El PowerPC se usa en millones de máquinas Apple Macintosh y en sistemas con microprocesadores embebidos. Como ejemplo de este último hecho puede citarse la familia IBM de chips de gestión de redes, que se utilizan embebidos en numerosos equipos que proporcionan la gestión del acceso a red usual a los usuarios con sistemas de distintos fabricantes.

Hasta ahora se han presentado cuatro miembros de la familia PowerPC (Tabla 2.8):

- 601: el propósito del 601 era llevar la arquitectura PowerPC al mercado lo más rápidamente posible. El 601 es una máquina de 32 bits.
- 603: pensado para computadores portátiles y de sobremesa. Es también una máquina de 32 bits, comparable en prestaciones con el 601, pero con costo más bajo e implementación más eficiente.
- 604: pensado para computadores de sobremesa y servidores finales. De nuevo es una máquina de 32 bits pero utiliza técnicas de diseño superescalares, mucho más avanzadas, para lograr mayores prestaciones.

Tabla 2.8. Resumen de Procesadores PowerPC

	601	603/603e	604/604e	740/750 (G3)	G4
fe comercialización	1993	1994	1994	1997	1999
dad de reloj (MHz)	50-120	100-300	168-350	200-386	500
L1	—	16 kbyte inst. 16 kbyte datos	32 kbyte inst. 32 kbyte datos	32 kbyte inst. 32 kbyte datos	32 kbyte inst. 32 kbyte datos
do con cache L2	—	—	—	256 kbyte-1 Mbyte	256 kbyte-1 Mbyte
transistores (10^6)	2,8	1,8-2,6	3,6-5,1	6,35	

- 620: pensado para servidores finales. El primer miembro de la familia PowerPC que implementa una arquitectura completa de 64 bits, incluyendo registros y buses de datos de 64 bits.
- 740/750: también conocido como procesador G3. Este procesador integra dos niveles de cache en el chip del procesador principal, ofreciendo mejoras significativas en las prestaciones sobre otras máquinas comparables con organización de cache fuera del chip.
- G4: este procesador continuará incrementando el paralelismo y la velocidad interna del chip del procesador.

LECTURAS Y SITIOS WEB RECOMENDADOS

Se puede encontrar una descripción de la serie IBM 7000 en [BELL71a]. Hay un buen tratamiento del IBM 360 en [SIEW82] y del PDP-8 y otras máquinas DEC en [BELL78a]. Estos tres libros también contienen numerosos y detallados ejemplos de otros computadores que abarcan la historia de los computadores de principios de los años 80. Un libro más reciente, que incluye un excelente conjunto de estudios sobre máquinas históricas, se encuentra en [BLAA97]. En [BETK97] se incluye una buena historia de los microprocesadores.

Uno de los mejores tratamientos del Pentium se encuentra en [SHAN98]. La propia documentación de Intel también es buena [INTE98a, INTE98b]. [BREY97] proporciona una buena visión global de la línea de microprocesadores Intel, centrándose en las máquinas de 32 bits.

[IBM94] hace un estudio intensivo de la arquitectura PowerPC. [SHAN95] hace un estudio similar más una descripción del 601. [WEIS94] trata las arquitecturas Power y PowerPC.

Ver [HUTC96], [SCHA97], y [BOHR98] para discusiones interesantes sobre la ley de Moore y sus consecuencias.

BELL71a Bell, C., y Newell, A. *Computer Structures: Readings and Examples*. New York: McGraw-Hill, 1971.

BELL78a Bell, C.; Mudge, J.; y McNamara, J. *Computer Engineering: A DEC View of Hardware Systems Design*. Bedford, MA: Digital Press, 1978.

BETK97 Betker, M.; Fernando, J.; y Whalen, S. «The History of the Microprocessor.» *Bell Labs Technical Journal*, Autumn, 1997.

BLAA97 Blaauw, G., y Brooks, F. *Computer Architecture: Concepts and Evolution*. Reading, MA: Addison-Wesley, 1997.

BOHR98 Bohr, M. «Silicon Trends and Limits for Advanced Microprocessors». *Communications of the ACM*, March, 1998.

BREY97 Brey, B. *The Intel Microprocessors: 8086/8066, 80186/80188, 80286, 80386, 80486, Pentium, and Pentium Processor*. Upper Saddle River, NJ: Prentice Hall, 1997.

HUTC96 Hutcheson, G., y Hutcheson, J. «Technology and Economics in the Semiconductor Industry.» *Scientific American*, January, 1996.

IBM94 International Business Machines, Inc. *The PowerPC Architecture: A Specification for a New Family of RISC Processors*. San Francisco, CA: Morgan Kaufmann, 1994.

INTE98a Intel Corp. *Pentium Processors and Related Products*. Aurora, CO, 1998.

- INTE98b Intel Corp. *Pentium Pro and Pentium II Processors and Related Products*. Aurora, CO, 1998.
- SCHA97 Schaller, R. «Moore's Law: Past, Present, and Future.» *IEEE Spectrum*, June, 1997.
- SHAN98 Shanley, T. *Pentium Pro and Pentium II System Architecture*. Reading, MA: Addison-Wesley, 1998.
- SHAN95 Shanley, T. *PowerPC System Architecture*. Reading, MA: Addison-Wesley, 1995.
- SIEW82 Siewiorek, D.; Bell, C.; y Newell, A. *Computer Structures: Principles and Examples*. New York: McGraw-Hill, 1982.
- WEIS94 Weiss, S., y Smith, J. *POWER and PowerPC*. San Francisco: Morgan Kaufmann, 1994.



SITIOS WEB RECOMENDADOS:

- **Instituto Charles Babbage:** proporciona enlaces a numerosos sitios Web sobre la historia de los computadores.
- **PowerPC:** página principal de IBM para el PowerPC.
- **Developer Home:** página Web de Intel de desarrollos; es un punto de inicio para acceder a la información sobre Pentium.

2.5. PROBLEMAS

- 2.1. Sean $A = A(1), A(2), \dots, A(1.000)$ y $B = B(1), B(2), \dots, B(1.000)$ dos vectores (unidimensionales) que comprenden 1.000 números cada uno, que van a ser sumados para formar un vector C , tal que $C(l) = A(l) + B(l)$, donde $l = 1, 2, \dots, 1.000$. Usando el conjunto de instrucciones IAS, escribir un programa para resolver este problema.
- 2.2. En el IBM 360 modelos 65 y 75, las direcciones están situadas en dos unidades de memoria principal separadas (por ejemplo, todas las palabras pares en una unidad y todas las impares en otra). ¿Cuál puede ser el propósito de esta técnica?

PARTE II

EL COMPUTADOR

CUESTIONES A TRATAR EN LA SEGUNDA PARTE

Un computador consta de procesador, memoria, E/S, y las interconexiones entre estos componentes principales. Con la excepción del procesador, que es suficientemente complejo para que dediquemos a su estudio toda la Parte III, la Parte II examina cada uno de estos componentes con detalle. Dos temas tratan sobre los aspectos correspondientes a esta segunda parte.

ESQUEMA DE LA SEGUNDA PARTE

CAPÍTULO 3. BUSES DEL SISTEMA

A alto nivel, podemos describir un computador a partir del funcionamiento de cada uno de sus componentes principales, la estructura de sus interconexiones y el tipo de señales que intercambian entre ellos. El Capítulo 3 se centra en la estructura de las interconexiones y en el intercambio de señales que se realiza a través de dicha estructura. La interconexión de los principales componentes de un computador se realiza usualmente a través de uno o varios buses, y este es el tema del Capítulo 3. El capítulo también considera los aspectos clave que afectan al diseño de las conexiones, especialmente el soporte que requieren las interrupciones.

CAPÍTULO 4. MEMORIA INTERNA

El Capítulo 4 considera la organización de la memoria principal y el uso de memoria cache para aumentar las prestaciones. El diseño del sistema de memoria principal constituye una guerra sin cuartel entre tres requisitos de diseño: capacidad de almacenamiento elevada, tiempo de acceso reducido y bajo coste. A medida que evoluciona la tecnología de las memorias, cada uno de estos aspectos cambia, de forma que las decisiones en el diseño de la organización de la memoria principal deben reconsiderarse en función de las nuevas implementa-

ciones. Dos áreas de particular interés, en las que el Capítulo 4 hace especial énfasis, son: la organización de la cache y los distintos esquemas de memoria RAM dinámica (DRAM).

CAPÍTULO 5. MEMORIA EXTERNA

Para disponer de una capacidad de almacenamiento verdaderamente elevada, y conseguir un almacenamiento más duradero que el que proporciona la memoria principal, se necesita la memoria externa. El tipo de memoria externa más ampliamente utilizado es el disco magnético, de forma que la mayor parte del Capítulo 5 se centra en dicho tema. En primer lugar, se analiza la tecnología de los discos magnéticos y las consideraciones de diseño. A continuación, se estudia el uso de la organización RAID para mejorar las prestaciones de la memoria de disco. El Capítulo 5 también examina el almacenamiento óptico y las cintas magnéticas.

CAPÍTULO 6. ENTRADA/SALIDA

El Capítulo 6 se dedica a distintos aspectos de la organización de E/S. Se trata de un área compleja, y mucho menos comprendida que otras áreas del diseño de un computador en lo que se refiere a cómo satisfacer los niveles de prestaciones exigidos. El Capítulo 6 examina los mecanismos a través de los cuales un módulo de E/S interactúa con el resto del computador, utilizando las técnicas de E/S programada, E/S por interrupciones y acceso directo a memoria (DMA). También se describe la interfaz entre los módulos de E/S y los dispositivos externos.

CAPÍTULO 7. EL SOPORTE DEL SISTEMA OPERATIVO

Un examen detallado de los sistemas operativos está fuera del ámbito de este libro. No obstante, es importante entender sus funciones básicas y cómo aprovecha el hardware para proporcionar el nivel de prestaciones deseado. El Capítulo 7 describe los principios básicos de los sistemas operativos, y discute las características de diseño específicas del hardware del computador orientadas a dar soporte al sistema operativo.

CAPÍTULO 3

Buses del sistema

3.1. Componentes del computador

3.2. Funcionamiento del computador

Los ciclos de captación y ejecución
Interrupciones
Funcionamiento de las E/S

3.3. Estructuras de interconexión

3.4. Interconexión con buses

Estructura del bus
Jerarquías de buses
Elementos de diseño de un bus

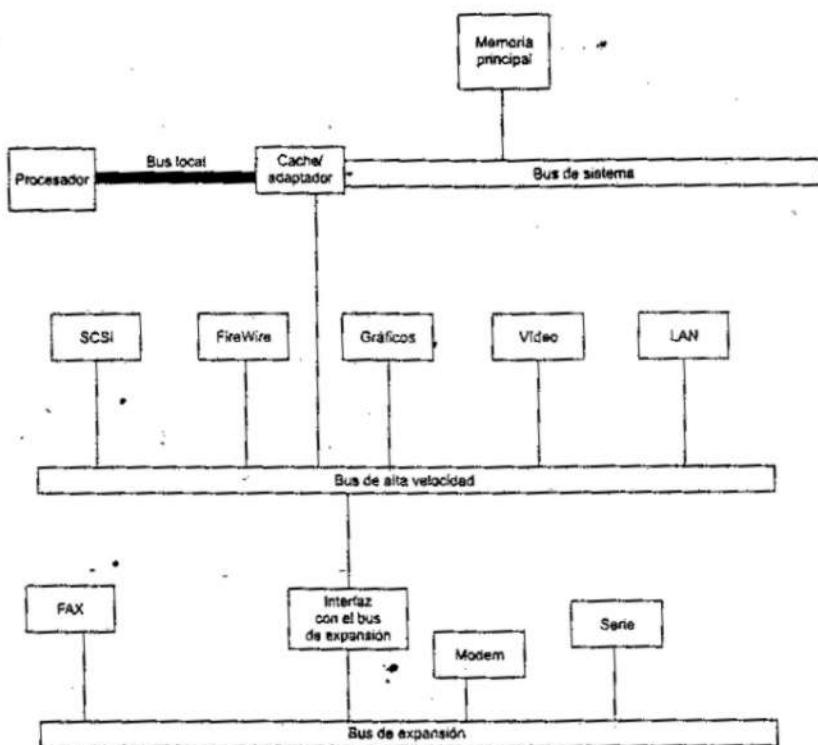
3.5. PCI

Estructura del bus
Órdenes del PCI
Transferencias de datos
Arbitraje

3.6. Lecturas y sitios Web recomendados

3.7. Problemas

Apéndice 3A. Diagramas de tiempo



- Un ciclo de instrucción consiste en la captación de la instrucción, seguida de ninguno o varios accesos a operandos, ninguno o varios almacenamientos de operandos, y la comprobación de las interrupciones (si éstas están habilitadas).
- Los principales componentes del computador (procesador, memoria principal y módulos de E/S) necesitan estar interconectados para intercambiar datos y señales de control. El medio de interconexión más popular es un bus compartido, constituido por un conjunto de líneas. En los computadores actuales, es usual utilizar una jerarquía de buses para mejorar el nivel de prestaciones.
- Los aspectos clave del diseño de los buses son: el arbitraje (si el permiso para enviar las señales a través de las líneas del bus se controla de forma centralizada o distribuida); la temporización (si las señales del bus se sincronizan mediante un reloj central o se envían asincrónicamente); y la anchura (número de líneas de dirección y de datos).

Alto nivel, un computador está constituido por CPU, memoria, y unidades de E/S, con uno o varios módulos de cada tipo. Estos componentes se interconectan de modo que se pueda llevar a cabo la función básica del computador, que es ejecutar programas. Así, a este nivel, se puede describir un computador (1) mediante el comportamiento de cada uno de sus componentes, es decir, mediante los datos y las señales de control que un componente intercambia con los otros, y (2) mediante la estructura de interconexión y los controles necesarios para gestionar el uso de dicha estructura.

Esta visión de alto nivel en términos de estructura y funcionamiento es importante, debido a su capacidad explicativa de cara a la comprensión de la naturaleza del computador. Igualmente importante es su utilidad para entender los cada vez más complejos problemas de evaluación de prestaciones. Entender la estructura y el funcionamiento a alto nivel permite hacerse una idea de los cuellos de botella del sistema, las soluciones alternativas, la importancia de los fallos del sistema si hay un componente defectuoso, y la facilidad con que se pueden mejorar las prestaciones. En muchos casos, los requisitos de mayor potencia y capacidad de funcionamiento tolerante ante los fallos se satisfacen mediante cambios en el diseño, más que con un incremento en la velocidad y en la fiabilidad de los componentes individuales.

Este capítulo se centra en las estructuras básicas utilizadas para la interconexión de los componentes del computador. A modo de revisión, el capítulo comienza con un somero examen de los componentes básicos y sus necesidades de interconexión. Después, se revisan los aspectos funcionales.

Después se examinará el uso de los buses que interconectan los componentes del sistema.

COMPONENTES DE UN COMPUTADOR

Como se discutió en el Capítulo 2, prácticamente todos los computadores actuales se han diseñado basándose en los conceptos desarrollados por John von Neumann en el Instituto de Estudios Avanzados (Institute for Advances Studies) de Princeton. Tal diseño se conoce con el nombre de *arquitectura de von Neumann*, y se basa en tres conceptos clave:

- Los datos y las instrucciones se almacenan en una sola memoria de lectura-escritura.
- Los contenidos de esta memoria se direccionan indicando su posición, sin considerar el tipo de dato contenido en la misma.
- La ejecución se produce siguiendo una secuencia de instrucción tras instrucción (a no ser que dicha secuencia se modifique explícitamente).

Las razones que hay detrás de estos conceptos se discutieron en el Capítulo 1, pero merecen ser resumidas aquí. Hay un conjunto pequeño de componentes lógicos básicos, que pueden combinarse de formas diferentes para almacenar datos binarios y realizar las operaciones aritméticas y lógicas con esos datos. Si se desea realizar un cálculo concreto, es posible utilizar una configuración de componentes lógicos diseñada específicamente para dicho cálculo. Se puede pensar en el proceso de conexión de los diversos componentes para obtener la configuración deseada, como si se tratase de una forma de programación. El «programa» resultante es hardware y se denomina *programa cableado* (*hardwired program*).

Considérese ahora la siguiente alternativa. Se construye una configuración de uso general de funciones lógicas y aritméticas. Este hardware realizará funciones diferentes según las señales de control aplicadas. En el caso del hardware específico, el sistema acepta datos y produce resultados (Figura 3.1a). Con el hardware de uso general, el sistema acepta datos y señales de control, y produce resultados. Así, en lugar de reconfigurar el hardware para cada

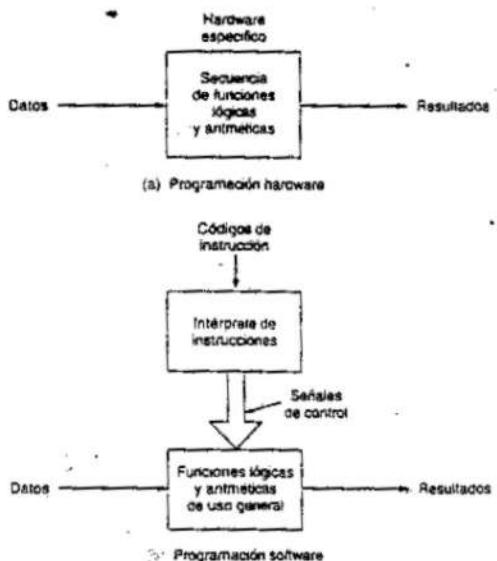


Figura 3.1. Alternativas hardware y software.

nuevo programa, el programador simplemente necesita proporcionar un nuevo conjunto de señales de control.

¿Cómo se suministran las señales de control? La respuesta es simple, pero ingeniosa. El programa es realmente una secuencia de pasos. En cada paso, se realiza una operación aritmética o lógica con ciertos datos. Para cada paso, se necesita un nuevo conjunto de señales de control. La solución consiste en asociar un código específico a cada posible conjunto de señales de control, y añadir al hardware de uso general una parte encargada de generar las señales de control a partir del código (Figura 3.1b).

Programar es ahora mucho más fácil. En lugar de tener que reconfigurar el hardware para cada programa, todo lo que se necesita es proporcionar una nueva secuencia de códigos. Cada código es, de hecho, una instrucción, y una parte del hardware interpreta cada instrucción y genera las señales de control. Para distinguir este nuevo método de programación, una secuencia de códigos o instrucciones se denomina *software*.

La Figura 3.1b muestra dos componentes esenciales del sistema: un intérprete de instrucciones y un módulo de uso general para las funciones aritmáticas y lógicas. Estos dos elementos constituyen la CPU. Se requieren varios componentes adicionales para que el computador pueda funcionar. Los datos y las instrucciones deben introducirse en el sistema. Para eso se necesita algún tipo de módulo de entrada. Este módulo contiene los componentes básicos para captar datos e instrucciones en cierto formato y traducirlos al formato de señales que utiliza el sistema. Se necesita un medio para proporcionar los resultados, el módulo de salida. Globalmente, estos módulos se conocen con el nombre de *componentes de E/S* (entrada/salida).

Se necesita un componente más. Un dispositivo de entrada proporcionará los datos y las instrucciones secuencialmente, uno tras otro. Pero un programa no siempre ejecuta las instrucciones según la misma secuencia; puede saltarse ciertas instrucciones (por ejemplo, al ejecutar la instrucción de salto iAS). De la misma forma, las operaciones con datos pueden necesitar acceder a más de un operando y según una secuencia determinada. Por ello, debe existir un sitio para almacenar temporalmente, tanto las instrucciones como los datos. Ese módulo se llama *memoria*, o *memoria principal* para distinguirlo de los periféricos y la memoria externa. Von Neumann indicó que la misma memoria podría ser usada tanto para las instrucciones como para los datos.

La Figura 3.2 muestra estos componentes de alto nivel y sugiere las interacciones entre ellos. Típicamente, la CPU se encarga del control. Intercambia datos con la memoria. Para ello, usualmente utiliza dos registros internos (en la CPU): un registro de direcciones de memoria (MAR, Memory Address Register), que especifica la dirección en memoria de la próxima lectura o escritura, y un registro para datos de memoria (MBR, Memory Buffer Register), que contiene el dato que se va a escribir en memoria o donde se escribe el dato que se va a leer de memoria. Igualmente, un registro de direcciones de E/S (E/SAR, E/S Address Register) especifica un dispositivo de E/S. Un registro para datos de E/S (E/S BR, E/S Buffer Register) se utiliza para intercambiar datos entre un módulo de E/S y la CPU.

Un módulo de memoria consta de un conjunto de posiciones, designadas por direcciones numeradas secuencialmente. Cada posición contiene un número binario que puede ser interpretado como una instrucción o como un dato. Un módulo de E/S transfiere datos desde los dispositivos externos a la CPU y a la memoria, y viceversa. Contiene los registros («buffers») internos para almacenar los datos temporalmente, hasta que puedan enviarse.

Tras esta breve descripción de los principales componentes, revisaremos como funcionan éstos cuando ejecutan programas.

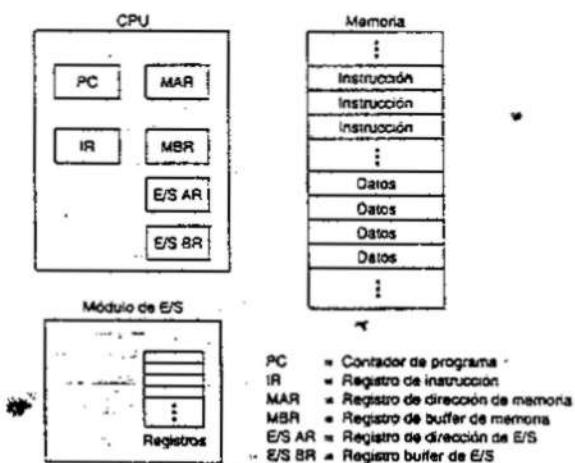


Figura 3.2. Componentes del computador: esquema de dos niveles.

3.2. FUNCIONAMIENTO DEL COMPUTADOR

La función básica que realiza un computador es la ejecución de un programa, constituido por un conjunto de instrucciones almacenadas en memoria. El procesador es, precisamente, el que se encarga de ejecutar las instrucciones especificadas en el programa. Esta sección proporciona una revisión de los aspectos claves en la ejecución de un programa, que en su forma más simple consta de dos etapas: El procesador lee (capta) la instrucción de memoria, y la ejecuta. La ejecución del programa consiste en la repetición del proceso de captación de instrucción y ejecución de instrucción. Por supuesto, la ejecución de la instrucción puede a su vez estar compuesta de cierto número de pasos (obsérvese, por ejemplo, la parte inferior de la Figura 2.4).

El procesamiento que requiere una instrucción se denomina *ciclo de instrucción*. Se representa en la Figura 3.3 utilizando la descripción de dos etapas explicada más arriba. Los dos pasos se denotan como *ciclo de captación* y *ciclo de ejecución*. La ejecución del programa se para, sólo si la máquina se desconecta, se produce algún tipo de error, o ejecuta una instrucción del programa que detiene al computador.

LOS CICLOS DE CAPTACIÓN Y EJECUCIÓN

Al comienzo de cada ciclo de instrucción, la CPU capta una instrucción de memoria. En una CPU típica, se utiliza un registro llamado «contador de programa» (PC. Program Counter) para seguir la pista de la instrucción que debe captar a continuación. A no ser que se indique otra cosa, la CPU siempre incrementa el PC después de captar cada instrucción, de forma que captará la siguiente instrucción de la secuencia (es decir, la instrucción situada en la siguiente dirección de memoria). Considérese, por ejemplo, un computador en el que cada instrucción ocupa una palabra de memoria de 16 bits. Se supone que el contador de programa almacena el valor 300. La CPU captará la próxima instrucción almacenada en la posición 300. En los siguientes ciclos de instrucción, captará las instrucciones almacenadas en las posiciones 301, 302, 303, y así sucesivamente. Esta secuencia se puede alterar, como se explica luego.

La instrucción captada se almacena en un registro de la CPU conocido como «registro de instrucción» (IR. Instruction Register). La instrucción se escribe utilizando un código binario que especifica la acción que debe realizar la CPU. La CPU interpreta la instrucción y lleva a cabo la acción requerida. En general, ésta puede ser de cuatro tipos:

- **Procesador-memoria:** deben transferirse datos desde la CPU a la memoria, o desde la memoria a la CPU.
- **Procesador-E/S:** deben transferirse datos a o desde el exterior mediante transferencias entre la CPU y un módulo de E/S.
- **Procesamiento de datos:** la CPU ha de realizar alguna operación aritmética o lógica con los datos.

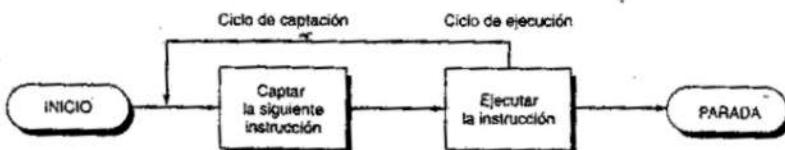


Figura 3.3. Ciclo de Instrucción básico.

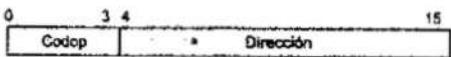
- **Control:** una instrucción puede especificar que la secuencia de ejecución se altere (como la instrucción de salto IAS, Tabla 2.1). Por ejemplo, la CPU capta una instrucción de la posición 149 que especifica que la siguiente instrucción debe captarse de la posición 182. La CPU registrará este hecho poniendo en el contador de programa 182. Así, en el próximo ciclo de captación, la instrucción se cargará desde la posición 182 en lugar de hacerlo desde la posición 150.

La ejecución de una instrucción puede implicar una combinación de estas acciones.

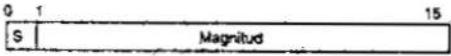
Considérese un ejemplo sencillo utilizando una máquina hipotética, que incluye las características enumeradas en la Figura 3.4. El procesador posee un único registro de datos, llamado *acumuladores* (AC). Tanto las instrucciones como los datos son de 16 bits. Así, es conveniente organizar la memoria utilizando posiciones de 16 bits, o palabras. El formato de instrucción indica que puede haber $2^4 = 16$ códigos de operación (codops) diferentes, y se pueden direccionar directamente hasta $2^{12} = 4096$ (4K) palabras de memoria.

La Figura 3.5 ilustra la ejecución de una parte de un programa, mostrando las partes relevantes de la memoria y los registros de la CPU. Se utiliza notación hexadecimal¹. El fragmento de programa suma el contenido de la palabra de memoria en la dirección 940_{16} con el contenido de la palabra de memoria en la dirección 941_{16} y almacena el resultado en esta última posición. Se requieren tres instrucciones, que consumen tres ciclos de captación y tres de ejecución:

1. El contador de programa (PC) contiene el valor 300, la dirección de la primera instrucción. Esta instrucción se carga en el registro de instrucción (IR). Obsérvese que este proceso implicaría el uso del registro de dirección de memoria (MAR) y el registro de datos de memoria (MBR). Por simplicidad, se han ignorado estos registros intermedios.
2. Los primeros cuatro bits de IR indican que el acumulador (AC) se va a cargar. Los restantes 12 bits especifican la dirección, que es 940.
3. El registro PC se incrementa, y se capta la siguiente instrucción.



(a) Formato de instrucción



(b) Formato de enteros

Contador de programa (PC) = Dirección de instrucción
 Registro de instrucción (IR) = Instrucción en ejecución
 Acumulador (AC) = Almacenamiento temporal

(c) Registros internos de la CPU

0001 = Cargar AC desde memoria.
 0010 = Almacenar AC en memoria.
 0101 = Sumar a AC un dato de memoria.

(d) Lista parcial de Codops («códigos de operación»)

Figura 3.4. Características de una máquina hipotética.

¹ En notación hexadecimal, cada dígito representa cuatro bits. Esta es la notación más conveniente para representar los contenidos de la memoria y los registros cuando la longitud de palabra es múltiplo de 4 (por ejemplo, 8, 16 o 32). Para los lectores no familiarizados con esta notación, se resume en el apéndice del Capítulo 3.

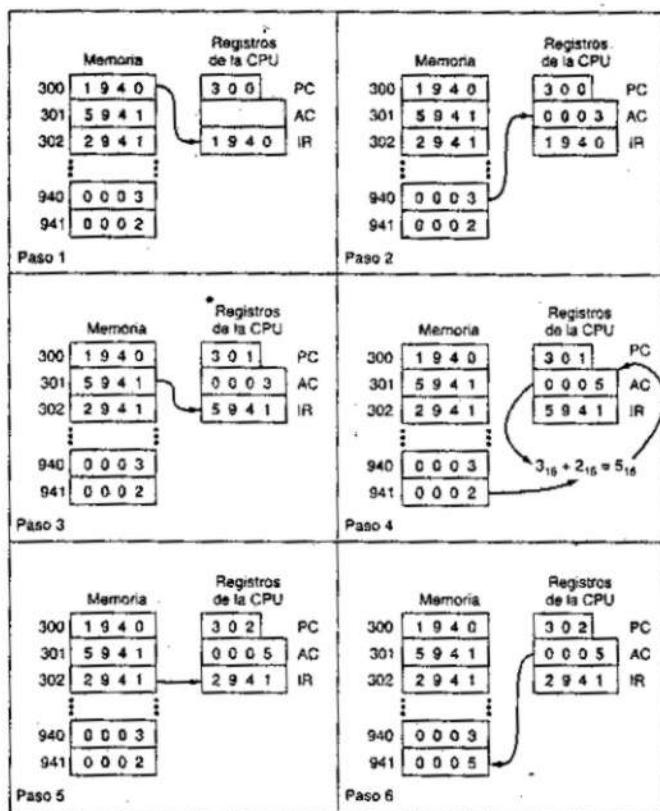


Figura 3.5. Ejemplo de ejecución de programa.

4. El contenido anterior de AC y el de la posición de memoria 941 se suman, y el resultado se almacena en AC.
5. El registro PC se incrementa, y se capta la siguiente instrucción.
6. El contenido de AC se almacena en la posición 941.

En este ejemplo, se necesitan tres ciclos de instrucción, cada uno con un ciclo de captación y un ciclo de ejecución, para sumar el contenido de la posición 940 y el contenido de la 941. Con un conjunto de instrucciones más complejo se hubieran necesitado menos ciclos. Los procesadores actuales incluyen instrucciones que contienen más de una dirección. Así, el ciclo de ejecución de una instrucción puede incluir más de una referencia a memoria. Además, en lugar de referencias a memoria, una instrucción puede especificar una operación de E/S.

Por ejemplo, la instrucción del PDP-11 expresada simbólicamente como ADD B,A almacena

cena la suma de los contenidos de las posiciones B y A en la posición de memoria A. Se produce un solo ciclo de instrucción con los siguientes pasos:

- Se capta la instrucción ADD.
- El contenido de la posición de memoria A se lee y pasa al procesador
- El contenido de la posición de memoria B se lee y pasa al procesador. Para que el contenido de A no se pierda, el procesador debe tener al menos dos registros para almacenar valores de memoria, en lugar de un solo acumulador.
- Se suman los dos valores.
- El procesador escribe el resultado en la posición de memoria A.

Así, el ciclo de ejecución de una instrucción particular puede ocasionar más de una referencia a memoria. Además, en lugar de referencias a memoria, una instrucción puede especificar una operación de E/S. Con estas consideraciones adicionales en mente, la Figura 3.6 proporciona una visión más detallada del ciclo de instrucción básico de la Figura 3.3. La figura tiene la forma de un diagrama de estados. Para un ciclo de instrucción dado, algunos estados pueden no darse y otros pueden visitarse más de una vez. Los estados se describen a continuación:

- **Cálculo de la dirección de la instrucción (iac, instruction address calculation):** determina la dirección de la siguiente instrucción a ejecutar. Normalmente, esto implica añadir un número fijo a la dirección de la instrucción previa. Por ejemplo, si las instrucciones tienen un tamaño de 16 bits y la memoria se organiza en palabras de 16 bits, se suma 1 a la dirección previa. En cambio, si la memoria se organiza en bytes (8 bits) direccionables individualmente, entonces hay que sumar 2 a la dirección previa.
- **Captación de instrucción (if, instruction fetch):** la CPU lee la instrucción desde su posición en memoria.
- **Decodificación de la operación indicada en la instrucción (iod, instruction operation decoding):** analiza la instrucción para determinar el tipo de operación a realizar y el/los operando(s) a utilizar.
- **Cálculo de la dirección del operando (oac, operand address calculation):** si la instrucción implica una referencia a un operando en memoria o disponible mediante E/S, determina la dirección del operando.



Figura 3.6. Diagrama de estados del ciclo de instrucción.

- **Captación de operando (of, operand fetch):** capta el operando desde memoria o se lee desde el dispositivo de E/S.
- **Operación con los datos (do, data operation):** realiza la operación indicada en la instrucción.
- **Almacenamiento de operando (os, operand store):** escribe el resultado en memoria o lo saca a través de un dispositivo de E/S.

Los estados en la parte superior de la Figura 3.6 ocasionan intercambios entre la CPU y la memoria o un módulo de E/S. Los estados en la parte inferior del diagrama sólo ocasionan operaciones internas a la CPU. El estado oac aparece dos veces, puesto que una instrucción puede ocasionar una lectura, una escritura, o ambas cosas. No obstante, la acción realizada en ese estado es la misma en ambos casos y, por eso, sólo se necesita un único identificador de estado.

Obsérvese además que en el diagrama se considera la posibilidad de múltiples operandos y múltiples resultados, puesto que se necesitan en algunas instrucciones de ciertas máquinas. Por ejemplo, la instrucción ADD A,B del PDP-11 da lugar a la siguiente secuencia de estados: iac, if, iod, oac, of, oac, of, do, oac, os.

Por último, en algunas máquinas, se puede especificar con una única instrucción una operación a realizar con un vector (matriz unidimensional) de números o con una cadena (matriz unidimensional) de caracteres. Como indica la Figura 3.6, esto implicaría una repetición de estados de captación y/o almacenamiento de operando.

INTERRUPCIONES

Prácticamente todos los computadores disponen de un mecanismo mediante el que otros módulos (E/S, memoria) pueden interrumpir el procesamiento normal de la CPU. La Tabla 3.1 enumera las clases de interrupciones más comunes. La naturaleza específica de estas interrupciones se examina en este libro más tarde, especialmente en los Capítulos 6 y 11. Sin embargo, necesitamos introducir el concepto ahora para comprender más claramente la esencia del ciclo de instrucción y los efectos de las interrupciones en la estructura de interconexión. En este momento, el lector no necesita conocer los detalles de la generación y el procesamiento de las interrupciones, sino sólamente concentrarse en la comunicación entre módulos que resultan de las interrupciones.

En primer lugar, las interrupciones proporcionan una forma de mejorar la eficiencia del procesador. Por ejemplo, la mayoría de los dispositivos externos son mucho más lentos que el procesador. Supóngase que el procesador está transfiriendo datos a una impresora utili-

Tabla 3.1. Clases de interrupciones

Programa	Generadas por alguna condición que se produce como resultado de la ejecución de una instrucción, tal como desbordamiento aritmético (<i>overflow</i>), división por cero, intento de ejecutar una instrucción máquina inexistente, e intento de acceder fuera del espacio de memoria permitido para el usuario.
Temporización	Generadas por un temporizador interno al procesador. Esto permite al sistema operativo realizar ciertas funciones de manera regular.
E/S	Generadas por un controlador de E/S, para indicar la finalización sin problemas de una operación o para avisar de ciertas condiciones de error.
Fallo de hardware	Generadas por un fallo tal como la falta de potencia de alimentación o un error de paridad en la memoria.

zando el esquema del ciclo de instrucción de la Figura 3.3. Después de cada operación de escritura, el procesador tendrá que parar y permanecer ocioso hasta que la impresora complete la escritura. La longitud de esta pausa puede ser del orden de muchos cientos, o incluso miles, de ciclos de instrucción que no implican acceso a memoria. Claramente, esto supone un derroche en el uso del procesador.

La Figura 3.7a ilustra la situación del ejemplo referido en el párrafo precedente. El programa de usuario realiza una serie de llamadas de escritura (WRITE) entremezcladas con el procesamiento. Los segmentos de código 1, 2, y 3 corresponden a secuencias de instrucciones que no ocasionan operaciones de E/S. Las llamadas de escritura (WRITE) corresponden a llamadas a un programa de E/S que es una de las utilidades del sistema operativo y que se encarga de la operación de E/S considerada. El programa de E/S está constituido por tres secciones:

- Una secuencia de instrucciones, rotulada con 4 en la figura, de preparación para la operación de E/S a realizar. Esto puede implicar la copia del dato que se va a proporcionar en un registro intermedio («buffer») especial, y preparar los parámetros de control del dispositivo de E/S.
- La orden de E/S propiamente dicha. Si no se utilizan interrupciones, una vez que se ejecuta esta orden, el programa debe esperar a que el dispositivo de E/S complete la operación solicitada. El programa esperaría simplemente comprobando repetidamente una condición que indique si se ha realizado la operación de E/S.
- Una secuencia de instrucciones, rotulada con 5 en la figura, que terminan la operación de E/S. Estas pueden incluir la activación de un indicador («flag») que señale si la operación se ha completado correctamente o con errores.

Debido a que la operación de E/S puede necesitar un tiempo relativamente largo, el programa de E/S debe detenerse a esperar que concluya dicha operación; por consiguiente, el programa de usuario estará parado en las llamadas de escritura (WRITE) durante un período de tiempo considerable.

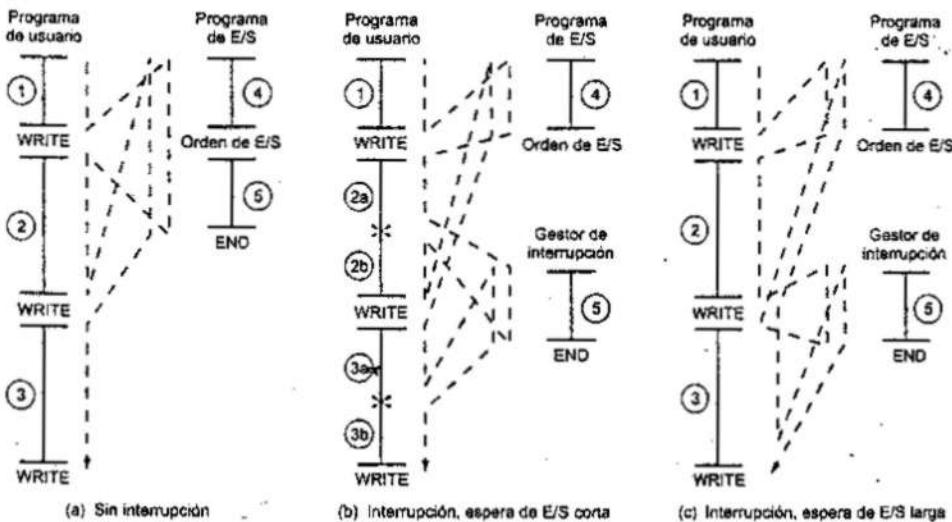


Figura 3.7. Flujo de control de un programa sin y con interrupciones.

Las interrupciones y el ciclo de instrucción

Con el uso de interrupciones, el procesador puede dedicarse a ejecutar otras instrucciones mientras una operación de E/S está en curso. Considérese el flujo de control de la Figura 3.7b. Como antes, el programa de usuario llega a un punto en el que realiza una llamada al sistema para realizar una escritura (WRITE). El programa de E/S al que se llama en este caso, está constituido sólo por el código de preparación y la orden de E/S propiamente dicha. Después de que estas pocas instrucciones se hayan ejecutado, el control se devuelve al programa de usuario. Mientras tanto, el dispositivo externo está ocupado aceptando el dato de la memoria del computador e imprimiéndolo. Esta operación de E/S se realiza concurrentemente con la ejecución de instrucciones del programa de usuario.

Cuando el dispositivo externo pasa a estar preparado para actuar, es decir, cuando está listo para aceptar más datos del procesador, el módulo de E/S de este dispositivo externo envía una señal de *peticIÓN de interrupción* al procesador. El procesador responde suspendiendo la operación del programa que estaba ejecutando y salta a un programa, conocido como «gestor de interrupción», que da servicio a ese dispositivo concreto y prosigue con la ejecución del programa original, después de haber dado dicho servicio al dispositivo. En la Figura 3.7b, los puntos en los que se producen las interrupciones se indican con un asterisco (*).

Desde el punto de vista del programa de usuario, una interrupción es precisamente eso: una interrupción en la secuencia normal de funcionamiento. Cuando el procesamiento de la interrupción se completa, la ejecución prosigue (Figura 3.8). Así, el programa de usuario no tiene que incluir ningún código especial para possibilitar las interrupciones; el procesador y el sistema operativo son los responsables de detener el programa de usuario y después permitir que prosiga en el mismo punto.

Para permitir el uso de interrupciones, se añade un *ciclo de interrupción* al ciclo de instrucción, como muestra la Figura 3.9. En el ciclo de interrupción, el procesador comprueba si se ha generado alguna interrupción, indicada por la presencia de una señal de interrupción. Si no hay señales de interrupción pendientes, el procesador continúa con el ciclo de captación y accede a la siguiente instrucción del programa en curso. Si hay alguna interrupción pendiente, el procesador hace lo siguiente:

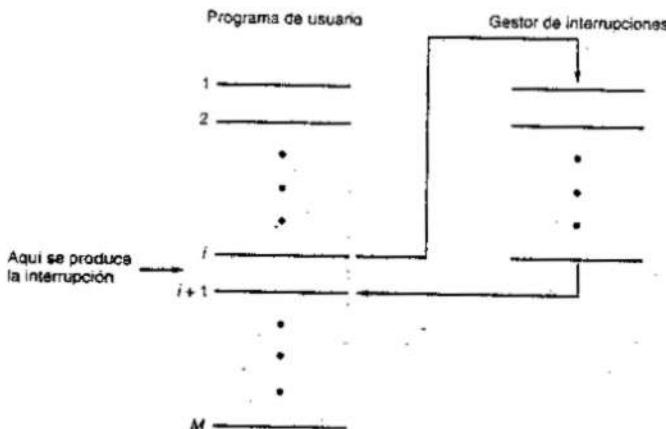


Figura 3.8. Transferencia de control debida a una interrupción.

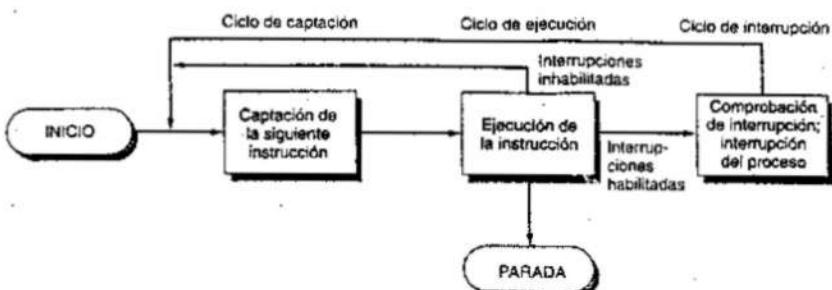


Figura 3.8. Ciclo de instrucción con interrupciones.

1. Suspende la ejecución del programa en curso y guarda su contexto. Esto significa almacenar la dirección de la siguiente instrucción a ejecutar (contenido actual del contador de programa) y cualquier otro dato relacionado con la actividad en curso del procesador.
2. Carga el contador de programa con la dirección de comienzo de una rutina de *gestión de interrupción*.

A continuación, el procesador prosigue con el ciclo de captación y accede a la primera instrucción del programa de gestión de interrupción, que dará servicio a la interrupción. Generalmente, este programa determina el origen de la interrupción y realiza todas las acciones que sean necesarias. Por ejemplo, en el caso que hemos estado analizando, el gestor determina qué módulo de E/S generó la interrupción, y puede saltar a un programa que escribe más datos en ese módulo de E/S. Cuando la rutina de gestión de interrupción se completa, el procesador puede proseguir la ejecución del programa de usuario en el punto en el que se interrumpió.

Es claro que este proceso supone una cierta penalización («overhead»). Deben ejecutarse instrucciones extra (en el gestor de interrupción) para determinar el origen de la interrupción y para decidir la acción apropiada. No obstante, debido a la cantidad relativamente grande de tiempo que se perdería simplemente por la espera asociada a la operación de E/S, el procesador puede emplearse de manera mucho más eficiente utilizando interrupciones.

Para apreciar el aumento de eficiencia, considérese la Figura 3.10, que es un diagrama de tiempos basado en el flujo de control de las Figuras 3.7a y 3.7b. Las Figuras 3.7b y 3.10 asumen que el tiempo necesario para la operación de E/S es relativamente corto: menor que el tiempo para completar la ejecución de las instrucciones del programa de usuario que hay entre operaciones de escritura. La situación más frecuente, especialmente para un dispositivo lento como una impresora, es que la operación de E/S requiera mucho más tiempo que ejecutar una secuencia de instrucciones de usuario. La Figura 3.7c ilustra esta situación. En este caso, el programa de usuario llega a la segunda llamada de escritura (WRITE) antes de que la operación de E/S generada por la primera llamada se complete. El resultado es que el programa de usuario se para en este punto. Cuando la operación de E/S precedente se completa, esta nueva llamada de escritura se puede procesar, y se puede iniciar una nueva operación de E/S. La Figura 3.11 muestra la temporización para esta situación con y sin interrupciones. Podemos ver que existe una mejora de eficiencia porque parte del tiempo durante el cual la operación de E/S está en marcha se solapa con la ejecución de instrucciones de usuario.

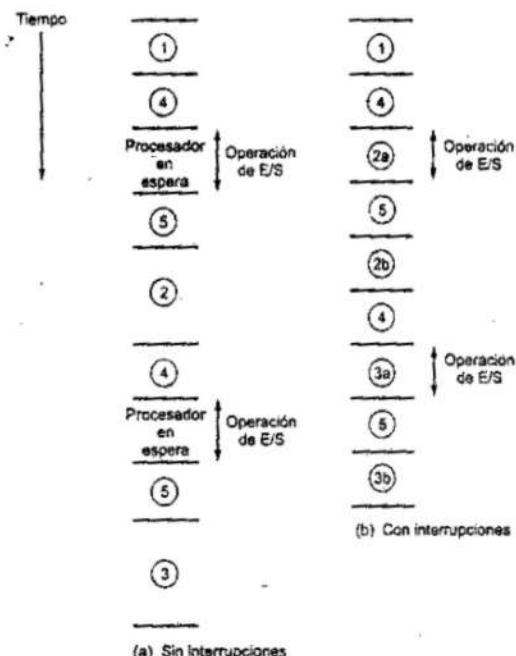


Figura 3.10. Temporización en un programa: espera de E/S corta.

La Figura 3.12 muestra el diagrama de estados del ciclo de instrucción modificado para incluir al procesamiento del ciclo de interrupción.

Interrupciones múltiples

Hasta ahora únicamente se ha discutido la existencia de una sola interrupción. Supóngase, no obstante, que se puedan producir varias interrupciones. Por ejemplo, un programa puede estar recibiendo datos a través de una línea de comunicación e imprimiendo resultados. La impresora generará interrupciones cada vez que complete una operación de escritura. El controlador de la línea de comunicación generará una interrupción cada vez que llegue una unidad de datos. La unidad de datos puede ser un carácter o un bloque, según el protocolo de comunicación. En cualquier caso, es posible que se produzca una interrupción de comunicaciones mientras se está procesando la interrupción de la impresora.

Se pueden seguir dos alternativas para tratar las interrupciones múltiples. La primera es desactivar las interrupciones mientras se está procesando una interrupción. Una *interrupción inhabilitada* (*disabled interrupt*) simplemente significa que el procesador puede y debe ignorar la señal de petición de interrupción. Si se produce una interrupción en ese momento, generalmente se mantiene pendiente, y será examinada por el procesador una vez éste haya activado las interrupciones. Así, cuando un programa de usuario se está ejecutando y se pro-

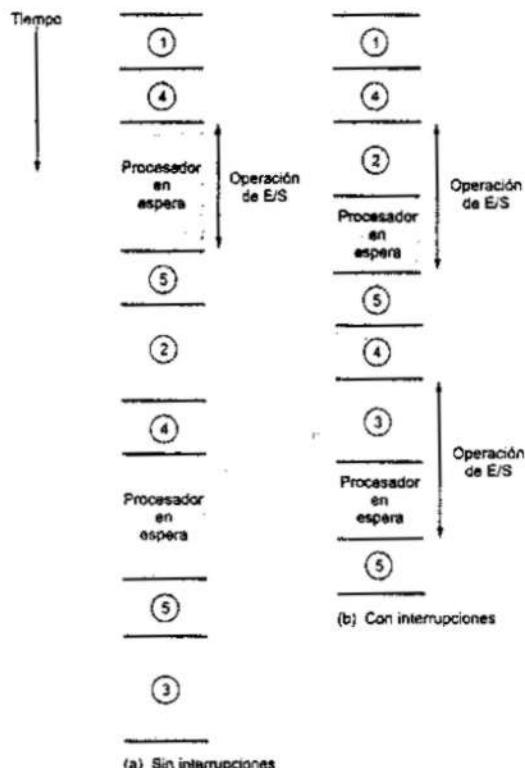


Figura 3.11. Temporización en un programa: Espera de E/S larga..

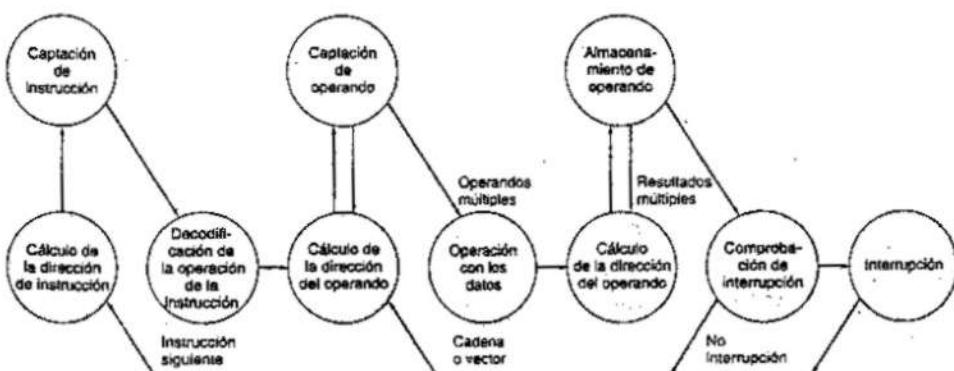


Figura 3.12. Diagrama de estados de un ciclo de instrucción, con interrupciones.

duce una interrupción, las interrupciones se inhabilitan inmediatamente. Después de que la rutina de gestión de interrupción termine, las interrupciones se habilitan antes de que el programa de usuario prosiga, y el procesador comprueba si se han producido interrupciones adicionales. Esta aproximación es correcta y simple, puesto que las interrupciones se manejan en un orden secuencial estricto (Figura 3.13a).

El inconveniente del enfoque anterior es que no tiene en cuenta la prioridad relativa, ni las solicitudes con un tiempo crítico. Por ejemplo, cuando llega una entrada desde la línea de comunicaciones, ésta debe tramitarse rápidamente, para dejar espacio a los datos siguientes. Si los primeros datos no se han procesado antes de que lleguen los siguientes, se pueden perder.

Una segunda alternativa consiste en definir prioridades para las interrupciones, y permitir que una interrupción de prioridad más alta pueda interrumpir a un gestor de interrupción de prioridad menor (Figura 3.13b). Como ejemplo de esta segunda alternativa, considérese un sistema con tres dispositivos de E/S: una impresora, un disco y una línea de comunicaciones, con prioridades crecientes de 2, 4 y 5 respectivamente. La Figura 3.14 muestra una posible secuencia. Un programa de usuario comienza en $t = 0$. En $t = 10$, la impresora produce una interrupción; la información del programa de usuario se sitúa en la pila del sistema, y la ejec-

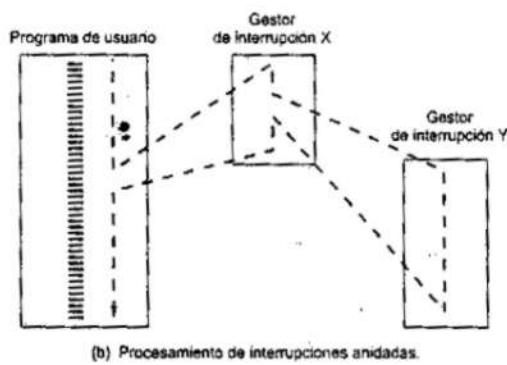
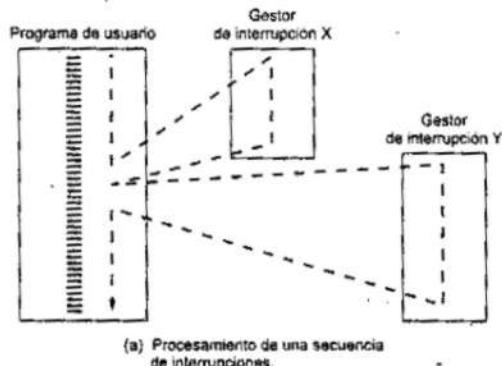


Figura 3.13. Transferencia de control con interrupciones múltiples.

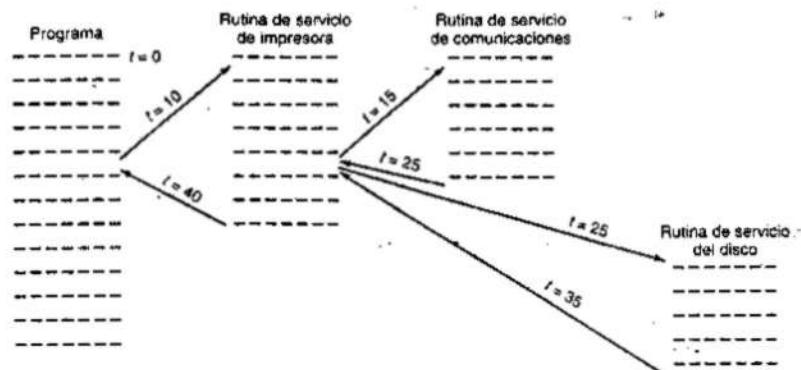


Figura 3.14. Ejemplo de secuencia temporal de varias interrupciones [TANE90].

cución continúa con la rutina de servicio de interrupción de la impresora. Mientras se está ejecutando esta rutina, en $t = 15$, se produce una interrupción de comunicaciones. Como la línea de comunicaciones tiene una prioridad mayor que la impresora, se acepta la interrupción. La rutina de la impresora se interrumpe, su estado se introduce en la pila, y la ejecución continúa con la rutina de comunicaciones. Mientras se está ejecutando esta rutina, el disco ocasiona una interrupción ($t = 20$). Puesto que esta interrupción tiene una prioridad menor, simplemente se retiene, y la rutina de comunicaciones se ejecuta hasta que termina.

Cuando la rutina de comunicaciones termina ($t = 25$), se restaura el estado previo del procesador, que corresponde a la ejecución de la rutina de la impresora. No obstante, antes incluso de que pueda ejecutarse una sola instrucción de esa rutina, el procesador acepta la interrupción, de mayor prioridad, generada por el disco, y el control se transfiere a la rutina de disco. Sólo cuando esta rutina termina ($t = 35$) la rutina de la impresora puede reanudarse. Cuando termina esa rutina ($t = 40$), el control vuelve finalmente al programa de usuario.

FUNCIONAMIENTO DE LAS E/S

Hasta aquí, hemos discutido el funcionamiento del computador controlado por el procesador, y nos hemos fijado esencialmente en la interacción del procesador y la memoria. La discusión sólo ha aludido al papel de los componentes de E/S. Este papel se discute con detalle en el Capítulo 6, pero es conveniente hacer aquí un breve resumen. -

Un módulo de E/S (por ejemplo, un controlador de disco) puede intercambiar datos directamente con el procesador. Igual que el procesador puede iniciar una lectura o escritura en memoria, especificando la dirección de una posición concreta de la misma, el procesador también puede leer o escribir datos de, o en, un módulo de E/S. En este caso, el procesador identifica un dispositivo específico controlado por un módulo de E/S determinado. Por consiguiente, se puede producir una secuencia de instrucciones similar a la de la Figura 3.5, con instrucciones de E/S en lugar de las instrucciones de referencia a memoria.

En algunos casos, es deseable permitir que los intercambios de E/S se produzcan directamente con la memoria. En ese caso, el procesador cede a un módulo de E/S la autoridad para leer de, o escribir en, memoria, para que así la transferencia E/S-memoria pueda producirse sin la intervención del procesador. Durante esas transferencias, el módulo de E/S proporcio-

na a la memoria las órdenes de lectura o escritura, liberando al procesador de cualquier responsabilidad en el intercambio. Esta operación se conoce con el nombre de acceso directo a memoria (DMA, Direct Memory Access), y se estudiará con detalle en el Capítulo 6. Por ahora, todo lo que se necesita conocer es que la estructura de interconexión del computador puede tener que permitir la interacción directa E/S-memoria.

3.3. ESTRUCTURAS DE INTERCONEXIÓN

Un computador está constituido por un conjunto de unidades o módulos de tres tipos elementales (procesador, memoria y E/S) que se comunican entre sí. En efecto, un computador es una red de módulos elementales. Por consiguiente, deben existir líneas para interconectar estos módulos.

El conjunto de líneas que conectan los diversos módulos se denomina *estructura de interconexión*. El diseño de dicha estructura dependerá de los intercambios que deban producirse entre los módulos.

La Figura 3.15 sugiere los tipos de intercambios que se necesitan, indicando las formas de las entradas y las salidas en cada tipo de módulo:

- **Memoria:** generalmente, un módulo de memoria está constituido por N palabras de la misma longitud. A cada palabra se le asigna una única dirección numérica ($0, 1, \dots, N - 1$). Una palabra de datos puede leerse o escribirse en la memoria. El tipo de operación se indica mediante las señales de control Read (leer) y Write (escribir). La posición de memoria para la operación se especifica mediante una dirección.
- **Módulo de E/S:** desde un punto de vista interno (al computador), la E/S es funcionalmente similar a la memoria. Hay dos tipos de operaciones: leer y escribir. Además, un módulo de E/S puede controlar más de un dispositivo externo. Nos referiremos a cada una de estas interfaces con un dispositivo externo con el nombre de *puerto* («porto»), y se le asignará una dirección a cada uno ($0, 1, \dots, M - 1$). Por otra parte, existen líneas externas de datos para la entrada y la salida de datos por un dispositivo externo. Por último, un módulo de E/S puede enviar señales de interrupción al procesador.
- **Procesador:** el procesador lee instrucciones y datos, escribe datos una vez los ha procesado, y utiliza ciertas señales para controlar el funcionamiento del sistema. También puede recibir señales de interrupción.

La lista precedente especifica los datos que se intercambian. La estructura de interconexión debe dar cobertura a los siguientes tipos de transferencias:

- **Memoria a procesador:** el procesador lee una instrucción o un dato desde la memoria.
- **Procesador a memoria:** el procesador escribe un dato en la memoria.
- **E/S a procesador:** el procesador lee datos de un dispositivo de E/S a través de un módulo de E/S.
- **Procesador a E/S:** el procesador envía datos al dispositivo de E/S.
- **Memoria a E/S y viceversa:** en estos dos casos, un módulo de E/S puede intercambiar datos directamente con la memoria, sin que tengan que pasar a través del procesador, utilizando el acceso directo a memoria (DMA).

A través de los años, se han probado diversas estructuras de interconexión. Las más comunes son, con diferencia, las estructuras de bus y de buses múltiples. El resto de este capítulo se dedica a evaluar las estructuras de buses.

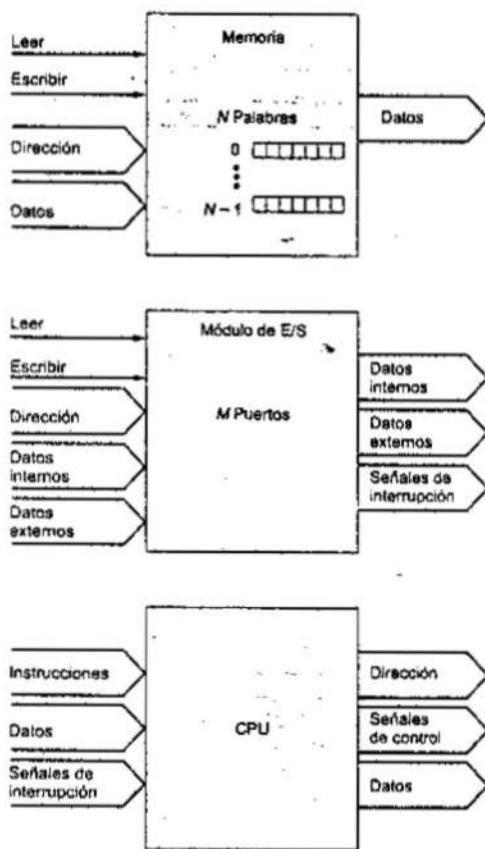


Figura 3.15. Módulos de un computador.

3.4. INTERCONEXIÓN CON BUSES

Un bus es un camino de comunicación entre dos o más dispositivos. Una característica clave de un bus es que se trata de un medio de transmisión compartido. Al bus se conectan varios dispositivos, y cualquier señal transmitida por uno de esos dispositivos está disponible para que los otros dispositivos conectados al bus puedan acceder a ella. Si dos dispositivos transmiten durante el mismo periodo de tiempo, sus señales pueden solaparse y distorsionarse. Consiguientemente, sólo un dispositivo puede transmitir con éxito en un momento dado.

Usualmente, un bus está constituido por varios caminos de comunicación, o líneas. Cada línea es capaz de transmitir señales binarias representadas por 1 y por 0. En un intervalo de tiempo, se puede transmitir una secuencia de dígitos binarios a través de una única línea. Se pueden utilizar varias líneas del bus para transmitir dígitos binarios simultáneamente (en paralelo). Por ejemplo, un dato de 8 bits puede transmitirse mediante ocho líneas del bus.

Los computadores poseen distintos tipos de buses que proporcionan comunicación entre sus componentes a distintos niveles dentro de la jerarquía del sistema. El bus que conecta los componentes principales del computador (procesador, memoria y E/S) se denomina *bus del sistema* («system bus»). Las estructuras de interconexión más comunes dentro de un computador están basadas en el uso de uno o más buses del sistema.

ESTRUCTURA DEL BUS.

El bus del sistema está constituido, usualmente, por entre 50 y 100 líneas. A cada línea se le asigna un significado o una función particular. Aunque existen diseños de buses muy diversos, en todos ellos las líneas se pueden clasificar en tres grupos funcionales (Figura 3.16): líneas de datos, de direcciones, y de control. Además, pueden existir líneas de alimentación para suministrar energía a los módulos conectados al bus.

Las *líneas de datos* proporcionan un camino para transmitir datos entre los módulos del sistema. El conjunto constituido por estas líneas se denomina *bus de datos*. El bus de datos generalmente consta de 8, 16 o 32 líneas distintas, cuyo número se conoce como *anchura del bus de datos*. Puesto que cada línea sólo puede transportar un bit cada vez, el número de líneas determina cuántos bits se pueden transferir al mismo tiempo. La anchura del bus es un factor clave a la hora de determinar las prestaciones del conjunto del sistema. Por ejemplo, si el bus de datos tiene una anchura de 8 bits, y las instrucciones son de 16 bits, entonces el procesador debe acceder al módulo de memoria dos veces por cada ciclo de instrucción.

Las *líneas de dirección* se utilizan para designar la fuente o el destino del dato situado en el bus de datos. Por ejemplo, si el procesador desea leer una palabra (8, 16 o 32 bits) de datos de la memoria, sitúa la dirección de la palabra deseada en las líneas de direcciones. Claramente, la anchura del bus de direcciones determina la máxima capacidad de memoria posible en el sistema. Además, las líneas de direcciones generalmente se utilizan también para direccionar los puertos de E/S. Usualmente, los bits de orden más alto se utilizan para seleccionar una posición de memoria o un puerto de E/S dentro de un módulo. Por ejemplo, en un bus de 8 bits, la dirección 0111111 e inferiores harían referencia a posiciones dentro de un módulo de memoria (el módulo 0) con 128 palabras de memoria, y las direcciones 10000000 y superiores designarían dispositivos conectados a un módulo de E/S (módulo 1).

Las *líneas de control* se utilizan para controlar el acceso y el uso de las líneas de datos y de direcciones. Puesto que las líneas de datos y de direcciones son compartidas por todos los componentes, debe existir una forma de controlar su uso. Las señales de control transmiten tanto órdenes como información de temporización entre los módulos del sistema. Las señales de temporización indican la validez de los datos y las direcciones. Las señales de órdenes especifican las operaciones a realizar. Algunas líneas de control típicas son:

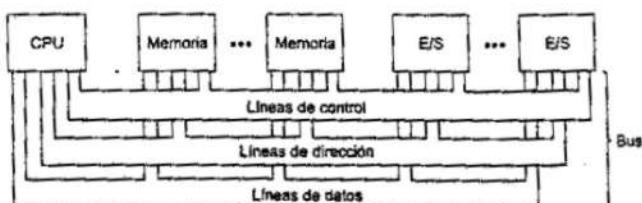


Figura 3.16. Esquema de interconexión mediante un bus.

- **Escritura en memoria (Memory Write):** hace que el dato del bus se escriba en la posición direccionada.
- **Lectura de memoria (Memory Read):** hace que el dato de la posición direccionada se sitúe en el bus.
- **Escritura de E/S (I/O Write):** hace que el dato del bus se transfiera a través del puerto de E/S direccionado.
- **Lectura de E/S (E/S Read):** hace que el dato del puerto de E/S direccionado se sitúe en el bus.
- **Transferencia reconocida (Transfer ACK):** indica que el dato se ha aceptado o se ha situado en el bus.
- **Petición de bus (Bus Request):** indica que un módulo necesita disponer del control del bus.
- **Cesión de bus (Bus Grant):** indica que se cede el control del bus a un módulo que lo había solicitado.
- **Petición de interrupción (Interrupt Request):** indica si hay una interrupción pendiente.
- **Interrupción reconocida (Interrupt ACK):** señala que la interrupción pendiente se ha aceptado.
- **Reloj (Clock):** se utiliza para sincronizar las operaciones.
- **Inicio (Reset):** pone los módulos conectados en su estado inicial.

El funcionamiento del bus se describe a continuación. Si un módulo desea enviar un dato a otro debe hacer dos cosas: (1) obtener el uso del bus, y (2) transferir el dato a través del bus. Si un módulo desea pedir un dato a otro módulo, debe (1) obtener el uso del bus, y (2) transferir la petición al otro módulo mediante las líneas de control y dirección apropiadas. Después debe esperar a que el segundo módulo envíe el dato.

Físicamente, el bus de sistema es de hecho un conjunto de conductores eléctricos paralelos. Estos conductores son líneas de metal grabadas en una tarjeta (tarjeta de circuito impreso). El bus se extiende a través de todos los componentes del sistema, cada uno de los cuales se conecta a algunas o a todas las líneas del bus. Una disposición física muy común se muestra en la Figura 3.17. En este ejemplo, el bus consta de dos columnas verticales de conductores. A lo largo de esas columnas, a intervalos regulares, hay puntos de conexión en forma de

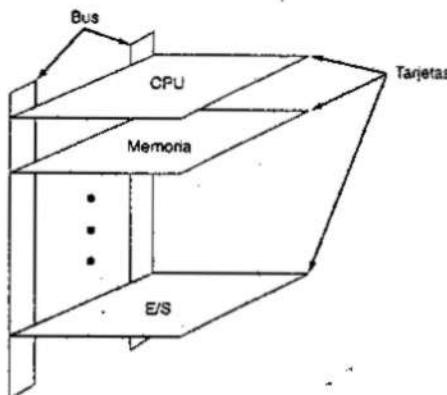


Figura 3.17. Implementación física típica de una arquitectura de bus.

ranuras («slots») dispuestas en sentido horizontal para sostener las tarjetas de circuito impreso. Cada uno de los componentes principales del sistema ocupa una o varias tarjetas y se conecta al bus a través de esas ranuras. El sistema completo se introduce dentro de un chasis.

Ésta es la disposición más conveniente. Así se puede adquirir un computador pequeño y expandirlo (ampliar memoria, módulos de E/S) más adelante añadiendo más tarjetas. Si un componente de una tarjeta falla, la tarjeta puede quitarse y sustituirse fácilmente.

JERARQUÍAS DE BUSES

Si se conecta un gran número de dispositivos al bus, las prestaciones pueden disminuir. Hay dos causas principales:

1. En general, a más dispositivos conectados al bus, mayor es el retardo de propagación. Este retardo determina el tiempo que necesitan los dispositivos para coordinarse en el uso del bus. Si el control del bus pasa frecuentemente de un dispositivo a otro, los retardos de propagación pueden afectar sensiblemente a las prestaciones.
2. El bus puede convertirse en un cuello de botella a medida que las peticiones de transferencia acumuladas se aproximan a la capacidad del bus. Este problema se puede resolver en alguna medida incrementando la velocidad a la que el bus puede transferir los datos, y utilizando buses más anchos (por ejemplo incrementando el bus de datos de 32 a 64 bits). Sin embargo, puesto que la velocidad de transferencia que necesitan los dispositivos conectados al bus (por ejemplo, controladores de gráficos y de vídeo, o interfaces de red) está incrementándose rápidamente, es un hecho que el bus único está destinado a dejar de utilizarse.

Por consiguiente, la mayoría de los computadores utilizan varios buses, normalmente organizados jerárquicamente. Una estructura típica se muestra en la Figura 3.18a. Hay un bus local que conecta el procesador a una memoria caché, y al que pueden conectarse también uno o más dispositivos locales. El controlador de memoria cache conecta la cache no sólo al bus local, sino también al bus de sistema, donde se conectan todos los módulos de memoria principal. Como se discute en el Capítulo 4, el uso de una cache alivia la exigencia de soportar los accesos frecuentes del procesador a memoria principal. De hecho, la memoria principal puede pasar del bus local al bus de sistema. De esta forma, las transferencias de E/S con la memoria principal a través del bus de sistema no interfieren la actividad del procesador.

Es posible conectar controladores de E/S directamente al bus de sistema. Una solución más eficiente consiste en utilizar uno o más buses de expansión. La interfaz del bus de expansión regula las transferencias de datos entre el bus de sistema y los controladores conectados al bus de expansión. Esta disposición permite conectar al sistema una amplia variedad de dispositivos de E/S y, al mismo tiempo, aislar el tráfico de información entre la memoria y el procesador del tráfico correspondiente a las E/S.

La Figura 3.18a muestra algunos ejemplos típicos de dispositivos de E/S que pueden estar conectados al bus de expansión. Las conexiones a red incluyen conexiones a redes de área local (LAN, Local Area Networks), tales como una red Ethernet de 10 Mbps y conexiones a redes de área amplia (Wide Area Networks), tales como la red de commutación de paquetes («packet-switching network»). La interfaz SCSI (Small Computer System Interface) es en sí un tipo de bus utilizado para conectar controladores de disco y otros periféricos. El puerto serie puede utilizarse para conectar una impresora o un escáner.

Esta arquitectura de buses tradicional es razonablemente eficiente, pero muestra su debilidad a medida que los dispositivos de E/S ofrecen prestaciones cada vez mayores. La respuesta común a esta situación, por parte de la industria, ha sido proponer un bus de alta

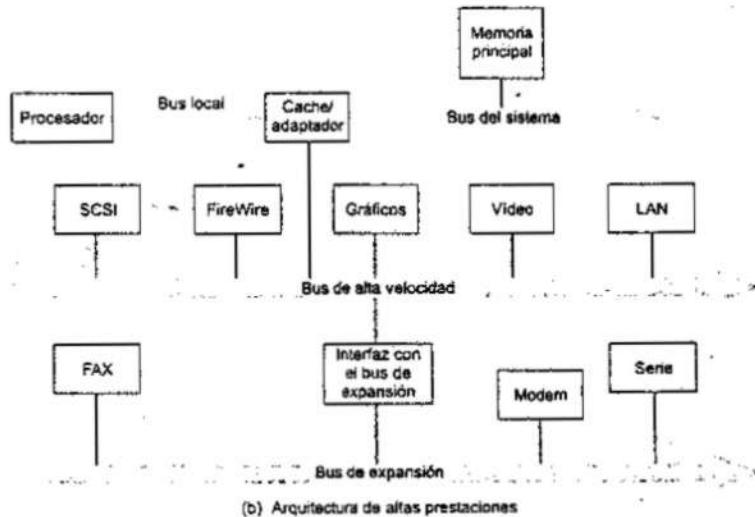
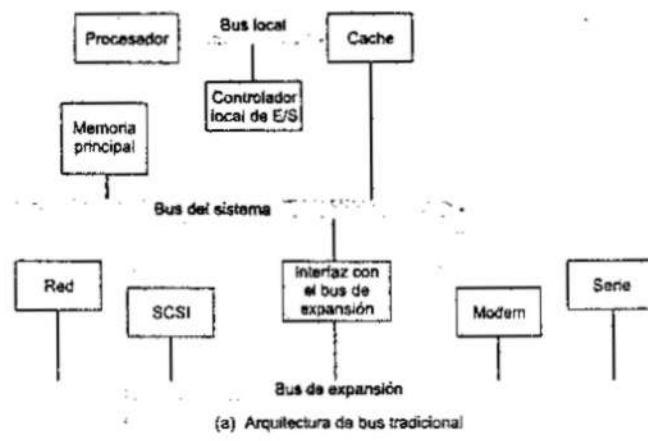


Figura 3.18. Ejemplos de configuraciones de bus.

velocidad, que está estrechamente integrado con el resto del sistema, y requiere sólo un adaptador («bridge») entre el bus del procesador y el bus de alta velocidad. En algunas ocasiones, esta disposición es conocida como arquitectura de entreplanta («mezzanine architecture»).

La Figura 3.18b muestra un ejemplo típico de esta aproximación. De nuevo, hay un bus local que conecta el procesador a un controlador de cache, que a su vez está conectado al bus de sistema que soporta a la memoria principal. El controlador de cache está integrado junto con el adaptador, o dispositivo de acople, que permite la conexión al bus de alta velocidad. Este bus permite la conexión de LAN de alta velocidad, tales como Fast Ethernet a

100 Mbps, controladores de estaciones de trabajo específicos para aplicaciones gráficas y de video, y también controladores de interfaz para buses de periféricos, tales como SCSI y Fire-wire. Este último es un bus de alta velocidad diseñado específicamente para conectar dispositivos de E/S de alta capacidad. Los dispositivos de velocidad menor pueden conectarse al bus de expansión, que utiliza una interfaz para adaptar el tráfico entre el bus de expansión y el bus de alta velocidad.

La ventaja de esta organización es que el bus de alta velocidad acerca al procesador los dispositivos que exigen prestaciones elevadas y, al mismo tiempo, es independiente del procesador. Así, se pueden tolerar las diferencias de velocidad entre el procesador y el bus de altas prestaciones y las variaciones en la definición de las líneas de los buses. Los cambios en la arquitectura del procesador no afectan al bus de alta velocidad, y viceversa.

ELEMENTOS DE DISEÑO DE UN BUS

Aunque existe una gran diversidad de diseños de buses, hay unos pocos parámetros o elementos de diseño que sirven para distinguir y clasificar los buses. La Tabla 3.2 enumera los elementos clave.

Tipos de buses

Las líneas del bus se pueden dividir en dos tipos genéricos: dedicadas y multiplexadas. Una línea de bus dedicada está permanentemente asignada a una función o a un subconjunto físico de componentes del computador.

Un ejemplo de dedicación funcional, común en muchos buses, es el uso de líneas separadas para direcciones y para datos. Sin embargo, no es esencial. Por ejemplo, la información de dirección y datos podría transmitirse a través del mismo conjunto de líneas si se utiliza una línea de control de dirección válida. Al comienzo de la transferencia de datos, la dirección se sitúa en el bus y se activa la línea de dirección válida. En ese momento, cada módulo dispone de un período de tiempo para copiar la dirección y determinar si es él el módulo direccionado. Después la dirección se quita del bus, y las mismas conexiones se utilizan para la subsecuente transferencia de lectura o escritura de datos. Este método de uso de las mismas líneas para usos diferentes se llama *multiplexado en el tiempo*.

La ventaja del multiplexado en el tiempo es el uso de menos líneas, cosa que ahorra espacio y, normalmente, costes. La desventaja es que se necesita una circuitería más compleja en cada módulo. Además, existe una posible reducción en las prestaciones debido a que los eventos que deben compartir las mismas líneas no pueden producirse en paralelo.

La *dedicación física* se refiere al uso de múltiples buses, cada uno de los cuales conecta sólo un subconjunto de módulos. Un ejemplo típico es el uso de un bus de E/S para interco-

Tabla 3.2. Elementos del diseño de un bus

Tipo	Anchura del bus
Dedicado	Dirección
Multiplexado	Datos
Método de arbitraje	Tipo de transferencia de datos
Centralizado	Lectura
Distribuido	Escritura
Temporización	Lectura-modificación-escritura
Síncrono	Lectura-después-de-escritura
Asíncrono	Bloque

nectar todos los módulos de E/S: este bus, a su vez, se conecta al bus principal a través de algún tipo de módulo adaptador de E/S. La ventaja potencial de la dedicación física es su elevado rendimiento, debido a que hay menos disputas por el acceso al bus («bus contention»). Una desventaja es el incremento en el tamaño y el costo del sistema.

Método de arbitraje

En todos los sistemas, exceptuando los más simples, puede necesitar el control del bus más de un módulo. Por ejemplo, un módulo de E/S puede necesitar leer o escribir directamente en memoria, sin enviar el dato al procesador. Puesto que, en un instante dado, sólo una unidad puede transmitir a través del bus, se requiere algún método de arbitraje. Los diversos métodos se pueden clasificar aproximadamente como centralizados o distribuidos. En un esquema centralizado, un único dispositivo hardware, denominado *controlador del bus* o *árbitro*, es responsable de asignar tiempos en el bus. El dispositivo puede estar en un módulo separado o ser parte del procesador. En un esquema distribuido, no existe un controlador central. En su lugar, cada módulo dispone de lógica para controlar el acceso, y los módulos actúan conjuntamente para compartir el bus. En ambos métodos de arbitraje, el propósito es designar un dispositivo, el procesador o un módulo de E/S, como maestro del bus. El maestro podría entonces iniciar una transferencia de datos (lectura o escritura) con otro dispositivo, que actúa como esclavo en este intercambio concreto. Veremos ejemplos de ambos métodos de arbitraje más adelante, en esta sección.

Temporización

El término temporización hace referencia a la forma en la que se coordinan los eventos en el bus. Con temporización sincrona, la presencia de un evento en el bus está determinada por un reloj. El bus incluye una línea de reloj a través de la que se transmite una secuencia en la que se alternan intervalos regulares de igual duración a uno y a cero. Un único intervalo a uno seguido de otro a cero se conoce como *ciclo de reloj* o *ciclo de bus* y define un intervalo de tiempo unidad («time slot»). Todos los dispositivos del bus pueden leer la línea de reloj, y todos los eventos empiezan al principio del ciclo de reloj. La Figura 3.19a muestra el diagrama de tiempos de una operación de lectura sincrona (en el Apéndice 3A se puede consultar una descripción de los diagramas de tiempo). Otras señales del bus pueden cambiar en el flanco de subida de la señal de reloj (reaccionan con un ligero retardo). La mayoría de los eventos se prolongan durante un único ciclo de reloj. En este ejemplo sencillo, la CPU activa una señal de lectura y sitúa una dirección de memoria en las líneas de dirección. Además, activa una señal de inicio para indicar la presencia en el bus de la dirección y de la información de control. El módulo de memoria reconoce la dirección y, después de un retardo de un ciclo, sitúa el dato y la señal de reconocimiento en el bus.

Con la temporización asíncrona, la presencia de un evento en el bus es consecuencia y depende de que se produzca un evento previo. En el ejemplo sencillo de la Figura 3.19b, el procesador sitúa las señales de dirección y lectura en el bus. Después de un breve intervalo para que las señales se estabilicen, activa la señal MSYN (master sync, «sincronización del maestro»), indicando la presencia de señales de dirección y control válidas. El módulo de memoria responde proporcionando el dato y una señal SSYN (slave sync, «sincronización del esclavo»).

La temporización sincrona es más fácil de implementar y comprobar. Sin embargo, es menos flexible que la temporización asíncrona. Debido a que todos los dispositivos en un bus sincrónico deben utilizar la misma frecuencia de reloj, el sistema no puede aprovechar las

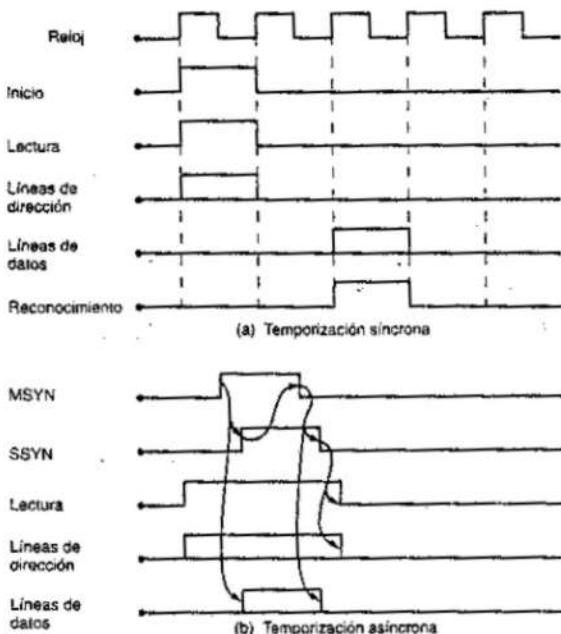


Figura 3.19. Temporización de una operación de lectura.

mejoras en las prestaciones de los dispositivos. Con la temporización asíncrona, pueden compartir el bus, una mezcla de dispositivos lentos y rápidos, utilizando tanto las tecnologías más antiguas, como las más recientes.

Anchura del bus

El concepto de anchura del bus se ha presentado ya. La anchura del bus de datos afecta a las prestaciones del sistema: cuanto más ancho es el bus de datos, mayor es el número de bits que se transmiten a la vez. La anchura del bus de direcciones afecta a la capacidad del sistema: cuanto más ancho es el bus de direcciones, mayor es el rango de posiciones a las que se puede hacer referencia.

Tipos de transferencia de datos

Por último, un bus permite varios tipos de transferencias de datos, tal y como ilustra la Figura 3.20. Todos los buses permiten tanto transferencias de escritura (dato de maestro a esclavo), como de lectura (dato de esclavo a maestro). En el caso de un bus con direcciones y datos multiplexados, el bus se utiliza primero para especificar la dirección y, luego, para transferir el dato. En una operación de lectura, generalmente, hay un tiempo de espera mientras el dato se está captando del dispositivo esclavo para situarlo en el bus. Tanto para la lectura como para la escritura, puede haber también un retardo si se necesita utilizar algún

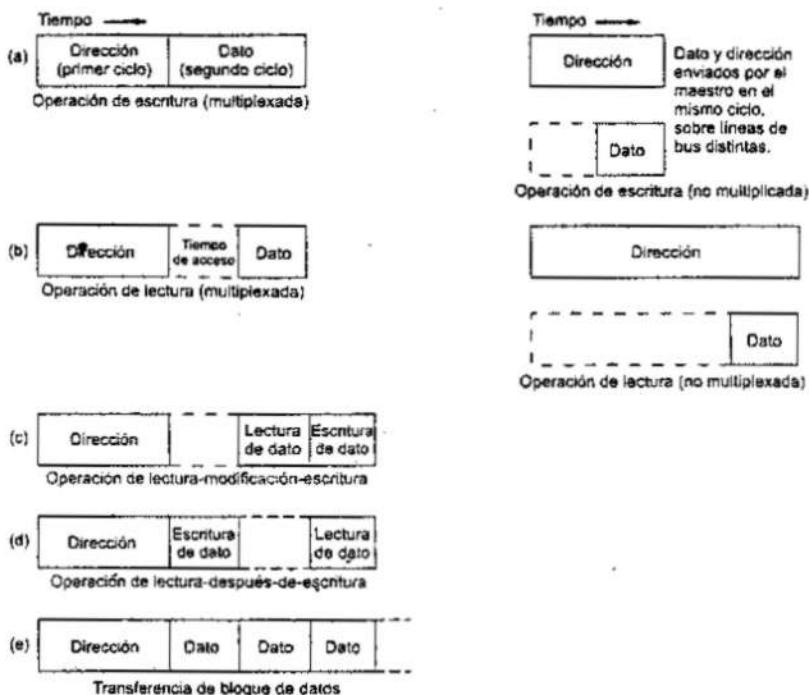


Figura 3.20. Tipos de transferencias de datos en un bus [GOOR89].

procedimiento de arbitraje para acceder al control del bus en el resto de la operación (es decir, tomar el bus para solicitar una lectura o una escritura y, después, tomar el bus de nuevo para realizar la lectura o la escritura).

En el caso de que haya líneas dedicadas a datos y a direcciones, la dirección se sitúa en el bus de direcciones y se mantiene ahí, mientras que el dato se ubica en el bus de datos. En una escritura, el maestro pone el dato en el bus de datos tan pronto como se han estabilizado las líneas de dirección y el esclavo ha podido reconocer su dirección. En una operación de lectura, el esclavo pone el dato en el bus de datos tan pronto como haya reconocido su dirección y disponga del mismo.

En ciertos buses también son posibles algunas operaciones combinadas. Una operación de lectura-modificación-escritura es simplemente una lectura seguida inmediatamente de una escritura en la misma dirección. La dirección se proporciona una sola vez al comienzo de la operación. La operación completa es generalmente indivisible, de cara a evitar cualquier acceso al dato por otros posibles maestros del bus. El objetivo primordial de esta posibilidad es proteger los recursos de memoria compartida en un sistema con multiprogramación (véase el Capítulo 7).

La lectura-después-de-escritura es una operación indivisible, que consiste en una escritura seguida inmediatamente de una lectura en la misma dirección. La operación de lectura se puede realizar con el propósito de comprobar el resultado.

Algunos buses también permiten transferencias de bloques de datos. En este caso, un ciclo de dirección viene seguido por n ciclos de datos. El primer dato se transfiere a, o desde, la dirección especificada; el resto de datos se transfieren a, o desde, las direcciones siguientes.

3.25. PCI

El bus PCI (Peripheral Component Interconnect, «interconexión de componente periférico») es un bus muy popular, de ancho de banda elevado, independiente del procesador, que se puede utilizar como bus de periféricos o bus para una arquitectura de entreplanta. Comparado con otras especificaciones comunes de bus, el PCI proporciona mejores prestaciones para los subsistemas de E/S de alta velocidad (por ejemplo, los adaptadores de pantalla gráfica, los controladores de interfaz de red, los controladores de disco, etc.). El estándar actual permite el uso de hasta 64 líneas de datos a 66 MHz, para una velocidad de transferencia de 528 MBytes/s. o 4.224 Gbps. Pero no es precisamente su elevada velocidad la que hace atractivo al PCI. El PCI ha sido diseñado específicamente para ajustarse económicamente a los requisitos de E/S de los sistemas actuales; se implementa con muy pocos circuitos integrados, y permite que otros buses se conecten al bus PCI.

Intel empezó a trabajar en el PCI en 1990, pensando en sus sistemas basados en el Pentium. Muy pronto, Intel cedió sus patentes al dominio público y promovió la creación de una asociación industrial, la PCI SIG (Special Interest Group), para continuar el desarrollo y mantener la compatibilidad de las especificaciones del PCI. El resultado ha sido que el PCI ha sido ampliamente adoptado, y se está incrementando su uso en los computadores personales, estaciones de trabajo, y servidores de sistema. La versión actual, PCI 2.1, apareció en 1995. Puesto que las especificaciones son de dominio público y están soportadas por una amplia banda de la industria de procesadores y periféricos, los productos PCI fabricados por compañías diferentes son compatibles.

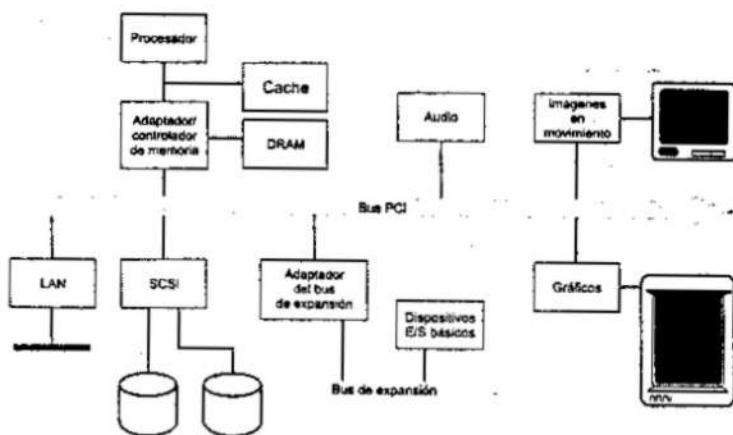
El PCI está diseñado para permitir una cierta variedad de configuraciones basadas en microprocesadores, incluyendo sistemas tanto de uno como de varios procesadores. Por consiguiente, proporciona un conjunto de funciones de uso general. Utiliza temporización sincronizada y un esquema de arbitraje centralizado.

La Figura 3.21a muestra la forma usual de utilizar el bus PCI en un sistema uniprocesador. Un dispositivo que integra el controlador de DRAM y el adaptador al bus PCI proporciona el acoplamiento al procesador y la posibilidad de generar datos a velocidades elevadas. El adaptador actúa como un registro de acopio («buffer») de datos, puesto que la velocidad del bus PCI puede diferir de la capacidad de E/S del procesador. En un sistema multiprocesador (Figura 3.21b), se pueden conectar mediante adaptadores una o varias configuraciones PCI al bus de sistema del procesador. Al bus de sistema se conectan únicamente las unidades procesador/cache, la memoria principal, y los adaptadores de PCI. De nuevo, el uso de adaptadores mantiene al PCI independiente de la velocidad del procesador, y proporciona la posibilidad de recibir y enviar datos rápidamente.

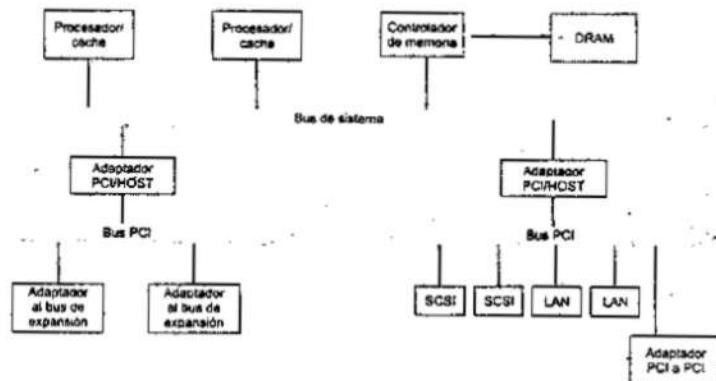
ESTRUCTURA DEL BUS

El bus PCI puede configurarse como un bus de 32 o 64 bits. La Tabla 3.3 define las 49 líneas de señal obligatorias en el PCI. Se dividen en los grupos funcionales siguientes:

- Terminales («patillas») de sistema: constituidas por los terminales de reloj y de inicio («reset»).



(a) Sistema de sobremesa típico



(b) Servidor típico

Figura 3.21. Ejemplos de configuraciones PCI.

- **Terminales de direcciones y datos:** incluye 32 líneas para datos y direcciones multiplexadas en el tiempo. Las otras líneas del grupo se utilizan para interpretar y validar las líneas de señal correspondientes a los datos y a las direcciones.
- **Terminales de control de la interfaz:** controlan la temporización de las transferencias y proporcionan coordinación entre los que las iniciaran y los destinatarios.
- **Terminales de arbitraje:** a diferencia de las otras líneas de señal del PCI, éstas no son líneas compartidas. Cada maestro del PCI tiene su par propio de líneas que lo conectan directamente al árbitro del bus PCI.
- **Terminales para señales de error:** utilizadas para indicar errores de paridad, u otros.

Tabla 3.3. Líneas de señal PCI obligatorias

Denominación	Tipo	Descripción
Terminales de sistema		
CLK	in	Proporciona la temporización para todas las transacciones y es muestrada en el flanko de subida. Se pueden utilizar frecuencias de reloj de hasta 33 MHz.
RST#	in	Hace que todos los registros, sacuencladores y señales específicas de PCI pasen a su estado de inicio.
Terminales de direcciones y datos		
AD[31:0]	t/s	Líneas multiplexadas para direcciones y datos.
C/BE[3:0]#	t/s	Señales multiplexadas de órdenes del bus y de byte activo («byte enable»). Durante la fase de datos, las líneas indican cual de los cuatro grupos de líneas de bytes transporta información válida.
PAR	t/s	Proporciona paridad para las líneas AD y C/BE, un ciclo de reloj después. El maestro proporciona PAR para las fases de dirección y escritura de datos, y el dispositivo de lectura genera PAR en la fase de lectura de datos.
Terminales de control de interfaz		
FRAME#	s/t/s	Suministradas por el maestro actual para indicar el comienzo y la duración de una transferencia. Las activa al comienzo y las desactiva cuando el maestro está preparado para empezar al final de la fase de datos.
IRDY#	s/t/s	Indicador preparado (Initiator Ready). La proporciona el maestro actual del bus (el indicador de la transacción). Durante una lectura, indica que el maestro está preparado para aceptar datos; durante una escritura indica que el dato válido está en AD.
TRDY#	s/t/s	Dispositivo preparado (Target Ready). La proporciona el dispositivo seleccionado. Durante una lectura, el dato válido está en AD; durante una escritura indica que el destino está preparado para aceptar datos.
STOP#	s/t/s	Indica que el dispositivo seleccionado desea que el maestro pare la transacción actual.
IDSEL	in	Selector de inicio de dispositivo (Initialization Device Select). Usada como señal de selección de circuito («chip select») durante las lecturas y escrituras de configuración.
DEVSEL#	in	Selector de dispositivo (Device Select). Activada por el dispositivo seleccionado cuando ha reconocido su dirección. Indica al maestro actual si se ha seleccionado un dispositivo.
Terminales de arbitraje		
REQ#	t/s	Indica al árbitro que el dispositivo correspondiente solicita utilizar el bus. Es una línea punto-a-punto específica para cada dispositivo.
GNT#	t/s	Indica al dispositivo que el árbitro le ha cedido el acceso al bus. Es una línea punto-a-punto específica para cada dispositivo.
Terminales para señales de error		
PERR#	s/t/s	Error de paridad. Indica que se ha detectado un error de paridad en los datos por parte del dispositivo en el caso de una escritura, o por parte del maestro en el caso de una lectura.
SERR#	o/d	Error del sistema. La puede activar cualquier dispositivo para comunicar errores de paridad en la dirección u otro tipo de errores críticos distintos de la paridad.

Además, la especificación del PCI define 51 señales opcionales (Tabla 3.4), divididas en los siguientes grupos funcionales:

Tabla 3.4. Líneas de señal PCI opcionales

Denominación	Tipo	Descripción
Terminales de interrupción		
INTA#	o/d	Utilizada para pedir una interrupción.
INTB#	o/d	Utilizada para pedir una interrupción; sólo tiene significado en un dispositivo multifunción.
INTC#	o/d	Utilizada para pedir una interrupción; sólo tiene significado en un dispositivo multifunción.
INTD#	o/d	Utilizada para pedir una interrupción; sólo tiene significado en un dispositivo multifunción.
Terminales de soporte de cache		
SBO#	in/out	(Snoop Backoff). Indica un acceso a una línea de cache modificada.
SDONE	in/out	(Snoop Done). Indica el estado del módulo de sondeo del bus («snoop») en el acceso actual a cache. Se activa cuando el sondeo ha terminado.
Terminales de extensión a bus de 64 bits		
AD(63:32)	t/s	Líneas multiplexadas de direcciones y datos para ampliar el bus a 64 bits.
C/BE[7:4]#	t/s	Líneas multiplexadas de órdenes y habilitación de byte. Durante la fase de dirección, las líneas proporcionan las órdenes adicionales del bus. Durante la fase de datos, las líneas indican cuál de los cuatro bytes del bus ampliado contiene datos válidos.
REQ64#	s/t/s	Utilizada para pedir una transferencia de 64 bits.
ACK64#	s/t/s	Indica que el dispositivo seleccionado está dispuesto para realizar una transferencia de 64 bits.
PAR64	t/s	Proporciona paridad par para las líneas ampliadas AD y para C/BE un ciclo de reloj después.
Terminales de test (JTAG/Boundary Scan Pins)		
TCK	in	Reloj de test. Utilizado para sincronizar la entrada y salida de la información de estado y los datos de test del dispositivo testeado (mediante Boundary Scan).
TDI	in	Entrada de test. Utilizada para introducir bit a bit los datos y las instrucciones de test en el dispositivo a testear.
TDO	out	Salida de test. Utilizada para obtener bit a bit los datos y las instrucciones de test desde el dispositivo testeado.
TMS	in	Selector de modo de test. Utilizado para establecer el estado del controlador del puerto de acceso para el test.
TRST#	in	Inicio de test (Test Reset). Utilizada para iniciar el controlador del puerto de acceso para el test.

in Señal de entrada.

out Señal de salida.

t/s Señal de E/S, bidireccional, tri-estado.

s/t/s Señal tri-estado activada sólo por un dispositivo en cada momento.

o/d Drenador abierto: permite que varios dispositivos lo comparten como en una OR cableada.

La señal se activa en el nivel inferior de tensión (activa en baja).

- **Terminales de Interrupción:** Para los dispositivos PCI que deben generar peticiones de servicio. Igual que los terminales de arbitraje, no son líneas compartidas sino que cada dispositivo PCI tiene su propia línea o líneas de petición de interrupción a un controlador de interrupciones.
- **Terminales de Soporte de Cache:** Necesarios para permitir memorias cache en el bus PCI asociadas a un procesador o a otro dispositivo. Estos terminales permiten el uso de protocolos de coherencia de cache de sondeo de bus («snoopy cache») (en el Capítulo 16 se discuten estos protocolos).
- **Terminales de Ampliación a Bus de 64 bits:** Incluye 32 líneas multiplexadas en el tiempo para direcciones y datos y se combinan con las líneas obligatorias de dirección y datos para constituir un bus de direcciones y datos de 64 bits. Hay otras líneas de este grupo que se utilizan para interpretar y validar las líneas de datos y direcciones. Por último, hay dos líneas que permiten que dos dispositivos PCI se pongan de acuerdo para usar los 64 bits.
- **Terminales de Test (JTAG/Boundary Scan):** Estas señales se ajustan al estándar IEEE 1149.1 para la definición de procedimientos de test.

ÓRDENES DEL PCI

La actividad del bus consiste en transferencias entre dos elementos, denominándose maestro al que inicia la transacción. Cuando un maestro del bus adquiere el control del mismo, determina el tipo de transferencia que se producirá a continuación. Durante la fase de direccionamiento de transferencia, se utilizan las líneas C/BE para indicar el tipo de transferencia. Los tipos de órdenes son:

- Reconocimiento de interrupción
- Ciclo especial
- Lectura de E/S
- Escritura en E/S
- Lectura de memoria
- Lectura de línea de memoria
- Lectura múltiple de memoria
- Escritura en memoria
- Escritura e invalidación de memoria
- Lectura de configuración
- Escritura de configuración
- Ciclo de dirección dual

El reconocimiento de interrupción es una orden de lectura proporcionada por el dispositivo que actúa como controlador de interrupciones en el bus PCI. Las líneas de direcciones no se utilizan en la fase de direccionamiento, y las líneas de byte activo («byte enable») indican el tamaño del identificador de interrupción a devolver.

La orden de ciclo especial se utiliza para iniciar la difusión de un mensaje a uno o más destinos.

Las órdenes de lectura de E/S y escritura en E/S se utilizan para intercambiar datos entre el módulo que inicia la transferencia y un controlador de E/S. Cada dispositivo de E/S tiene

su propio espacio de direcciones, y las líneas de direcciones se utilizan para indicar un dispositivo concreto y para especificar los datos a trasferir a o desde ese dispositivo. El concepto de direcciones de E/S se explora en el Capítulo 6.

Las órdenes de lectura y escritura en memoria se utilizan para especificar la transferencia de una secuencia de datos, utilizando uno o más ciclos de reloj. La interpretación de estas órdenes depende de si el controlador de memoria del bus PCI utiliza el protocolo PCI para transferencias entre memoria y cache, o no. Si lo utiliza, la transferencia de datos a y desde la memoria normalmente se produce en términos de líneas o bloques de cache². El uso de las tres órdenes de lectura de memoria se resumen en la Tabla 3.5. La orden de escritura en memoria se utiliza para transferir datos a memoria en uno o más ciclos de datos. La orden de escritura e invalidación de memoria transfiere datos a memoria en uno o más ciclos. Además, indica que se ha escrito en al menos una línea de cache. Esta orden permite el funcionamiento de la cache con postescritura («write back») en memoria.

Las dos órdenes de configuración permiten que un dispositivo maestro lea y actualice los parámetros de configuración de un dispositivo conectado al bus PCI. Cada dispositivo PCI puede disponer de hasta 256 registros internos, utilizados para configurar dicho dispositivo durante la inicialización del sistema.

La orden de ciclo de dirección dual se utiliza por el dispositivo que inicia la transferencia para indicar que está utilizando direcciones de 64 bits.

TRANSFERENCIAS DE DATOS

Toda transferencia de datos en el bus PCI es una transacción única, que consta de una fase de direccionamiento y una o más fases de datos. En esta discusión, se ilustra una operación de lectura típica; las operaciones de escritura se producen de forma análoga.

La Figura 3.22 muestra la temporización de una operación de lectura. Todos los eventos se sincronizan en las transiciones de bajada del reloj, cosa que sucede a la mitad de cada ciclo de reloj. Los dispositivos del bus interpretan las líneas del bus en los flancos de subida al comienzo del ciclo de bus. A continuación, se describen los eventos significativos señalados en el diagrama:

- Una vez que el maestro del bus ha obtenido el control del bus, debe iniciar la transacción activando FRAME. Esta línea permanece activa hasta que el maestro está dispuesto para terminar la última fase de datos. El maestro también sitúa la dirección de inicio en el bus de direcciones, y la orden de lectura en las líneas C/BE.

Tabla 3.5. Interpretación de las órdenes de lectura PCI

Tipo de orden de lectura	Para memoria transferible a cache	Para memoria no transferible a cache
Lectura de memoria	Secuencia de la mitad o menos de una línea	Secuencia de 2 o menos ciclos de transferencia de datos
Lectura de línea de memoria	Secuencia de más de media línea y menos de 3 líneas de cache	Secuencia de 3 a 12 ciclos de transferencia
Lectura de memoria múltiple	Secuencia de más de 3 líneas de cache	Secuencia de más de 12 ciclos de transferencia de datos

² Los principios fundamentales de la memoria cache se describen en la Sección 4.3; los protocolos de cache basados en buses se describen en la Sección 16.3.

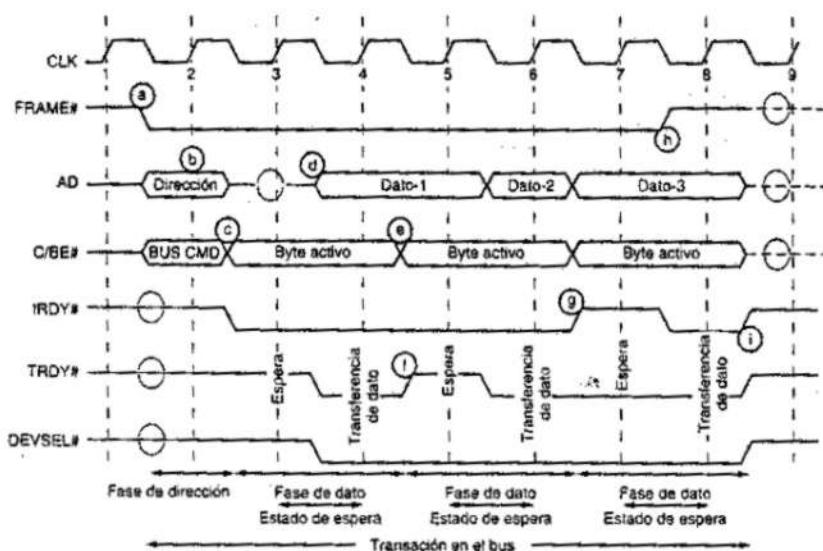


Figura 3.22. Operación de lectura PCI.

- b) Al comienzo del ciclo de reloj 2, el dispositivo del que se lee reconocerá su dirección en las líneas AD.
- c) El maestro deja libres las líneas AD del bus. En todas las líneas de señal que pueden ser activadas por más de un dispositivo, se necesita un ciclo de cambio (indicado por las dos flechas circulares) para que la liberación de las líneas de dirección permita que el bus pueda ser utilizado por el dispositivo de lectura. El maestro cambia la información de las líneas C/BE para indicar cuáles de las líneas AD se utilizan para transferir el dato direccionado (de 1 a 4 bytes). El maestro también activa IRDY para indicar que está preparado para recibir el primer dato.
- d) El dispositivo de lectura seleccionado activa DEVSEL para indicar que ha reconocido las direcciones y va a responder. Sitúa el dato solicitado en las líneas AD y activa TRDY para indicar que hay un dato válido en el bus.
- e) El maestro lee el dato al comienzo del ciclo de reloj 4 y cambia las líneas de habilitación de byte según se necesite para la próxima lectura.
- f) En este ejemplo, el dispositivo de lectura necesita algún tiempo para preparar el segundo bloque de datos para la transmisión. Por consiguiente, desactiva TRDY para señalar al maestro que no proporcionará un nuevo dato en el próximo ciclo. En consecuencia, el maestro no lee las líneas de datos al comienzo del quinto ciclo de reloj y no cambia la señal de habilitación de byte durante ese ciclo. El bloque de datos es leído al comienzo del ciclo de reloj 6.
- g) Durante el ciclo 6, el dispositivo de lectura sitúa el tercer dato en el bus. No obstante, en este ejemplo, el maestro todavía no está preparado para leer el dato (por ejemplo, puede tener lleno el registro de almacenamiento temporal). Para indicarlo, desactiva

IRDY. Esto hará que el dispositivo de lectura mantenga el tercer dato en el bus durante un ciclo de reloj extra.

- El maestro sabe que el tercer dato es el último, y por eso desactiva FRAME para indicar al dispositivo de lectura que este es el último dato a transferir. Además activa IRDY para indicar que está listo para completar esa transferencia.
- El maestro desactiva IRDY, haciendo que el bus vuelve a estar libre, y el dispositivo de lectura desactiva TRDY y DEVSEL.

ARBITRAJE

El bus PCI utiliza un esquema de arbitraje centralizado sincrónico, en el que cada maestro tiene una única señal de petición (REQ) y cesión (GNT) del bus. Estas líneas se conectan a un árbitro central (Figura 3.23), y se utiliza un simple intercambio de las señales de petición y cesión para permitir el acceso al bus.

La especificación PCI no indica un algoritmo particular de arbitraje. El árbitro puede utilizar un procedimiento de primero-en llegar-primer-en servirse, un procedimiento de cesión cíclica («round-robin»), o cualquier clase de esquema de prioridad. El maestro del PCI establece, para cada transferencia que deseé hacer, si tras la fase de dirección sigue una o más fases de datos consecutivas.

La Figura 3.24 es un ejemplo en el que se arbitra a cuál de los dispositivos A y B se cede el bus. Se produce la siguiente secuencia:

- En algún momento anterior al comienzo del ciclo de reloj 1, A ha activado su señal REQ. El árbitro muestrea esa señal al comienzo del ciclo de reloj 1.
- Durante el ciclo de reloj 1, B solicita el uso del bus activando su señal REQ.
- Al mismo tiempo, el árbitro activa GNT-A para ceder el acceso al bus a A.
- El maestro del bus A muestrea GNT-A al comienzo del ciclo de reloj 2 y conoce que se le ha cedido el acceso al bus. Además, encuentra IRDY y TRDY desactivados, indicando que el bus está libre. En consecuencia, activa FRAME y coloca la información de dirección en el bus de direcciones, y la orden correspondiente en las líneas C/BE (no mostradas). Además mantiene activa REQ-A, puesto que tiene que realizar otra transferencia después de la actual.
- El árbitro del bus muestrea todas las líneas GNT al comienzo del ciclo 3, y toma la decisión de ceder el bus a B para la siguiente transacción. Entonces activa GNT-B y desactiva GNT-A. B no podrá utilizar el bus hasta que éste no vuelva a estar libre.

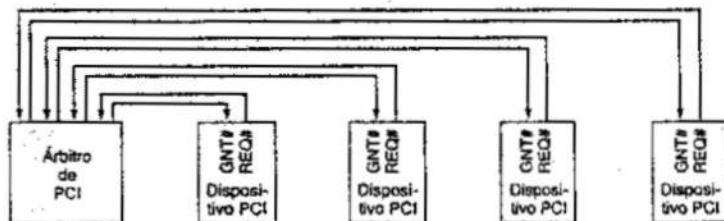


Figura 3.23. Árbitro de bus PCI.

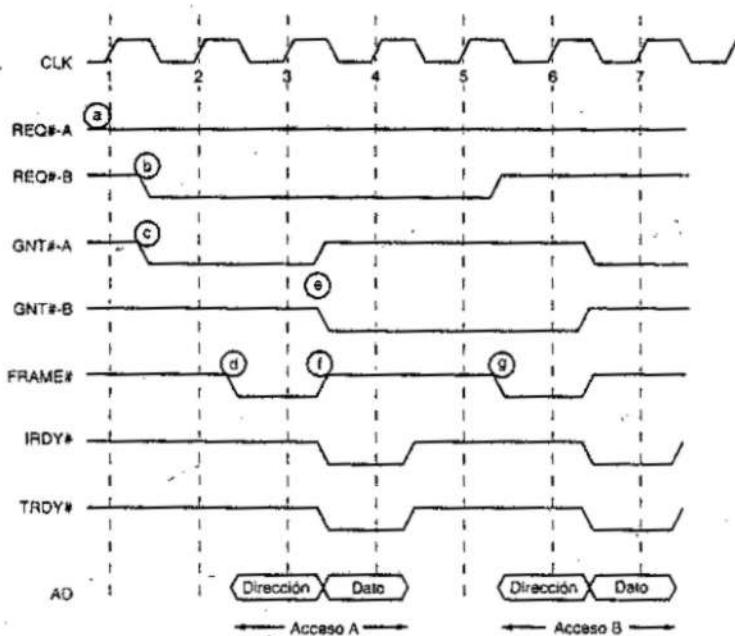


Figura 3.24. Arbitraje del bus PCI entre dos maestros.

- f) A desactiva FRAME para indicar que la última (y la única) transferencia de dato está en marcha. Pone los datos en el bus de datos y se lo indica al dispositivo destino con IRDY. El dispositivo lee el dato al comienzo del siguiente ciclo de reloj.
- g) Al comienzo del ciclo 5, B encuentra IRDY y FRAME desactivados y, por consiguiente, puede tomar el control del bus activando FRAME. Además, desactiva su línea REQ, puesto que sólo deseaba realizar una transferencia.

Posteriormente, se cede el acceso al bus al maestro A para que realice su siguiente transferencia.

Hay que resaltar que el arbitraje se produce al mismo tiempo que el maestro del bus actual está realizando su transferencia de datos. Por consiguiente, no se pierden ciclos de bus en realizar el arbitraje. Esto se conoce como *arbitraje oculto o solapado* (*hidden arbitration*).

6. LECTURAS Y SITIOS WEB RECOMENDADOS.

Sorprendentemente, la literatura sobre buses y otras estructuras de interconexión no es muy extensa. [ALEX93] contiene un tratamiento en profundidad de las estructuras de bus y de los aspectos de las transferencias en buses, incluyendo información de algunos buses científicos específicos.

El libro con la descripción más clara de PCI es [SHAN95]. [SOLA94] también contiene bastante información rigurosa de PCI.

ALEX93 Alexandridis, N. *Design of Microprocessor-Based Systems*. Englewood Cliffs, NJ: Prentice Hall, 1993.

SHAN95 Shanley, T., y Anderson, D. *PCI Systems Architecture*. Richardson, TX: Mindshare Press, 1995.

SOLA94 Solari, E., y Willse, G. *PCI Hardware and Software: Architecture and Design*. San Diego, CA: Annabooks, 1994.



SITIOS WEB RECOMENDADOS:

- PCI Special Interest Group: información acerca de las especificaciones del bus PCI y productos basados en el mismo.

3.7. PROBLEMAS

3.1. La máquina hipotética de la Figura 3.4 también tiene dos instrucciones de E/S:

0011 = Cargar AC desde E/S

0111 = Almacenar AC en E/S

En estos casos, la dirección de 12 bits identifica un dispositivo concreto de E/S. Muestre la ejecución del programa (utilizando el formato de la Figura 3.5) para el siguiente programa:

1. Cargar AC desde el dispositivo 5.
2. Sumar el contenido de la posición de memoria 940.
3. Almacenar AC en el dispositivo 6.

Asuma que el siguiente valor obtenido desde el dispositivo 5 es 3, y que la posición 940 almacena el valor 2.

- 3.2. Considere un hipotético microprocesador de 32 bits cuyas instrucciones de 32 bits están compuestas por dos campos: el primer byte contiene el código de operación (codop) y los restantes un operando inmediato o una dirección de operando.
- a) ¿Cuál es la capacidad de memoria (en bytes) direccionable directamente?
 - b) Discuta el impacto que se produciría en la velocidad del sistema si el microprocesador tiene:
 1. Un bus de dirección local de 32 bits y un bus de datos local de 16 bits, o
 2. Un bus de dirección local de 16 bits y un bus de datos local de 16 bits.
 - c) ¿Cuántos bits necesitan el contador de programa y el registro de instrucción?

Fuente: [ALEX93].

- 3.3. Considere un microprocesador hipotético que genera direcciones de 16 bits (por ejemplo, suponga que el contador de programa y el registro de dirección son de 16 bits) y tiene un bus de datos de 16 bits.
- ¿Cuál es el máximo espacio de direcciones de memoria al que el procesador puede acceder directamente, si está conectado a una «memoria de 16 bits»?
 - ¿Cuál es el máximo espacio de direcciones de memoria al que el procesador puede acceder directamente, si está conectado a una «memoria de 8 bits»?
 - ¿Qué características de la arquitectura permitirán a este procesador acceder a un «espacio de E/S» separado?
 - Si una instrucción de entrada o de salida puede especificar un número de puerto de E/S de 8 bits, ¿cuántos puertos de E/S de 8 bits puede soportar el microprocesador? Cuántos puertos de E/S de 16 bits? Explíquelo.

Fuente: [ALEX93].

- 3.4. Considere un microprocesador de 32 bits con un bus externo de 16 bits y con una entrada de reloj de 8 MHz. Asuma que el procesador tiene un ciclo de bus cuya duración mínima es igual a cuatro ciclos de reloj. ¿Cuál es la velocidad de transferencia máxima que puede sostener el microprocesador? Para incrementar sus prestaciones, ¿sería mejor hacer que su bus externo de datos sea de 32 bits, o doblar la frecuencia de reloj que se suministra al microprocesador? Establezca las suposiciones que considere y explíquelo.

Fuente: [ALEX93]

- 3.5. Considere un computador que posee un módulo de E/S que controla un teletipo con teclado e impresora. La CPU tiene los siguientes registros conectados directamente al bus de sistema:

INPR:	Registro de entrada - 8 bits
OUTPR:	Registro de salida - 8 bits
FGI:	Indicador («flag») de entrada - 1 bit
FGO:	Indicador («flag») de salida - 1 bit
IEN:	Habilitación de interrupción - 1 bit

La entrada desde el teclado y la salida a la impresora del teletipo están controlados por el módulo de E/S. El teletipo es capaz de codificar un símbolo alfanumérico mediante una palabra de 8 bits y decodificar una palabra de 8 bits en un símbolo alfanumérico.

- Describa cómo la CPU, utilizando el primero de los cuatro registros enumerados anteriormente, puede realizar una E/S con el teletipo.
- Describa cómo se puede realizar dicha operación de manera más eficiente si además se utiliza IEN.

- 3.6. La Figura 3.25 muestra un esquema de arbitraje distribuido que puede utilizarse con Multibus I. Los módulos se conectan en cadena («daisy-chain») según su orden de prioridad. El módulo que está más a la izquierda en el diagrama recibe una *prioridad de bus* constante a través de la señal BPRN, indicando que ningún módulo de mayor prioridad solicita el bus. Si el módulo no desea utilizar el bus, activa su línea de *salida de prioridad del bus* (BPRO). Al comienzo de un ciclo de reloj, cualquier módulo puede pedir el control del bus, poniendo en baja su línea BPRO. Esto hace que pase a baja la línea BPRN

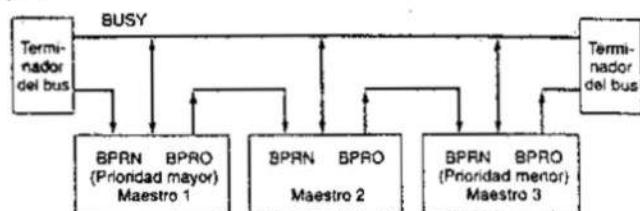


Figura 3.25. Arbitraje distribuido en el Multibus I.

del siguiente módulo de la cadena, que a su vez debe poner a baja su línea BPRO. Así, la señal se propaga a lo largo de la cadena. Al final de esta respuesta en cadena, únicamente hay un módulo cuya BPRN está activada y cuya BPRO no. Este módulo es el que tiene prioridad. Si al comienzo de un ciclo de bus el bus no está ocupado (BUSY no activada), el módulo que tiene prioridad puede tomar el control del bus activando la señal BUSY.

Se necesita una cierta cantidad de tiempo para que la señal BPR se propague desde el módulo de prioridad más alta hasta el de más baja. ¿Debe este tiempo ser menor que el ciclo de reloj? Explíquelo.

- 3.7. El bus VAX SBI utiliza un esquema de arbitraje distribuido síncrono. Cada dispositivo SBI (es decir, procesador, memoria y adaptador de Unibus) tiene una prioridad única, y se le asigna una única línea de solicitud de transferencia (TR. Transfer Request). El SBI tiene 16 de dichas líneas (TR0, TR1, ..., TR15), siendo TR0 la de prioridad más elevada. Cuando un dispositivo quiere utilizar el bus, realiza una reserva de un ciclo de bus futuro, activando su línea TR durante el ciclo de bus en curso. Al final del ciclo en curso, cada dispositivo con una reserva pendiente examina las líneas TR; el que tiene más alta prioridad utiliza el siguiente ciclo de bus.

Como máximo se pueden conectar al bus 17 dispositivos. El dispositivo con prioridad 16 no tiene línea TR. ¿Por qué?

- 3.8. Paradójicamente, el dispositivo de prioridad más baja tiene el tiempo de espera medio más bajo. Por esta razón, a la CPU se le da usualmente la prioridad más baja en el SBI. ¿Por qué el dispositivo de prioridad 16 tiene normalmente el menor tiempo de espera medio? ¿En qué circunstancias esto no sería cierto?
- 3.9. Dibuja y explícate un diagrama de tiempos para una operación de escritura en un bus PCI (similar a la Figura 3.22).

APÉNDICE 3A. DIAGRAMAS DE TIEMPO

En este capítulo, los diagramas de tiempo se utilizan para ilustrar las secuencias de eventos y las dependencias entre eventos. Para el lector que no esté familiarizado con los diagramas de tiempo, este apéndice proporciona una breve explicación.

La comunicación entre los dispositivos conectados a un bus se produce a través de un conjunto de líneas capaces de transmitir señales. Se pueden transmitir dos niveles diferentes de señal (niveles de tensión), representando un 0 binario y un 1 binario. Un diagrama de tiempo muestra el nivel de la señal en una línea en función del tiempo (Figura 3.26a).

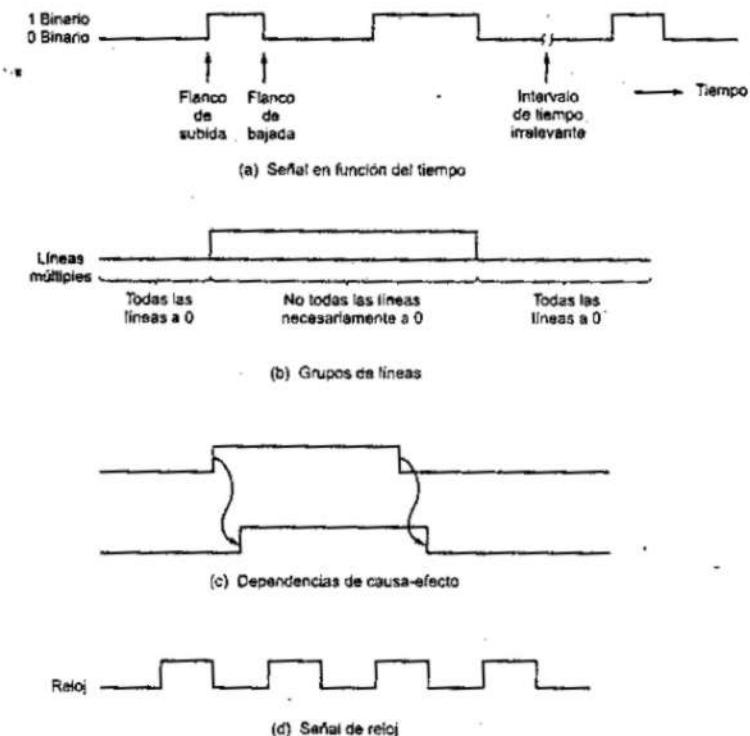


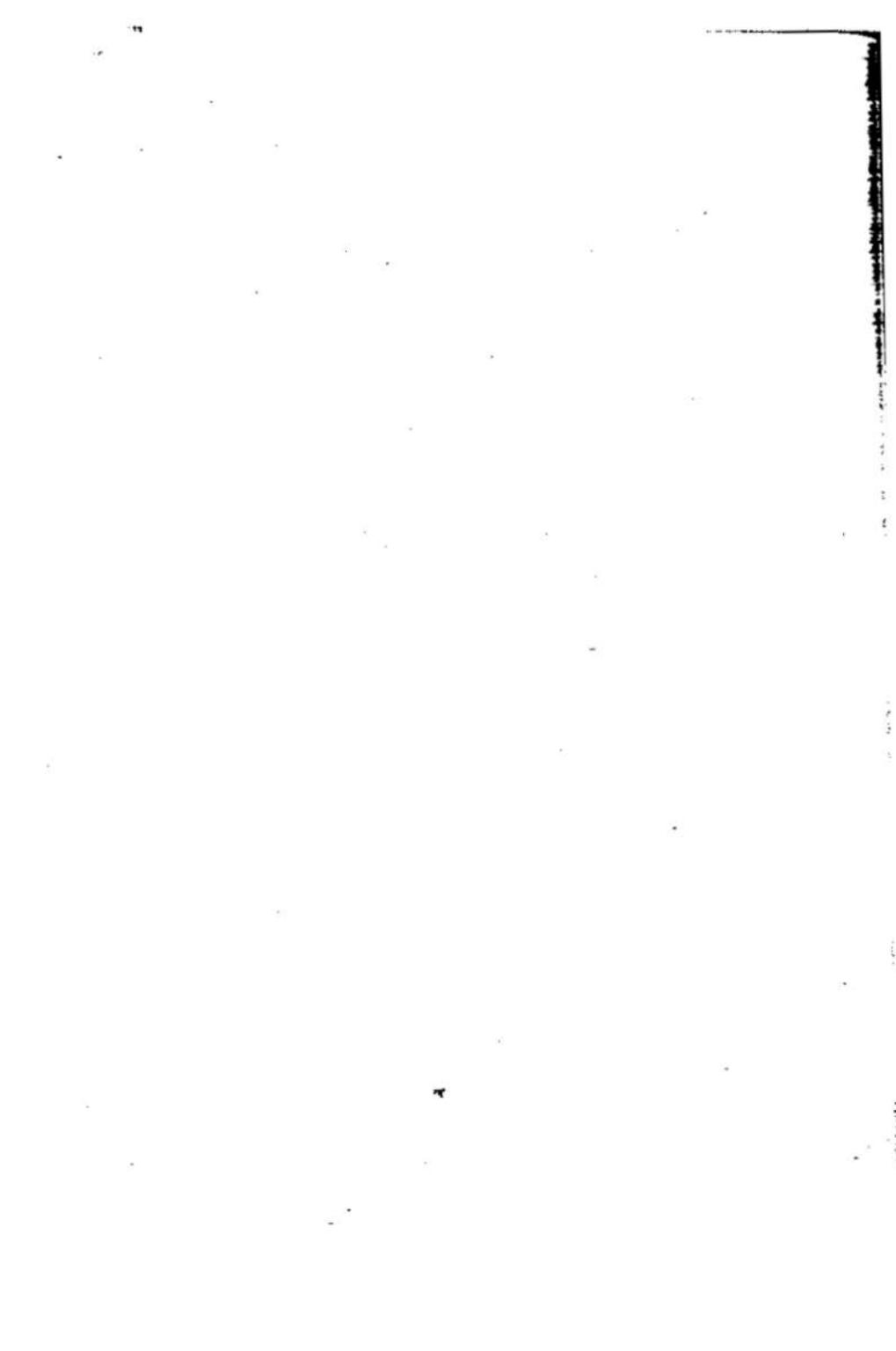
Figura 3.26. Diagramas de tiempo.

Por convención, el nivel 1 de la señal binaria se representa por un nivel más alto que el nivel binario 0. Usualmente, el 0 binario es el valor por defecto (o implícito). Es decir, si no se está transmitiendo un dato ni ninguna otra señal, entonces el nivel en la línea es el que representa al 0 binario. Una transición de 0 a 1 en la señal se denomina frecuentemente *flanco de subida de la señal* (*leading edge*); una transición de 1 a 0 se denomina *flanco de bajada* (*trailing edge*). Por claridad, a menudo las transiciones en las señales se dibujan como si ocurrieran instantáneamente. De hecho, una transición requiere un tiempo no nulo, pero este tiempo de transición es usualmente pequeño comparado con la duración de un nivel en la señal. En un diagrama de tiempos puede ocurrir que transcurra una cantidad de tiempo variable, o en todo caso irrelevante, entre dos eventos de interés. Esto se representa con un corte en la línea de tiempo.

A veces, las señales se representan agrupadas (Figura 3.26b). Por ejemplo, si se transfiere un dato de un byte cada vez, se necesitan ocho líneas. Generalmente, no es esencial conocer el valor exacto que se transfiere en dicho grupo, sino más bien si las señales están presentes o no.

Una transición de señal en una línea, provoca que un dispositivo conectado ocasione cambios de señal en otras líneas. Por ejemplo, si un módulo de memoria detecta una señal de control de lectura (transición a 0 ó a 1), situará las señales correspondientes a los datos en las líneas de datos. Estas relaciones de causa-efecto dan lugar a secuencias de sucesos. Las flechas se utilizan para mostrar estas dependencias en los diagramas de tiempo (Figura 3.26c).

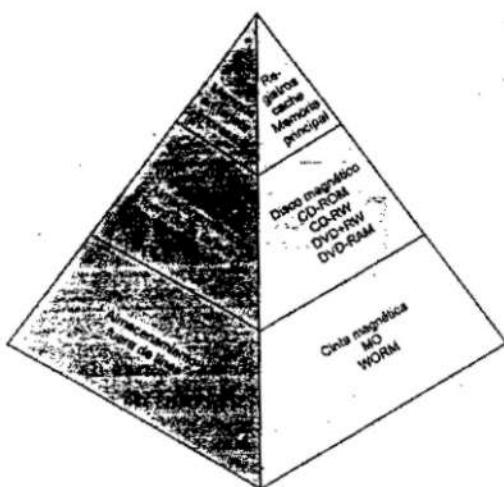
A menudo, el bus de sistema contiene una línea de reloj. Un reloj electrónico se conecta a la línea de reloj y proporciona una secuencia repetitiva y regular de transiciones (Figura 3.26d). Puede haber otros eventos sincronizados con la señal de reloj.



CAPÍTULO 4

Memoria interna

- 4.1. Conceptos básicos sobre sistemas de memoria de computadores
 - Características de los sistemas de memoria
 - Jerarquía de memoria
 - 4.2. Memoria principal semiconductor
 - Tipos de memorias semiconductoras de acceso aleatorio
 - Organización
 - Lógica del chip
 - Encapsulado de los chips
 - Organización en módulos
 - Corrección de errores
 - 4.3. Memoria cache
 - Principios básicos
 - Elementos de diseño de la cache
 - 4.4. Organización de la cache en el Pentium II y el PowerPC
 - Organización de cache en el Pentium II
 - Organización de cache en el PowerPC
 - 4.5. Organización avanzada de memorias DRAM
 - DRAM mejorada
 - DRAM cache
 - DRAM síncrona
 - DRAM rambus
 - RamLink
 - 4.6. Lecturas y sitios Web recomendados
 - 4.7. Problemas
- Apéndice 4A. Prestaciones de las memorias de dos niveles**
- Localidad
 - Funcionamiento de la memoria de dos niveles
 - Prestaciones



- La memoria de un computador tiene una organización jerárquica. En el nivel superior (el más próximo al procesador) están los registros del procesador. A continuación se encuentra una cache o dos niveles de cache, denominados L1 y L2. Posteriormente la memoria principal, normalmente construida con memorias dinámicas de acceso aleatorio (DRAM). Todas ellas se consideran memorias internas del computador. La jerarquía prosigue con la memoria externa, siendo el siguiente nivel, usualmente, un disco duro fijo y uno o más niveles de soportes extraibles, tales como cartuchos ZIP, discos ópticos y cintas magnéticas.
- A medida que descendemos en la jerarquía de memoria, disminuye el coste por bit, aumenta la capacidad y crece el tiempo de acceso. Sería deseable poder utilizar sólo la memoria más rápida, pero al ser la más costosa, se llega a un compromiso entre tiempo de acceso y coste, empleando más cantidad de memoria más lenta. La estrategia a seguir consiste en organizar los datos y los programas en memoria de manera que las palabras de memoria necesarias estén normalmente en la memoria más rápida.
- En general, es probable que la mayoría de los accesos futuros a la memoria principal, por parte del procesador, sean a posiciones accedidas recientemente. Por eso, la cache retiene automáticamente una copia de algunas de las palabras de la DRAM utilizadas recientemente. Si la cache se diseña adecuadamente, la mayor parte del tiempo el procesador solicitará palabras de memoria que están ya en la cache.

Las memorias de los computadores, aunque parezcan conceptualmente sencillas, presentan tal vez la más amplia diversidad de tipos, tecnología, estructura, prestaciones y coste, de entre todos los componentes de un computador. Ninguna tecnología es óptima para satisfacer las necesidades de memoria de un computador. En consecuencia, un computador convencional está equipado con una jerarquía de subsistemas de memoria, algunos internos (directamente accesibles por el procesador) y otros externos (accesibles por el procesador mediante módulos de entrada/salida).

Este capítulo se centra en el estudio de la memoria interna, mientras que el Capítulo 5 se dedicará a la memoria externa. Para comenzar, en la primera sección de este capítulo examinaremos características clave de las memorias de un computador. Después, veremos los subsistemas de memoria principal semiconductor, incluyendo memorias ROM, DRAM y SRAM. A continuación, analizaremos un elemento esencial de cualquier computador moderno: la memoria cache. Después de esto, estaremos en condiciones de volver a la memoria DRAM para examinar arquitecturas DRAM más avanzadas.

4.1. CONCEPTOS BASICOS SOBRE SISTEMAS DE MEMORIA DE COMPUTADORES

CARACTERÍSTICAS DE LOS SISTEMAS DE MEMORIA

El complejo tema de las memorias es más abordable si clasificamos los sistemas de memoria según sus características clave. Las más importantes se listan en la Tabla 4.1.

El término **ubicación**, que aparece en la Tabla 4.1, indica si la memoria es interna o externa al computador. La memoria interna suele identificarse con la memoria principal. Sin embargo, hay además otras formas de memoria interna. El procesador necesita su propia memoria local en forma de registros (véase, por ejemplo, la Figura 2.3). Además, como veremos, la unidad de control del procesador también puede necesitar su propia memoria interna. Posponemos la discusión de estos dos últimos tipos de memoria interna para capítulos posteriores. La memoria externa consta de dispositivos periféricos de almacenamiento, tales como discos y cintas, que son accesibles por la CPU a través de controladores de E/S.

Una característica obvia de las memorias es su **capacidad**. Para memorias internas se expresa normalmente en términos de bytes (1 byte = 8 bits) o de palabras. Longitudes de pa-

Tabla 4.1. Características clave de los sistemas de memoria de computadores

Ubicación	Prestaciones
CPU	Tiempo de acceso
Interna (principal)	Tiempo de ciclo
Externa (secundaria)	Velocidad de transferencia
Capacidad	Dispositivo físico
Tamaño de la palabra	Semiconductor
Número de palabras	Soporte magnético
Unidad de transferencia	Soporte óptico
Palabra	Magneto-óptico
Bloque	Características físicas
Método de acceso	Volátil/no volátil
Acceso secuencial	Borrable/no borrable
Acceso directo	Organización
Acceso aleatorio	
Acceso asociativo	

bra comunes son 8, 16 y 32 bits. La capacidad de las memorias externas se suele expresar en bytes.

Un concepto relacionado es la **unidad de transferencia**. Para memorias internas, la unidad de transferencia es igual al número de líneas de entrada/salida de datos del módulo de memoria. A menudo es igual a la longitud de palabra, pero puede no serlo. Para aclararlo, consideremos tres conceptos relacionados con la memoria interna:

- **Palabra:** Es la unidad «natural» de organización de la memoria. El tamaño de la palabra suele coincidir con el número de bits utilizados para representar números y con la longitud de las instrucciones. Por desgracia, hay muchas excepciones. Por ejemplo, el CRAY-1 tiene una longitud de palabra de 64 bits pero utiliza una representación de números enteros de 24 bits. El VAX tiene una gran variedad de longitudes de instrucción, expresadas como múltiplos de bytes, y una longitud de palabra de 32 bits.
- **Unidades direccionables:** En muchos sistemas la unidad direccionable es la palabra. Sin embargo, algunos de ellos permiten direccionar a nivel de bytes. En cualquier caso, la relación entre la longitud A de una dirección y el número N de unidades direccionables, es $2^A = N$.
- **Unidad de transferencia:** Para la memoria principal, es el número de bits que se leen o escriben en memoria a la vez. La unidad de transferencia no tiene por qué coincidir con una palabra o con una unidad direccionable. Para la memoria externa, los datos se transfieren normalmente en unidades más grandes que la palabra, denominadas «bloques».

Otro distintivo entre tipos de memorias es el **método de acceso**, que incluye las siguientes variantes:

- **Acceso secuencial:** La memoria se organiza en unidades de datos llamadas «registros». El acceso debe realizarse con una secuencia lineal específica. Se hace uso de información almacenada de direccionamiento, que permite separar los registros y ayudar en el proceso de recuperación de datos. Se utiliza un mecanismo de lectura/escritura compartida, que debe ir trasladándose desde su posición actual a la deseada, pasando y obviando cada registro intermedio. Así pues, el tiempo necesario para acceder a un registro dado es muy variable. Las unidades de cinta que se tratan en el Capítulo 5 son de acceso secuencial.
- **Acceso directo:** Como en el caso de acceso secuencial, el directo tiene asociado un mecanismo de lectura/escritura. Sin embargo, los bloques individuales o registros tienen una dirección única basada en su dirección física. El acceso se lleva a cabo mediante un acceso directo a una vecindad dada, seguido de una búsqueda secuencial, bien contando, o bien esperando hasta alcanzar la posición final. De nuevo, el tiempo de acceso es variable. Las unidades de disco, que se tratan en el Capítulo 5, son de acceso directo.
- **Acceso aleatorio («random»):** Cada posición direccionable de memoria tiene un único mecanismo de acceso, cableado físicamente. El tiempo para acceder a una posición dada es constante e independiente de la secuencia de accesos previos. Por tanto, cualquier posición puede seleccionarse aleatoriamente y puede ser direccionada y accedida directamente. La memoria principal y algunos sistemas de cache son de acceso aleatorio.
- **Asociativa:** Es una memoria del tipo de acceso aleatorio, que permite hacer una comparación de ciertas posiciones de bits dentro de una palabra buscando que coincidan con unos valores dados, y hacer esto para todas las palabras simultáneamente. Una palabra es, por tanto, recuperada, basándose en una porción de su contenido, en lugar de su dirección. Como en las memorias de acceso aleatorio convencionales, cada posición tiene su propio mecanismo de direccionamiento, y el tiempo de recuperación de un dato

es una constante independiente de la posición o de los patrones de acceso anteriores. Las memorias cache, descritas en la Sección 4.3, pueden emplear acceso asociativo.

Desde el punto de vista del usuario, las dos características más importantes de una memoria son su capacidad y sus prestaciones. Se utilizan tres parámetros de medida de prestaciones:

- **Tiempo de acceso:** Para memorias de acceso aleatorio es el tiempo que tarda en realizarse una operación de escritura o de lectura, es decir, el tiempo que transcurre desde el instante en el que se presenta una dirección a la memoria hasta que el dato ha sido memorizado o está disponible para su uso. Para memorias de otro tipo, el tiempo de acceso es el que se tarda en situar el mecanismo de lectura/escritura en la posición deseada.
- **Tiempo de ciclo de memoria:** Este concepto se aplica principalmente a las memorias de acceso-aleatorio y consiste en el tiempo de acceso y algún tiempo más que se requiere, antes de que pueda iniciarse un segundo acceso a memoria. Este tiempo adicional puede que sea necesario para que finalicen las transiciones en las líneas de señal o para regenerar los datos en el caso de lecturas destructivas.
- **Velocidad de transferencia:** Es la velocidad a la que se pueden transferir datos a, o desde, una unidad de memoria. Para memorias de acceso aleatorio coincide con el inverso del tiempo de ciclo. Para otras memorias se utiliza la siguiente relación:

$$T_N = T_A + \frac{N}{R}$$

donde:

T_N = Tiempo medio de escritura o de lectura de N bits

T_A = Tiempo de acceso medio

N = Número de bits

R = Velocidad de transferencia, en bits por segundo (bps)

Se han empleado **dispositivos físicos** muy diversos de memoria. Las más comunes en la actualidad son las memorias semiconductoras, las memorias de superficie magnética, utilizadas para discos y cintas, y las memorias ópticas y magneto-ópticas.

Del almacenamiento de datos son importantes varias **características físicas**. En memorias volátiles la información se va perdiendo o desaparece, cuando se desconecta la alimentación. En las memorias no volátiles la información, una vez grabada, permanece sin deteriorarse hasta que se modifique intencionadamente; no se necesita la fuente de alimentación para retener la información. Las memorias de superficie magnética son no volátiles. Las memorias semiconductoras pueden ser volátiles o no volátiles. Las memorias no borrables no pueden modificarse, salvo que se destruya la unidad de almacenamiento. Las memorias semiconductoras de este tipo se conocen por el nombre de *memorias de sólo lectura* (ROM, Read Only Memory).

En memorias de acceso aleatorio, su organización es un aspecto clave de diseño. Por organización se entiende su disposición o estructura física en bits para formar palabras. La estructura más obvia no es siempre la utilizada como explicaremos pronto.

JERARQUÍA DE MEMORIA

Las restricciones de diseño de la memoria de un computador se pueden resumir en tres cuestiones: ¿cuánta capacidad?, ¿cómo de rápida?, ¿de qué coste?

Así pues, la estrategia en principio funciona, pero sólo si se aplican las condiciones (a) a (d) anteriores. Empleando diversas tecnologías se tiene todo un espectro de sistemas de memoria que satisfacen las condiciones (a) a (c). Afortunadamente, la condición (d) es también generalmente válida.

La base para la validez de la condición (d) es el principio conocido como *localidad de las referencias* [DENN68]. En el curso de la ejecución de un programa, las referencias a memoria por parte del procesador, tanto para instrucciones como para datos, tienden a estar agrupadas. Los programas, normalmente, contienen un número de bucles iterativos y subrutinas. Cada vez que se entra en un bucle o una subrutina, hay repetidas referencias a un pequeño conjunto de instrucciones. De manera similar, las operaciones con tablas o con matrices llevan accesos a un conjunto de palabras de datos agrupadas. En períodos de tiempo largos, las agrupaciones («clusters») en uso cambian, pero en períodos de tiempo cortos, el procesador trabaja principalmente con agrupaciones fijas de referencias a memoria.

De acuerdo con lo anterior, es posible organizar los datos a través de la jerarquía, de tal manera que el porcentaje de accesos a cada nivel siguiente más bajo, sea sustancialmente menor que al nivel anterior. Considerese el ejemplo de dos niveles ya presentado, y que la memoria del nivel 2 contiene todos los datos e instrucciones de programa. Las agrupaciones actuales pueden ubicarse temporalmente en el nivel 1. De vez en cuando, una de las agrupaciones del nivel 1 tendrá que ser devuelta al nivel 2, a fin de que deje sitio para que entre otra nueva agrupación al nivel 1. En promedio, sin embargo, la mayoría de las referencias serán a instrucciones y datos contenidos en el nivel 1.

Este principio puede aplicarse a través de más de dos niveles de memoria, como sugiere la jerarquía mostrada en la Figura 4.1. El tipo de memoria más rápida, pequeña y costosa, lo constituyen los registros internos al procesador. Un procesador suele contener unas cuantas docenas de tales registros, aunque algunas máquinas contienen cientos de ellos. Descendiendo dos niveles, la memoria principal es el principal sistema de memoria interna del computador. Cada posición de memoria principal tiene una única dirección, y la mayoría de las instrucciones máquina referencian una o más direcciones de la memoria principal. La memoria principal es normalmente ampliada con una cache, que es más pequeña y rápida. La cache no suele estar visible al programador y, realmente, tampoco al procesador. Es un dispositivo para escalonar las transferencias de datos entre memoria principal y los registros del procesador, a fin de mejorar las prestaciones.

Las tres formas de memoria que acabamos de describir son, normalmente, volátiles y de tecnología semiconductora. El uso de tres niveles aprovecha la variedad existente de tipos de memorias semiconductoras, que difieren en velocidad y coste. El almacenamiento de datos de forma más permanente se hace en dispositivos de memoria masiva, de los cuales, los más comunes son el disco duro y los dispositivos extraíbles, tales como discos extraíbles, cintas, y dispositivos ópticos de almacenamiento. Las memorias externas no volátiles o permanentes se denominan también «memorias secundarias o auxiliares». Se utilizan para almacenar programas y ficheros de datos, y suelen estar visibles al programador sólo en términos de ficheros y registros, en lugar de bytes aislados o de palabras. El disco se emplea además para proporcionar una ampliación de la memoria principal, conocida como «memoria virtual», que será tratada en el Capítulo 7.

En la jerarquía pueden incluirse otras formas de memoria. Por ejemplo, los grandes computadores de IBM incluyen una forma de memoria interna, conocida como «almacenamiento expandido». Este utiliza una tecnología semiconductora que es más lenta y menos costosa que la de la memoria principal. Estrictamente hablando, esta memoria no encaja en la jerarquía, sino que es una rama lateral: los datos pueden transferirse entre memoria principal y almacenamiento expandido, pero no entre éste y la memoria externa. Otras formas de

memoria secundaria incluyen los discos ópticos y los magneto-ópticos. Finalmente, se pueden, en efecto, añadir más niveles a la jerarquía mediante software. Una parte de la memoria principal puede utilizarse como almacenamiento intermedio («buffer») para guardar temporalmente datos que van a ser volcados en disco. Esta técnica, a veces denominada «cache de disco»¹, mejora las prestaciones de dos maneras:

- Las escrituras en disco se hacen por grupos. En lugar de muchas transferencias cortas de datos, tenemos unas cuantas transferencias largas. Esto mejora las prestaciones del disco y minimiza la participación del procesador.
- Algunos datos destinados a ser escritos como salidas pueden ser referenciados por un programa antes de que sean volcados en disco. En ese caso, los datos se recuperan rápidamente desde la cache software, en lugar de hacerlo lentamente de disco.

El Apéndice 4A examina las implicaciones sobre prestaciones de las estructuras de memoria multinivel.

4.2. MEMORIA PRINCIPAL SEMICONDUCTORA

En computadores antiguos, la forma más común de almacenamiento de acceso aleatorio para la memoria principal consistía en una matriz de pequeños anillos ferromagnéticos denominados *núcleos*. Es por esto que la memoria principal recibía a menudo el nombre de *núcleo* («core»), un término que perdura en la actualidad. El advenimiento de la microelectrónica, y sus ventajas, acabó con las memorias de núcleos. Hoy en día es casi universal el uso de memorias semiconductoras para la memoria principal. En esta sección se exploran aspectos clave de esta tecnología.

TIPOS DE MEMORIAS SEMICONDUCTORAS DE ACCESO ALEATORIO

Todos los tipos de memorias que exploraremos en esta sección son de acceso aleatorio. Es decir, las palabras individuales de la memoria son accedidas directamente mediante lógica de direccionamiento cableada interna.

La Tabla 4.2 lista los principales tipos de memorias semiconductoras. La más común es la denominada *memoria de acceso aleatorio* (RAM, Random-Access Memory). Este es, por supuesto, un mal uso del término, ya que todas las memorias listadas en la tabla son de acceso aleatorio. Una característica distintiva de las RAM es que es posible, tanto leer datos, como escribir rápidamente nuevos datos en ellas. Tanto la lectura como la escritura se ejecutan mediante señales eléctricas.

La otra característica distintiva de una RAM es que es volátil. Una RAM debe estar continuamente alimentada. Si se interrumpe la alimentación, se pierden los datos. Así pues, las RAM pueden utilizarse sólo como almacenamiento temporal.

Las tecnologías de RAM se dividen en dos variantes: estáticas y dinámicas. Una RAM *dinámica* está hecha con celdas, que almacenan los datos como cargas en condensadores. La presencia o ausencia de carga en un condensador se interpretan como el 1 o el 0 binarios. Ya que los condensadores tienen una tendencia natural a descargarse, las RAM dinámicas requieren refrescos periódicos para mantener memorizados los datos. En una RAM *estática*, los valores binarios se almacenan utilizando configuraciones de puertas que forman biestables

¹ La cache de disco generalmente es una técnica software, y no es estudiada en este libro. Véase [STAL98] para una discusión del tema.

Tabla 4.2. Tipos de memorias semiconductoras

Tipo de memoria	Clase	Borrado	Mecanismos de escritura	Volatilidad
Memoria de acceso aleatorio (RAM)	Memoria de lectura/escritura	Eléctricamente por bytes	Eléctricamente	Volátil
Memoria de sólo lectura (ROM)	Memoria de sólo lectura	No posible	Mediante máscaras	No volátil
ROM programable (PROM)				
PROM borrible (EPROM)		Luz ultravioleta, chip completo		No volátil
Memoria FLASH	Memoria de sobre-todo-lectura	Eléctricamente por bloques	Eléctricamente	
ROM borrible eléctricamente (EEPROM)		Eléctricamente por bytes		

«flip-flops»). Una descripción de los biestables puede verse en el Apéndice A. Una RAM estática retendrá sus datos en tanto se mantenga alimentada.

Tanto las RAM estáticas como las dinámicas son volátiles. Una celda de memoria RAM dinámica es más simple que una estática y, en consecuencia, más pequeña. Por tanto, las RAM dinámicas son más densas (celdas más pequeñas = más celdas por unidad de superficie) y más económicas que las correspondientes RAM estáticas. Por otra parte, una RAM dinámica requiere circuitería para el refresco. En memorias grandes, el coste fijo de la circuitería de refresco se ve más que compensado por el menor coste de las celdas RAM dinámicas. Así pues, las RAM dinámicas tienden a ser las preferidas para memorias grandes. Un último detalle es que las RAM estáticas son generalmente algo más rápidas que las dinámicas.

En claro contraste con las RAM están las *memorias de sólo lectura* (ROM, Read-Only Memory). Como su nombre sugiere, una ROM contiene un patrón permanente de datos que no puede alterarse. Aunque es posible leer de una ROM, no se pueden escribir nuevos datos en ella. Una aplicación importante de las ROM es la microprogramación, estudiada en la Parte IV del libro. Otras aplicaciones son:

- Subrutinas de biblioteca para funciones de uso frecuente
- Programas del sistema
- Tablas de funciones

Cuando se requiere un tamaño modesto, la ventaja de una ROM es que el programa o los datos estarían permanentemente en memoria principal, y nunca sería necesario cargarlos desde un dispositivo de memoria secundaria.

Una ROM se construye como cualquier otro chip de circuito integrado, con los datos cableados en el chip durante el proceso de fabricación. Esto presenta dos problemas:

- La etapa de inserción de datos implica unos costes fijos relativamente grandes, tanto si se va a fabricar una, o miles de copias de una misma ROM.
- No se permiten fallos. Si uno de los bits es erróneo, debe desecharse la tirada completa de memorias ROM.

Cuando se necesitan sólo unas pocas ROM con un contenido particular, una alternativa más económica es la *ROM programable* (PROM). Al igual que las ROM, las PROM son no volátiles y pueden grabarse sólo una vez. Para la PROM, el proceso de escritura se lleva a cabo eléctricamente, y puede realizarlo el suministrador o el cliente con posterioridad a la fabricación del chip original. Se requiere un equipo especial para el proceso de escritura o «programación». Las PROM proporcionan flexibilidad y comodidad. Las ROM siguen siendo atractivas para tiradas de producción de gran volumen.

Otra variante de memoria de sólo lectura es la memoria de sobre-todo-lectura («read-mostly»), que es útil para aplicaciones en las que las operaciones de lectura son bastante más frecuentes que las de escritura, pero para las que se requiere un almacenamiento no volátil. Hay tres formas comunes de memorias de sobre-todo-lectura: EPROM, EEPROM, y memorias «flash».

- La memoria de sólo lectura programable y borrable ópticamente (EPROM, Erasable Programmable Read-Only Memory) se lee y escribe eléctricamente como la PROM. Sin embargo, antes de la operación de escritura, todas las celdas de almacenamiento deben primariamente borrarse a la vez, mediante exposición del chip encapsulado a radiación ultravioleta. Este proceso de borrado puede realizarse repetidas veces; cada borrado completo puede durar hasta 20 minutos. Así pues, las EPROM pueden modificarse múltiples veces y, al igual que las ROM y las PROM, retienen su contenido, en teoría indefinidamente. Para una capacidad similar, una EPROM es más costosa que una PROM, pero tiene como ventaja adicional la posibilidad de actualizar múltiples veces su contenido.

Una forma más atractiva de memoria de sobre-todo-lectura es la memoria de sólo lectura programable y borrable eléctricamente (EEPROM: Electrically Erasable Programmable Read-Only Memory). Esta es una memoria de sobre-todo lectura en la que se puede escribir en cualquier momento sin borrar su contenido anterior; sólo se actualiza el byte o bytes direccionalmente. La operación de escritura lleva considerablemente más tiempo que la de lectura; del orden de cientos de microsegundos por byte. La EEPROM combina la ventaja de ser no volátil, con la flexibilidad de ser actualizable *in situ* utilizando las líneas de datos, de direcciones y de control de un bus ordinario. Las EEPROM son más costosas que las EPROM y también menos densas, admitiendo menos bits por chip.

Otra forma de memoria semiconductor es la memoria flash (denominada así por la velocidad con la que puede reprogramarse). Introducidas a mediados de los 80, las memorias flash se encuentran, en coste y en funcionalidad, entre las EPROM y las EEPROM. Al igual que las EEPROM, las flash utilizan una tecnología de borrado eléctrico. Una memoria flash puede borrarse entera en unos cuantos segundos, mucho más rápido que las EPROM. Además, es posible borrar sólo bloques concretos de memoria, en lugar de todo el chip. Sin embargo, las memorias flash no permiten borrar a nivel de byte. Al igual que las EPROM, las flash utilizan sólo un transistor por bit, consiguiéndose las altas densidades (comparadas con las EEPROM) que alcanzan las EPROM.

ORGANIZACIÓN

El elemento básico de una memoria semiconductor es la celda de memoria. Aunque se utilizan diversas tecnologías electrónicas, todas las celdas de memoria comparten ciertas propiedades:

- Presentan dos estados estables (o semiestables), que pueden emplearse para representar el 1 y el 0 binarios.
- Puede escribirse en ellas (al menos una vez) para fijar su contenido.
- Pueden leerse para detectar su estado.



Figura 4.3. Funcionamiento de una celda de memoria.

La Figura 4.3 describe el funcionamiento de una celda de memoria. Lo más común es que la celda tenga tres terminales para transportar señales eléctricas. El terminal de selección, como su nombre indica, selecciona la celda para la operación de escritura o de lectura. El terminal de control indica el tipo de operación. Para la escritura, el tercer terminal proporciona la señal que fija el estado de la celda a 1 o a 0. En una lectura, el tercer terminal se utiliza como salida del estado de la celda. Los detalles sobre estructura interna, funcionamiento y temporización de la celda de memoria en cada tecnología específica de circuitos integrados, están más allá del alcance de este libro. Para nuestros propósitos, daremos por sentado que las celdas individuales pueden seleccionarse para operaciones de lectura y de escritura.

LÓGICA DEL CHIP

Como otros circuitos integrados, las memorias semiconductoras vienen en chips encapsulados (Figura 2.7). Cada chip contiene una matriz de celdas de memoria.

En toda la jerarquía de memoria vimos que existen compromisos de velocidad, capacidad y coste. Estos compromisos existen también cuando consideramos la organización de las celdas de memoria y del resto de funciones lógicas de un chip de memoria. Para las memorias semiconductoras, uno de los aspectos fundamentales de diseño es el número de bits de datos que pueden ser leídos/escritos a la vez. En un extremo está la estructura en la que la disposición física de las celdas de la matriz es la misma que la disposición lógica (tal y como la percibe el procesador) de las palabras de memoria. La matriz está organizada en W palabras de B bits cada una. Por ejemplo, un chip de 16 Mbits podría estar estructurado en 1 Mpalabra de 16 bits. En el otro extremo está la estructura denominada «un-bit-por-chip», en la que los datos se escriben/leen por bits. A continuación describimos la estructura de un chip de memoria DRAM; la estructura de una ROM integrada es similar, aunque más sencilla.

La Figura 4.4 muestra una organización típica de DRAM de 16 Mbits. En este caso se escriben o leen 4 bits a la vez. Lógicamente, la matriz de memoria está estructurada en 4 matrices cuadradas de 2.048×2.048 elementos. Son posibles varias disposiciones físicas. En cualquier caso, los elementos de la matriz conectan tanto a líneas horizontales (de fila) como a verticales (de columna). Cada línea horizontal conecta al terminal de selección de cada celda en la correspondiente fila; y cada línea vertical conecta al terminal Entrada-Datos/Detección («Data-In/Sense») de cada celda en la correspondiente columna.

Las líneas de direcciones suministran la dirección de la palabra a seleccionar. Se requiere un total de $\log_2 W$ líneas. En nuestro ejemplo se necesitan 11 líneas de direcciones para seleccionar una de entre 2.048 filas. Estas 11 líneas entran en un decodificador de filas, que tiene 11 líneas de entrada y 2.048 de salida. La lógica del decodificador activa una única salida de entre las 2.048, definida por el patrón de bits de las 11 líneas de entrada ($2^{11} = 2.048$).

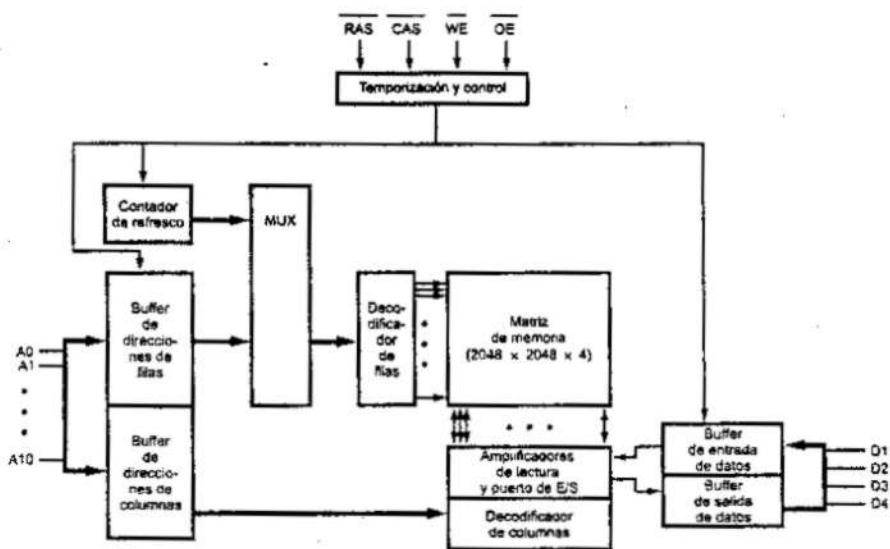


Figura 4.4. DRAM típica de 16 megabitos (4M x 4).

Otro grupo de 11 líneas de direcciones selecciona una de entre 2.048 columnas, con cuatro bits por columna. Se utilizan cuatro líneas para la entrada y salida de cuatro bits, a, y desde, un buffer de datos. Para la entrada (escritura), cada línea de bit se activa a 1 o a 0, de acuerdo con el valor de la correspondiente línea de datos. Para salida (lectura), el valor de cada línea de bit se pasa a través de un amplificador de lectura (término que emplearemos para referirnos al inglés «sense amplifier») y se presenta en la correspondiente línea de datos. La línea de fila selecciona la fila de celdas que es utilizada para lectura o escritura.

Ya que en esta DRAM se escriben/leen sólo cuatro bits, debe haber varias DRAM conectadas al controlador de memoria, a fin de escribir/leer una palabra de datos en el bus.

Obsérvese que hay sólo 11 líneas de direcciones (A0-A10), la mitad del número necesario para una matriz de 2.048×2.048 . Esto se hace así para ahorrar en número de terminales. Las señales de las 22 líneas de direcciones necesarias se transforman con lógica de selección externa al chip y se multiplexan en 11 líneas de direcciones. Primero se proporcionan al chip 11 señales de dirección que definen la dirección de fila de la matriz, y después se presentan las otras 11 señales para la dirección de columna. Estas señales se acompañan por las de selección de dirección de fila (RAS) y de selección de dirección de columna (CAS), que temporizan el chip.

Como comentario, el uso de direccionamiento multiplexado y de matrices cuadradas dan lugar a que el tamaño de memoria se cuadriplice con cada nueva generación de chips de memoria. Un terminal adicional dedicado a direccionamiento duplica el número de filas y de columnas y, por tanto, el tamaño del chip de memoria crece en un factor 4.

La Figura 4.4 también indica la inclusión de la circuitería de refresco. Todas las DRAM requieren operaciones de refresco. Una técnica simple de refresco consiste en inhabilitar el chip DRAM mientras se refrescan todas las celdas. El contador de refresco recorre todos

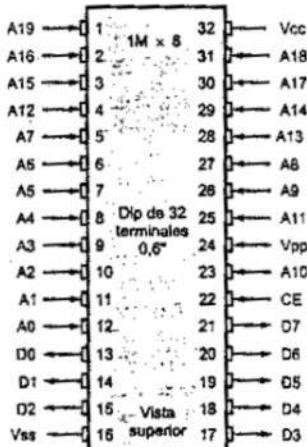
los valores de fila. Para cada valor, las salidas de dicho contador se conectan al decodificador de filas y se activa la línea RAS. Esto hace que se refresquen todas las celdas de una fila a la vez.

ENCAPSULADO DE LOS CHIPS

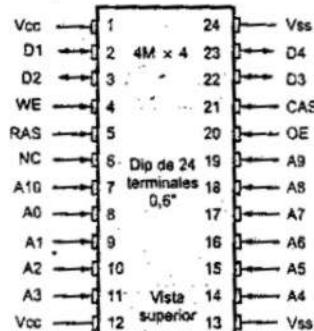
Como se mencionó en el Capítulo 2, los circuitos integrados se montan en cápsulas con terminales que los conectan con el mundo exterior.

La Figura 4.5a muestra un ejemplo de EPROM encapsulada, un chip de 8 Mbits organizados en $1M \times 8$. En este caso, la estructura es de una-palabra-por-chip. El chip encapsulado tiene 32 terminales, siendo éste uno de los tamaños estándar de encapsulado. Los terminales transfieren las siguientes señales:

- La dirección de la palabra a la que se accede. Para 1M de palabras, se necesita un total de $20 (2^{20} = 1M)$ terminales (A0-A19).
- El dato a leer, con 8 líneas (D0-D7).
- La línea de alimentación del chip (Vcc).
- Un terminal de tierra (Vss).
- Un terminal de habilitación de chip (CE, Chip Enable). Ya que puede haber varios chips de memoria, todos conectados al mismo bus de direcciones, el terminal CE se utiliza para indicar si la dirección es válida o no para el chip. El terminal CE se activa mediante lógica, cuya entrada son los bits de orden más alto del bus de direcciones (es decir, bits de direcciones superiores al A19).
- Una tensión de programación (Vpp) que se suministra durante la programación de la memoria (operaciones de escritura).



(a) EPROM de 8 Mbits.



(b) DRAM de 16 Mbits.

Figura 4.5. Terminales y señales típicas de un chip encapsulado de memoria.

La Figura 4.5b muestra la configuración de terminales de una DRAM típica de 16 Mbits organizada en $4M \times 4$. Hay varias diferencias respecto de un chip de ROM. Ya que una RAM puede ser actualizada, los terminales de datos son de entrada/salida. Los terminales de habilitación de escritura (WE, Write Enable) y de habilitación de salida (OE, Output Enable) indican si se trata de una operación de escritura o de lectura. Debido al acceso por filas y columnas de la DRAM, y a que las direcciones están multiplexadas, sólo se necesitan 11 terminales para especificar los 4M de combinaciones fila/columna ($2^{11} \times 2^{11} = 2^{22} = 4M$). La función de los terminales de selección de dirección de fila (RAS) y de selección de dirección de columna (CAS) ha sido descrita con anterioridad.

ORGANIZACIÓN EN MÓDULOS

Si un chip de RAM contiene 1 bit por palabra, se necesitarán claramente al menos un número de chips igual al número de bits por palabra. Como ejemplo, la Figura 4.6 muestra cómo

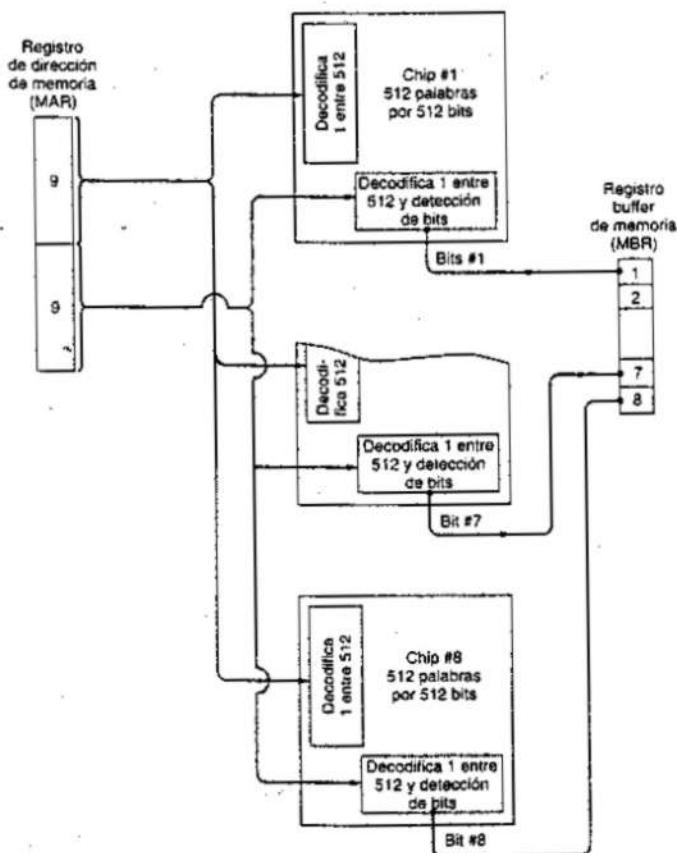


Figura 4.6. Organización de una memoria de 256 Kbytes.

podría organizarse un módulo de memoria de 256K palabras de 8 bits. Para 256K palabras se necesitan 18 bits, que se suministran al módulo desde alguna fuente externa (por ejemplo las líneas de direcciones de un bus al que está ligado el módulo). La dirección se presenta a 8 chips de $256K \times 1$ bit, cada uno de los cuales proporciona la entrada/salida de 1 bit.

Esta estructura funciona cuando el tamaño de memoria es igual al número de bits por chip. En caso de necesitar una memoria mayor, se requiere utilizar una matriz de chips. La Figura 4.7 muestra la posible organización de una memoria de 1M de palabras de 8 bits. En este caso, tenemos cuatro columnas de chips, donde cada columna contiene 256K palabras dispuestas como en la Figura 4.6. Para 1M de palabras, se necesitan 20 líneas de direcciones. Los 18 bits menos significativos se conectan a los 32 módulos. Los 2 bits de orden más alto son entradas a un módulo lógico de selección de grupo, que envía una señal de habilitación de chip a una de las cuatro columnas de módulos.

CORRECCIÓN DE ERRORES

Una memoria semiconductora está sujeta a errores. Estos pueden clasificarse en fallos permanentes («hard») y errores transitorios u ocasionales («soft»). Un *fallo permanente* corresponde a un defecto físico, de tal modo que la celda o celdas de memoria afectadas no pueden almacenar datos de manera segura, quedándose ancladas a 0 o a 1, o comutando erróneamente entre 0 y 1. Los errores permanentes pueden estar causados por funcionamiento en condiciones adversas, defectos de fabricación y desgaste. Un *error transitorio* es un evento aleatorio no destructivo que altera el contenido de una o más celdas de almacenamiento sin dañar a la memoria. Los errores transitorios pueden deberse a problemas de la fuente de alimentación o a partículas alfa. Estas partículas provienen de emisión radiactiva y son lamentablemente frecuentes debido a que, en pequeñas cantidades, hay núcleos radiactivos en casi todos los materiales. Obviamente, los errores, tanto permanentes como transitorios, no son nada deseables, y la mayoría de los sistemas de memoria modernos incluyen lógica para detectar y corregir errores.

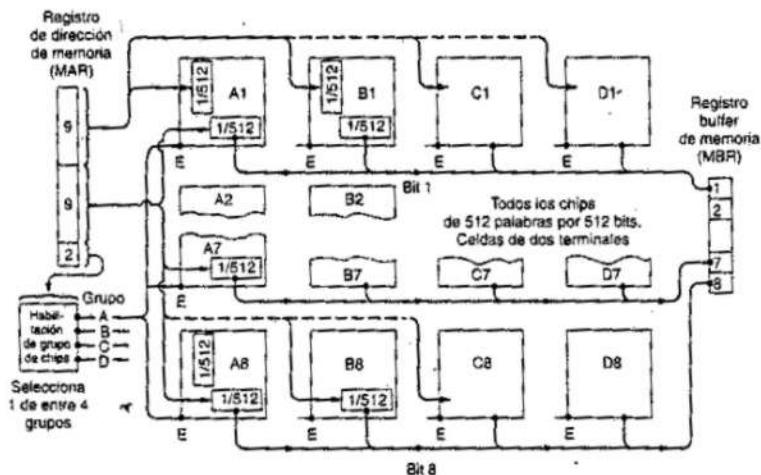


Figura 4.7. Organización de una memoria de 1M byte.

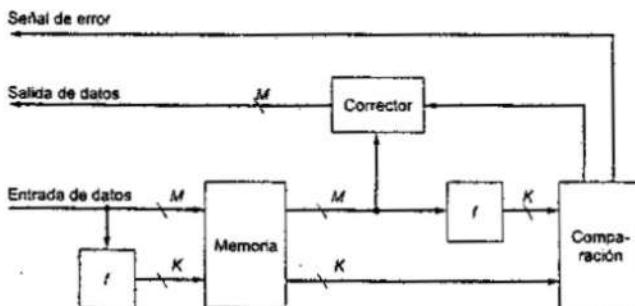


Figura 4.8. Código corrector de errores.

La Figura 4.8 ilustra, en términos generales, cómo se lleva a cabo el proceso. Cuando se van a escribir datos en memoria, se realiza un cálculo con los datos, representado por la función f , para producir un código. Se almacenan tanto los datos como el código. Así, si se va a almacenar una palabra de datos de M bits, y el código tiene una longitud de K bits, el tamaño real de la palabra almacenada es de $M + K$ bits.

Cuando se va a leer una palabra previamente almacenada, se utiliza el código para detectar errores, y puede incluso corregirlos. Se genera un nuevo código de K bits a partir de los M bits de datos, que se compara con los bits de código captados de memoria. Esta comparación produce uno de estos tres resultados posibles:

- No se detectan errores. Los bits de datos captados se envian al exterior.
- Se detecta un error y es posible corregirlo. Se dan a un corrector los bits de datos más los bits de corrección de error, lo que produce el conjunto reducido de M bits a ser enviados fuera.
- Se detecta un error, pero no es posible corregirlo. Se informa de esta situación.

Los códigos que operan de esta manera se denominan «códigos correctores de errores». Un código se caracteriza por el número de bits de error de una palabra que puede corregir y detectar.

El código corrector de errores más sencillo es el *código de Hamming*, ideado por Richard Hamming en los laboratorios Bell. La Figura 4.9 ilustra el uso de este código mediante diagramas de Venn, con palabras de 4 bits ($M = 4$). Al interseccarse tres círculos se tienen siete compartimentos. Asignamos los 4 bits del dato a los compartimentos interiores (Figura 4.9a). Los restantes compartimentos se rellenan con los denominados *bits de paridad*. Cada bit de paridad se elige de tal manera que el número total de unos en su círculo sea par (Figura 4.9b). Así pues, ya que el círculo A incluye tres unos del dato, el bit de paridad se pone a 1 en dicho círculo. Ahora, si un error cambia uno de los bits de datos (Figura 4.9c), se encuentra fácilmente. Comprobando los bits de paridad, se encuentran discrepancias en los círculos A y C, pero no en B. Sólo uno de los siete compartimentos está en A y en C pero no en B. El error puede pues corregirse, modificando el bit de dicho compartimento.

Para clarificar ideas, desarrollaremos un código que puede detectar y corregir errores de un solo bit en palabras de 8 bits.

Para empezar determinemos cuán largo debe ser el código. Con referencia a la Figura 4.8, la lógica de comparación recibe como entrada dos valores de K bits. La comparación bit a

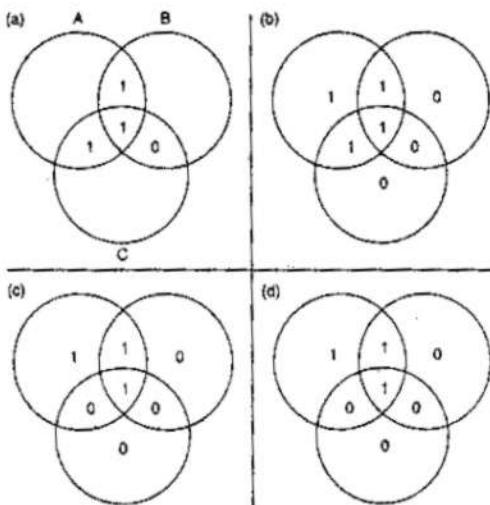


Figura 4.9. Código corrector de errores de Hamming.

bit se hace mediante la OR-exclusiva de las dos entradas. El resultado se denomina *palabra de síndrome*. Cada bit del síndrome es 0 o 1 según que haya o no coincidencia en esa posición de bit para las dos entradas.

La palabra de síndrome tiene, pues, una longitud de K bits, y tiene un rango entre 0 y $2^K - 1$. El valor 0 indica que no se ha detectado error, dejando $2^K - 1$ valores para indicar, si hay error, qué bit fue el erróneo. Ahora, ya que el error podría ocurrir en cualquiera de los M bits de datos o de los K bits de comprobación, se debe cumplir:

$$2^K - 1 \geq M + K$$

Esta ecuación da el número de bits necesarios para corregir el error, de un solo bit cualquiera, en una palabra que contenga M bits de datos. La Tabla 4.3 lista el número de bits de comprobación necesarios para diversas longitudes de palabra.

Tabla 4.3. Aumento de la longitud de palabra con la corrección de errores

Bits de datos	Corrección de errores simples			Corrección de errores simples/ detección de errores dobles	
	Bits de comprobación	% incremento		Bits de comprobación	% Incremento
8	4	50		5	62,5
16	5	31,25		6	37,5
32	6	18,75		7	21,875
64	7	10,94		8	12,5
128	8	6,25		9	7,03
256	9	3,52		10	3,91

Posición de bit	Número de posición	Bit de comprobación	Bit de datos
12	1 1 0 0	M8	
11	1 0 1 1	M7	
10	1 0 1 0	M6	
9	1 0 0 1	M5	
8	1 0 0 0 C8		C8
7	0 1 1 1	M4	
6	0 1 1 0	M3	
5	0 1 0 1	M2	
4	0 1 0 0 C4		C4
3	0 0 1 1	M1	
2	0 0 1 0 C2		C2
1	0 0 0 1 C1		C1

Figura 4.10. Posiciones de los bits de datos y de comprobación.

Según la tabla, vemos que una palabra de 8 bits de datos requiere 4 bits de comprobación. Por conveniencia, sería deseable generar un síndrome de 4 bits con las siguientes características:

- Si el síndrome contiene sólo ceros, no se ha detectado error.
- Si el síndrome contiene sólo un bit puesto a 1, ha ocurrido un error en uno de los cuatro bits de comprobación. No se requiere corrección.
- Si el síndrome contiene más de un bit puesto a 1, entonces el valor numérico de dicho síndrome indica la posición del bit de dato erróneo. Se invierte dicho bit de dato para corregirlo.

Para conseguir estas características, los bits de datos y de comprobación se distribuyen en una palabra de 12 bits, como se muestra en la Figura 4.10. Las posiciones de bit están numeradas de 1 a 12. A los bits de comprobación se asignan aquellas posiciones de bit cuyos números son potencias de dos. Los bits de comprobación se calculan como sigue:

$$\begin{aligned}
 C1 &= M1 \oplus M2 \oplus M4 \oplus M5 \oplus M7 \\
 C2 &= M1 \oplus M3 \oplus M4 \oplus M6 \oplus M7 \\
 C4 &= M2 \oplus M3 \oplus M4 \oplus M8 \\
 C8 &= M5 \oplus M6 \oplus M7 \oplus M8
 \end{aligned}$$

donde los símbolos \oplus designan la operación OR-exclusiva.

Cada bit de comprobación opera sobre todo bit de datos cuyo número de posición contiene un 1 en la correspondiente posición de columna. Así pues, las posiciones de bit de datos 3, 5, 7, 9 y 11 contienen el término 2^0 ; las posiciones 3, 6, 7, 10 y 11 contienen el término 2^1 ; las posiciones 5, 6, 7 y 12 contienen el término 2^2 ; y las posiciones 9, 10, 11 y 12 contienen el término 2^3 . Visto de otra forma, la posición de bit i es comprobada por aquellas posiciones de bit C_j , tal que $\sum i = j$. Por ejemplo, la posición 7 es comprobada por los bits en las posiciones 4, 2 y 1; y $7 = 4 + 2 + 1$.

Verifiquemos, con un ejemplo, que el esquema anterior funciona. Supongamos que la palabra de entrada de 8 bits es 00111001, con el bit M1 de dato en la posición más a la derecha. Los cálculos son los siguientes:

$$C_1 = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1$$

$$C_2 = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1$$

$$C_4 = 0 \oplus 0 \oplus 1 \oplus 0 = 1$$

$$C_8 = 1 \oplus 1 \oplus 0 \oplus 0 = 0$$

Supongamos ahora que el bit de datos 3 se ve afectado por un error, cambiando de 0 a 1. Cuando se recalculan los bits de comprobación, se tiene:

$$C_1 = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1$$

$$C_2 = 1 \oplus 1 \oplus 1 \oplus 1 \oplus 0 = 0$$

$$C_4 = 0 \oplus 1 \oplus 1 \oplus 0 = 0$$

$$C_8 = 1 \oplus 1 \oplus 0 \oplus 0 = 0$$

Cuando se comparan los nuevos bits de comprobación con los antiguos, se genera la palabra de síndrome:

C8	C4	C2	C1
0	1	1	1
$\oplus 0$	0	0	1
0	1	1	0

El resultado es 0110, indicando que la posición de bit 6, que contiene el bit 3 del dato, es errónea.

La Figura 4.11 ilustra el cálculo anterior. Los bits de datos y de comprobación se ubican adecuadamente en la palabra de 12 bits. Escribiendo en binario, y de arriba abajo, el número de posición de cada bit de datos en su respectiva columna, los unos de cada fila indican los bits de datos comprobados por el bit de comprobación correspondiente al peso binario de dicha fila. Ya que el resultado se ve afectado sólo por unos, sólo las columnas cuyo bit de datos es 1 (en las filas correspondientes a la palabra almacenada y la captada) han sido marcadas con un trazo cerrado para su mejor identificación. El valor de cada bit de comprobación (indicados en la columna de la derecha) es 1, si hay un número impar de unos en su correspondiente fila. Los resultados que se muestran son para los bits de datos originales y para los que incluyen el error.

Posición de bit	12	11	10	9	8	7	6	5	4	3	2	1
Bit de datos	M8	M7	M6	M5	M4	M3	M2	M1	C4	C2	C1	
Bit de comprobación					C8							
Palabra almacenada como:	1	1	1	1	0	0	0	0	0	0	0	C8 0
Palabra captada como:	1	0	0	0	1	1	1	1	0	0	0	C4 1
	0	1	1	0	1	1	1	0	1	1	1	C2 1
	0	1	0	1	1	0	1	0	1	1	1	C1 1
Palabra almacenada como:	0	1	0	1	1	0	1	0	1	1	1	
Palabra captada como:	0	0	1	1	1	0	1	1	0	1	1	
	0	0	1	1	1	0	1	1	0	1	1	C8 0
	1	1	1	1	0	0	0	0	0	0	0	C4 0
	1	0	0	0	1	1	1	1	0	0	0	C2 0
	0	1	1	0	1	1	0	0	1	1	1	C1 0
	0	1	0	1	1	0	1	0	1	1	1	

Figura 4.11. Generación de los bits de comprobación.

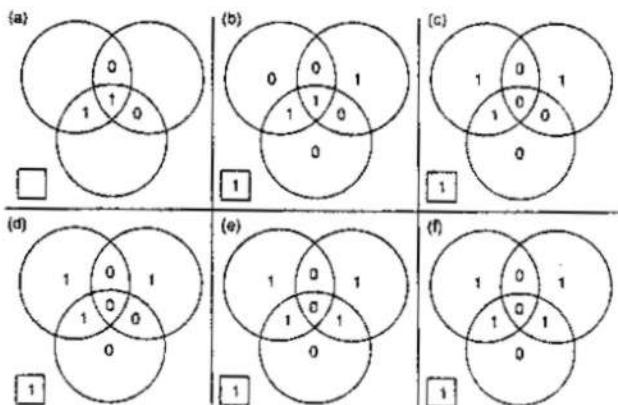


Figura 4.12. Código SEC-DED de Hamming.

El código que acabamos de describir es conocido como *código corrector de errores simples* (SEC). Es más común equipar las memorias semiconductoras con un *código corrector de errores simples y detector de errores dobles* (SEC-DED). Como muestra la Tabla 4.3, estos códigos necesitan un bit más que los SEC.

La Figura 4.12 ilustra, para una palabra de datos de 4 bits, cómo funciona un código SEC-DED. La secuencia de la figura muestra que, si ocurren dos errores (Figura 4.12c), el procedimiento de chequeo que conocíamos informa erróneamente (d), y empeora el problema creando un tercer error (e). Para superar el problema, se añade un octavo bit, cuyo valor se fija de manera que el número total de unos en el diagrama sea par. El bit de paridad añadido captura el error (f).

Un código corrector de errores mejora la seguridad de la memoria a costa de complejidad adicional. Con una estructura de un-bit-por-chip, es generalmente adecuado considerar un código SEC-DED. Por ejemplo, los IBM 30xx utilizan un código SEC-DED de 8 bits por cada 64 bits de datos en memoria principal. Así, el tamaño de la memoria principal es realmente un 12% mayor que el aparente para el usuario. Los computadores VAX utilizan un código SEC-DED de 7 bits por cada 32 bits de memoria; un 22% de incremento. Diversas memorias DRAM actuales emplean 9 bits de comprobación por cada 128 bits de datos; un 7% de incremento [SHAR97].

4.3. MEMORIA CACHE

PRINCIPIOS BÁSICOS

El objetivo de la memoria cache es lograr que la velocidad de la memoria sea lo más rápida posible, consiguiendo al mismo tiempo un tamaño grande al precio de memorias semiconductoras menos costosas. El concepto se ilustra en la Figura 4.13. Hay una memoria principal relativamente grande y más lenta, junto con una memoria cache más pequeña y rápida. La cache contiene una copia de partes de la memoria principal. Cuando el procesador intenta leer una palabra de memoria, se hace una comprobación para determinar si la palabra está en la cache. Si es así, se entrega dicha palabra al procesador. Si no, un bloque de memoria principal, consistente en un cierto número de palabras, se transfiere a la cache y, después,

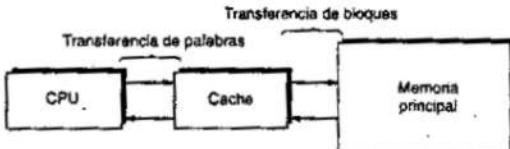


Figura 4.13. Memorias cache y principal.

palabra es entregada al procesador. Debido al fenómeno de localidad de las referencias, cuando un bloque de datos es captado por la cache para satisfacer una referencia a memoria simple, es probable que se hagan referencias futuras a otras palabras del mismo bloque.

La Figura 4.14 describe la estructura de un sistema de memoria cache/principal. La memoria principal consta de hasta 2^n palabras direccionables; teniendo cada palabra una única dirección de n bits. Esta memoria la consideramos dividida en un número de bloques de longitud fija, de K palabras por bloque. Es decir, hay $M = 2^n/K$ bloques. La cache consta de C líneas de K palabras cada una, y el número de líneas es considerablemente menor que el número de bloques de memoria principal ($C \ll M$). En todo momento, un subconjunto de los bloques de memoria reside en líneas de la cache. Ya que hay más bloques que líneas, una línea dada no puede dedicarse únicamente a un bloque. Por consiguiente, cada línea incluye una etiqueta que identifica qué bloque particular está siendo almacenado. La etiqueta es usualmente una porción de la dirección de memoria principal, como describiremos más adelante en esta sección.

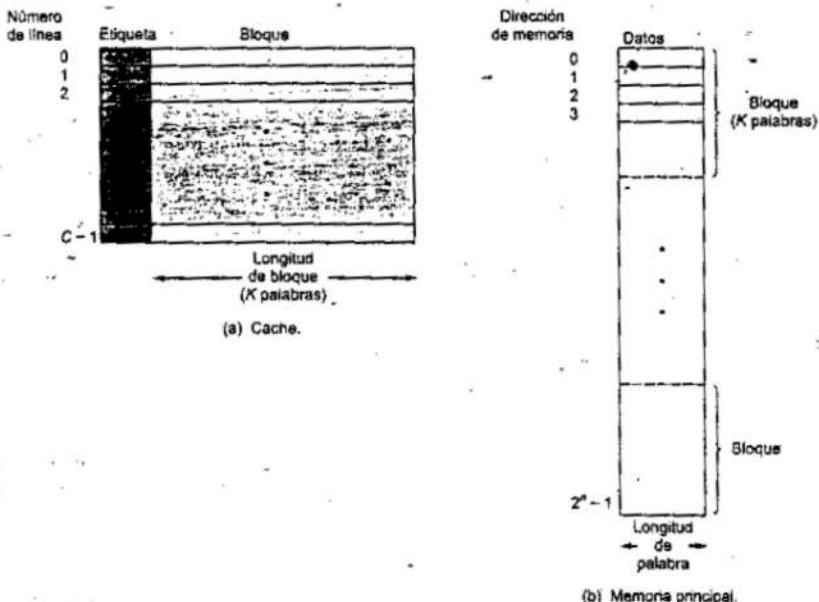


Figura 4.14. Estructura de memoria cache/principal.

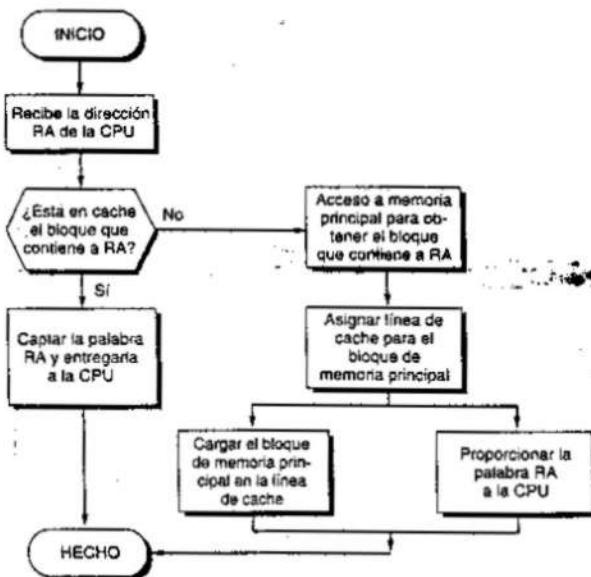


Figura 4.15. Operación de lectura de cache.

La Figura 4.15 ilustra una operación de lectura. El procesador genera la dirección, RA, de una palabra a leer. Si la palabra está en la cache, es entregada al procesador. Si no, el bloque que contiene dicha palabra se carga en la cache, y la palabra es llevada después al procesador. La Figura 4.15 indica cómo estas dos últimas operaciones se realizan en paralelo, y refleja la organización mostrada en la Figura 4.16, que es típica en las organizaciones de cache actuales. En ella, la cache conecta con el procesador mediante líneas de datos, de control y de direcciones. Las líneas de datos y de direcciones conectan también con buffers de datos y de direcciones que las comunican con un bus del sistema, a través del cual se accede a la memoria principal. Cuando ocurre un acierto de cache, los buffers de datos y de direcciones se inhabilitan, y la comunicación tiene lugar sólo entre procesador y cache, sin tráfico en el bus. Cuando ocurre un fallo de cache, la dirección deseada se carga en el bus del sistema, y el dato es llevado, a través del buffer de datos, tanto a la cache como al procesador. En otras formas de organización, la cache se interpone físicamente entre el procesador y la memoria principal para todas las líneas de datos, direcciones y control. En este caso, frente a un fallo de cache, la palabra deseada es primero leída por la cache y después transferida desde ésta al procesador.

El Apéndice 4A contiene un análisis de los parámetros de prestaciones relativos al uso de la cache.

ELEMENTOS DE DISEÑO DE LA CACHE

Aunque hay muy diversas implementaciones de cache, existen unos cuantos criterios básicos de diseño que sirven para clasificar y diferenciar entre arquitecturas de cache. La Tabla 4.4 lista algunos elementos clave.

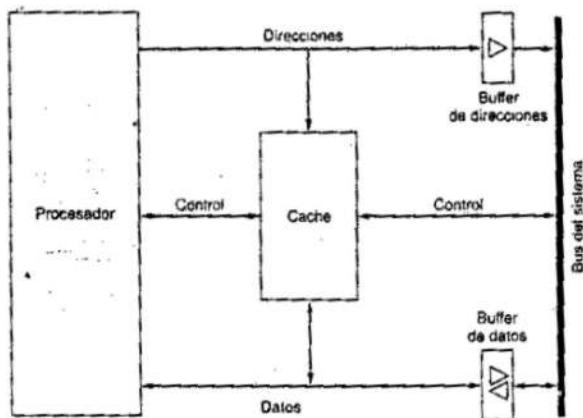


Figura 4.16. Organización típica de cache.

Tamaño de cache

El primer elemento, el tamaño de cache, ha sido ya tratado. Nos gustaría que el tamaño fuera lo suficientemente pequeño como para que el coste total medio por bit se aproximara al de la memoria principal sola, y que fuera lo suficientemente grande como para que el tiempo de acceso medio total fuera próximo al de la cache. Hay otras muchas motivaciones para minimizar el tamaño de la cache. Cuanto más grande es, mayor es el número de pueras implicadas en direccionar la cache. El resultado es que caches grandes tienden a ser ligeramente más lentas que las pequeñas (incluso estando fabricadas con la misma tecnología de circuito integrado y con la misma ubicación en el chip o en la tarjeta de circuito impreso). El tamaño de cache está también limitado por las superficies disponibles de chip y de tarjeta. En el Apéndice 4A se puntuiza que diversos estudios han sugerido que los tamaños óptimos de cache se encuentran entre 1K y 512K palabras. Como las prestaciones de la cache son muy sensibles al tipo de tarea, es imposible predecir un tamaño «óptimo».

Tabla 4.4. Elementos de diseño de la cache

Tamaño de cache	Política de escritura
Función de correspondencia	
Directa	Escr. inmediata
Asociativa	Post-escritura
Asociativa por conjuntos	Escr. única
Algoritmo de sustitución	Tamaño de línea
Utilizado menos recientemente (LRU)	Número de caches
Primer en entrar-primer en salir (FIFO)	Uno o dos niveles
Utilizado menos frecuentemente (LFU)	Unificada o partida
Aleatorio	

Función de correspondencia

Ya que hay menos líneas de cache que bloques de memoria principal, se necesita un algoritmo que haga corresponder bloques de memoria principal a líneas de cache. Además, se requiere algún medio de determinar qué bloque de memoria principal ocupa actualmente una línea dada de cache. La elección de la función de correspondencia determina cómo se organiza la cache. Pueden utilizarse tres técnicas: directa, asociativa y asociativa por conjuntos. Examinamos a continuación cada una de ellas. En cada caso veremos la estructura general y un ejemplo concreto. Para los tres casos, el ejemplo incluye los siguientes elementos:

- La cache puede almacenar 64 Kbytes.
- Los datos se transfieren entre memoria principal y la cache en bloques de 4 bytes. Esto significa que la cache está organizada en $16K = 2^{14}$ líneas de 4 bytes cada una.
- La memoria principal es de 16Mbytes, con cada byte directamente direccionable mediante una dirección de 24 bits ($2^{24} = 16M$). Así pues, al objeto de realizar la correspondencia, podemos considerar que la memoria principal consta de 4M bloques de 4 bytes cada uno.

La técnica más simple, denominada **correspondencia directa**, consiste en hacer corresponder cada bloque de memoria principal a sólo una línea posible de cache. La Figura 4.17 ilustra el mecanismo general. La correspondencia se expresa como:

$$i = j \text{ módulo } m$$

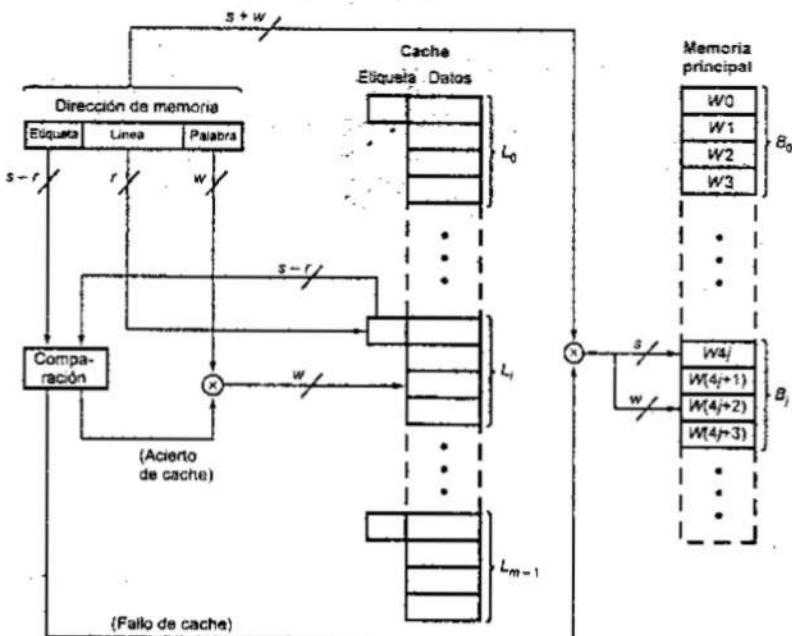


Figura 4.17. Organización de cache con correspondencia directa [HWAN93].

donde:

i = número de línea de cache

j = número de bloque de memoria principal

m = número de líneas en la cache

La función de correspondencia se implementa fácilmente utilizando la dirección. Desde el punto de vista del acceso a cache, cada dirección de memoria principal puede verse como dividida en tres campos. Los w bits menos significativos identifican cada palabra dentro de un bloqué de memoria principal; en la mayoría de las máquinas actuales, el direccionamiento es a nivel de bytes. Los s bits restantes especifican uno de los 2^s bloques de la memoria principal. La lógica de la cache interpreta estos s bits como una etiqueta de $s - r$ bits (parte más significativa) y un campo de línea de r bits. Este último campo identifica una de las $m = 2^r$ líneas de la cache. El efecto es que se hacen corresponder bloques de memoria principal a líneas de cache de la siguiente manera:

Línea de cache	Bloques de memoria principal asignados
0	0, m , $2m$, ..., $2^s - m$
1	1, $m + 1$, $2m + 1$, ..., $2^s - m + 1$
⋮	⋮
$m - 1$	$m - 1$, $2m - 1$, $3m - 1$, ..., $2^s - 1$

Por tanto, el uso de una parte de la dirección como número de línea proporciona una correspondencia o asignación única de cada bloque de memoria principal en la cache. Cuando un bloque es realmente escrito en la línea que tiene asignada, es necesario etiquetarlo para distinguirlo del resto de los bloques que pueden introducirse en dicha línea. Para ello se emplean los $s - r$ bits más significativos.

La Figura 4.18 muestra nuestro ejemplo de sistema utilizando correspondencia directa². En el ejemplo: $m = 16K = 2^{14}$, mientras que $i = j$ módulo 2^{14} . La asignación sería:

Línea de cache	Dirección de memoria de inicio de bloque
0	000000, 010000, ..., FF0000
1	000004, 010004, ..., FF0004
⋮	⋮
$m - 1$	00FFFC, 01FFFC, ..., FFFF0C

Obsérvese que no hay dos bloques que se asignen en la misma línea, que tengan el mismo número de etiqueta. Así, los bloques 000000, 010000, ..., FF0000 tienen respectivamente los números de etiqueta 00, 01, ..., FF.

² Por conveniencia, las direcciones y valores de memoria de la figura se dan en notación hexadecimal. El lector no familiarizado con esta notación puede ver el apéndice del Capítulo 8.

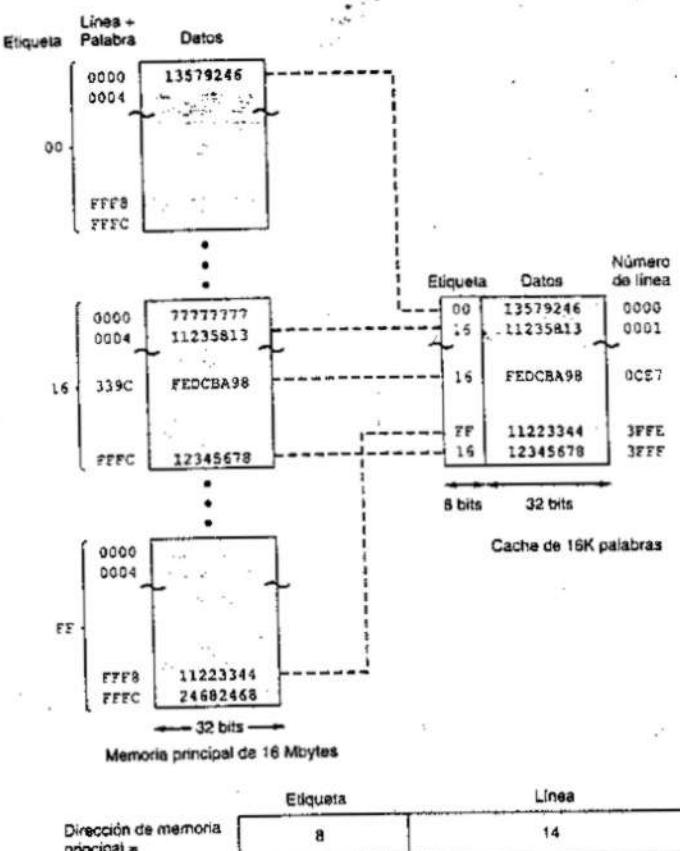


Figura 4.18. Ejemplo de correspondencia directa.

Refiriéndonos de nuevo a la Figura 4.15, una operación de lectura se lleva a cabo de la siguiente manera: Al sistema de cache se presenta una dirección de 24 bits. El número de línea, de 14 bits, se utiliza como índice para acceder a una línea particular dentro de la cache. Si el número de etiqueta, de 8 bits, coincide con el número de etiqueta almacenado actualmente en esa línea, el número de palabra de 2 bits se utiliza para seleccionar uno de los cuatro bytes de esa línea. Si no, el campo de 22 bits de etiqueta-línea se emplea para captar un bloque de memoria principal. La dirección real que se utiliza para la captación consta de los mencionados 22 bits concatenados con dos bits 0, y se captan 4 bytes a partir del comienzo del bloque.

La técnica de correspondencia directa es simple y poco costosa de implementar. Su principal desventaja es que hay una posición concreta de cache para cada bloque dado. Por ello, si un programa referencia repetidas veces a palabras de dos bloques diferentes asignados en

la misma línea, dichos bloques se estarían intercambiando continuamente en la cache, y la tasa de aciertos sería baja.

La correspondencia asociativa supera la desventaja de la directa, permitiendo que cada bloque de memoria principal pueda cargarse en cualquier línea de la cache. En este caso, la lógica de control de la cache interpreta una dirección de memoria simplemente como una etiqueta y un campo de palabra. El campo de etiqueta identifica únicamente un bloque de memoria principal. Para determinar si un bloque está en la cache, su lógica de control debe examinar simultáneamente todas las etiquetas de líneas para buscar una coincidencia. La Figura 4.19 muestra esta lógica.

La Figura 4.20 muestra nuestro ejemplo, utilizando correspondencia asociativa. Una dirección de memoria principal consta de una etiqueta de 22 bits y un número de byte de dos bits. La etiqueta de 22 bits debe almacenarse con el bloque de 32 bits de datos en cada línea de la cache. Obsérvese que son los 22 bits de la izquierda de la dirección (los más significativos) los que forman la etiqueta. De manera que, la dirección de 24 bits 16339C en hexadecimal, contiene la etiqueta de 22 bits 058CE7. Esto se ve fácilmente en notación binaria:

dirección de memoria:	0001 0110 0011 0011 1001 1100 1 6 3 3 9 C etiqueta (22 bits de la izda.)	(binario)	00 0101 1000 1100 1110 0111 0 5 8 C E 7 etiqueta (22 bits de la izda.)	(hexadecimal)

Con la correspondencia asociativa hay flexibilidad para que cualquier bloque sea reemplazado cuando se va a escribir uno nuevo en la cache. Los algoritmos de reemplazo o sustitución, discutidos más adelante en esta sección, se diseñan para maximizar la tasa de aciertos.

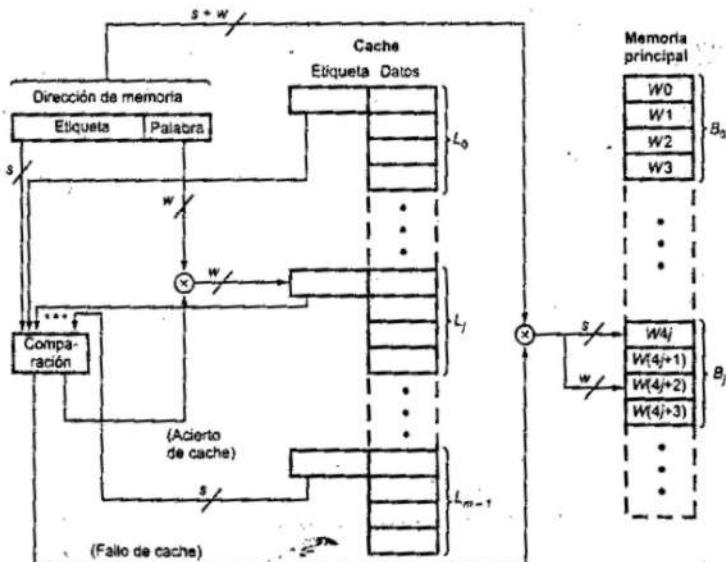


Figura 4.19. Organización de cache totalmente asociativa [HWAN93].

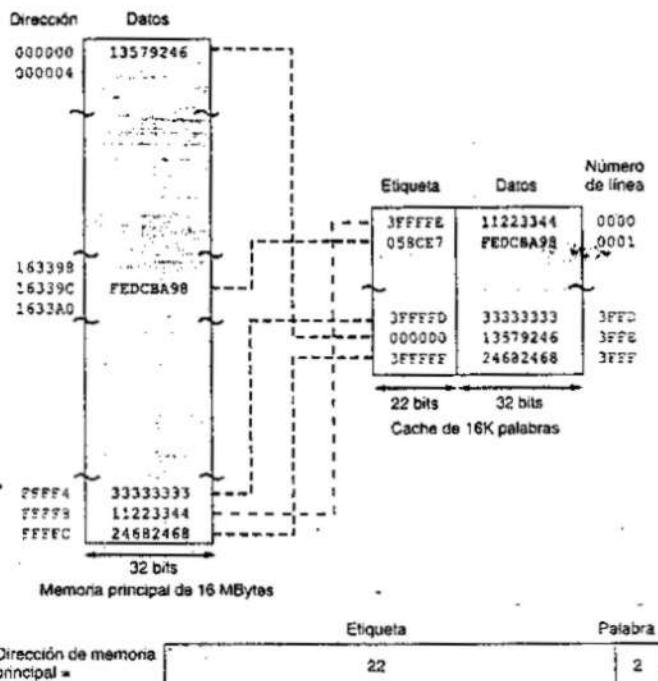


Figura 4.20. Ejemplo de correspondencia asociativa.

La principal desventaja de la correspondencia asociativa es la compleja circuitería necesaria para examinar en paralelo las etiquetas de todas las líneas de cache.

La correspondencia asociativa por conjuntos es una solución de compromiso que recoge lo positivo de las correspondencias directa y asociativa, sin presentar sus desventajas. En este caso, la cache se divide en v conjuntos, cada uno de k líneas. Las relaciones que se tienen son:

$$m = v \times k$$

$$i = j \text{ módulo } v$$

donde:

i = número de conjunto de cache.

j = número de bloque de memoria principal.

m = número de líneas de la cache.

En este caso, se denomina correspondencia asociativa por conjuntos de k vías. Con la asignación asociativa por conjuntos, el bloque B_j puede asignarse en cualquiera de las k líneas del conjunto i . En este caso, la lógica de control de la cache interpreta una dirección de memoria como tres campos: etiqueta, conjunto y palabra. Los d bits de conjunto especifican uno de entre $v = 2^d$ conjuntos. Los s bits de los campos de etiqueta y de conjunto especifican uno de los 2^s bloques de memoria principal. La Figura 4.21 ilustra la lógica de control de la

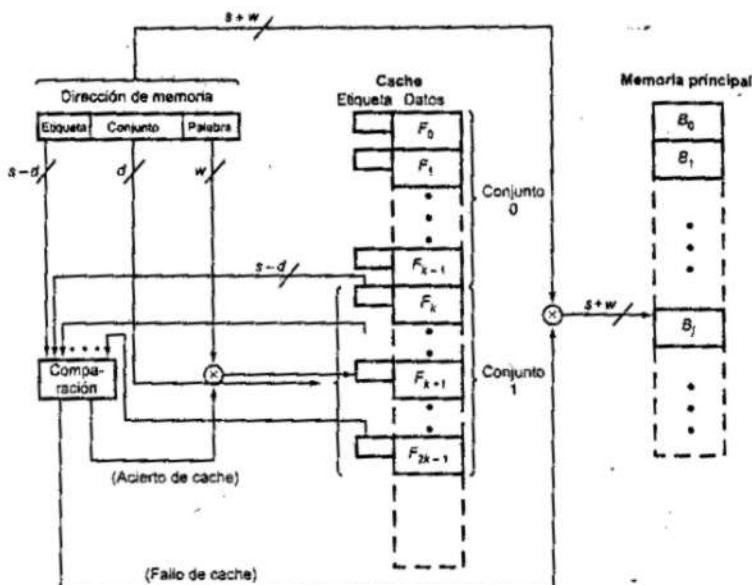


Figura 4.21. Estructura de cache asociativa por conjuntos [HWAN93].

cache. Con la correspondencia totalmente asociativa, la etiqueta en una dirección de memoria es bastante larga, y debe compararse con la etiqueta de cada línea en la cache. Con la correspondencia asociativa por conjuntos de k vías, la etiqueta de una dirección de memoria es mucho más corta, y se compara sólo con las k etiquetas dentro de un mismo conjunto.

La Figura 4.22 muestra nuestro ejemplo, utilizando correspondencia asociativa por conjuntos con dos líneas por cada conjunto, denominada «asociativa por conjuntos de dos vías». El número de conjunto, de 13 bits, identifica un único conjunto de dos líneas dentro de la cache. También da el número, módulo 2^{13} , del bloque de memoria principal. Esto determina la asignación de bloques en líneas. Así, los bloques de memoria principal 000000, 00A000, ..., FF1000, se hacen corresponder al conjunto 0 de la cache. Cualquier de dichos bloques puede cargarse en alguna de las dos líneas del conjunto. Obsérvese que no hay dos bloques que se hagan corresponder al mismo conjunto de la cache, que tengan el mismo número de etiqueta. Para una operación de lectura, el número de conjunto, de 13 bits, se utiliza para determinar qué conjunto de dos líneas va a examinarse. Ambas líneas del conjunto se examinan buscando una coincidencia con el número de etiqueta de la dirección a la que se va a acceder.

En el caso extremo de $v = m$, $k = 1$, la técnica asociativa por conjuntos se reduce a la correspondencia directa, y para $v = 1$, $k = m$, se reduce a la totalmente asociativa. El uso de dos líneas por conjunto ($v = m/2$, $k = 2$) es el caso más común, mejorando significativamente la tasa de aciertos respecto de la correspondencia directa. La asociativa por conjuntos de cuatro vías ($v = m/4$, $k = 4$) produce una modesta mejora adicional con un coste añadido relativamente pequeño [MAYB84, HILL89]. Un incremento adicional, en el número de líneas por conjunto tiene poco efecto.

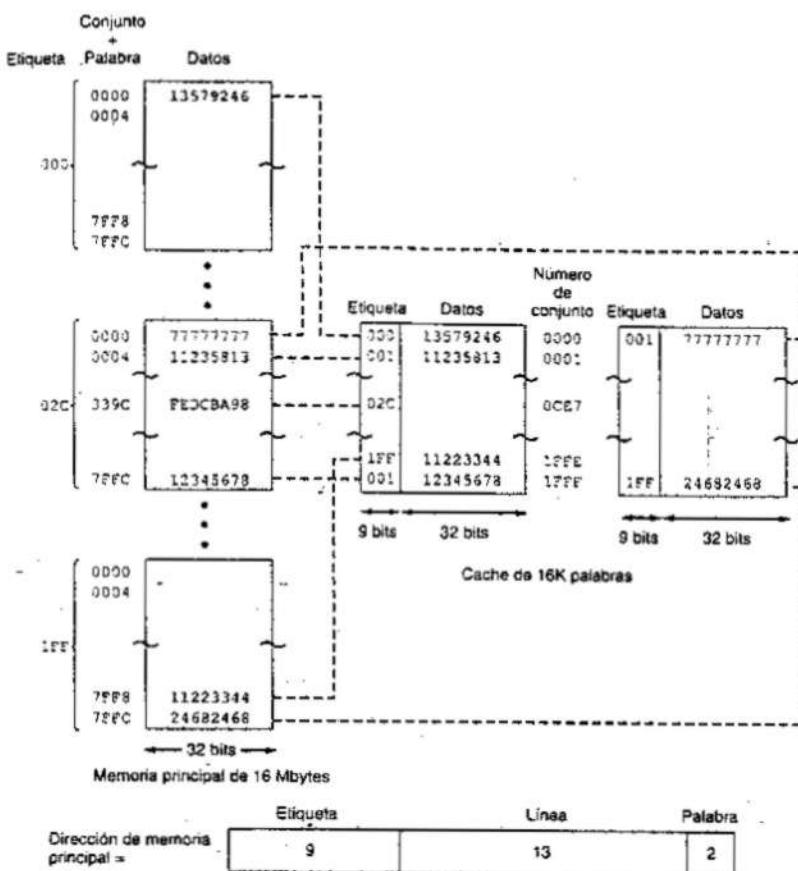


Figura 4.22. Correspondencia asociativa por conjuntos de dos vías.

Algoritmos de sustitución

Cuando se introduce un nuevo bloque en la cache, debe sustituirse uno de los bloques existentes. Para el caso de correspondencia directa, sólo hay una posible línea para cada bloque particular, y no hay elección posible. Para las técnicas asociativas, se requieren algoritmos de sustitución. Para conseguir alta velocidad, tales algoritmos deben implementarse en hardware. Se han probado diversos algoritmos; mencionaremos cuatro de los más comunes. El más efectivo es probablemente el denominado «utilizado menos recientemente» (LRU, least-recently used): se sustituye el bloque que se ha mantenido en la cache por más tiempo sin haber sido referenciado. Esto es fácil de implementar para la asociativa por conjuntos de dos vías. Cada línea incluye un bit USO. Cuando una línea es referenciada, se pone a 1 su bit USO y a 0 el de la otra línea del mismo conjunto. Cuando va a transferirse un bloque al conjunto, se utiliza la línea cuyo bit USO es 0. Ya que estamos suponiendo que son más pro-

bables de referenciar las posiciones de memoria utilizadas más recientemente, el LRU debiera dar la mejor tasa de aciertos. Otra posibilidad es el «primero en entrar-primero en salir» (FIFO, «first-in-first-out»); se sustituye aquel bloque del conjunto que ha estado más tiempo en la cache. El algoritmo FIFO puede implementarse fácilmente mediante una técnica cíclica («round-robin») o buffer circular. Otra posibilidad más es la del «utilizado menos frecuentemente» (LFU, «least-frequently used»); se sustituye aquel bloque del conjunto que ha experimentado menos referencias. LFU podría implementarse asociando un contador a cada línea. Una técnica no basada en el grado de utilización consiste simplemente en coger una línea al azar (aleatoria) entre las posibles candidatas. Estudios realizados mediante simulación han mostrado que la sustitución aleatoria proporciona unas prestaciones sólo ligeramente inferiores a un algoritmo basado en la utilización [SMIT82].

Política de escritura

Antes de que pueda ser reemplazado un bloque que está en una línea de cache, es necesario comprobar si ha sido alterado en cache pero no en memoria principal. Si no lo ha sido, puede de escribirse sobre la línea de cache. Si ha sido modificado, esto significa que se ha realizado al menos una operación de escritura sobre una palabra de la línea correspondiente de la cache, y la memoria principal debe actualizarse de acuerdo con ello. Son posibles varias políticas de escritura, con distintos compromisos entre prestaciones y coste económico. Hay dos problemas contra los que luchar. En primer lugar, más de un dispositivo puede tener acceso a la memoria principal. Por ejemplo, un módulo de E/S puede escribir/leer directamente en/de memoria. Si una palabra ha sido modificada sólo en la cache, la correspondiente palabra de memoria no es válida. Además, si el dispositivo de E/S ha alterado la memoria principal, entonces la palabra de cache no es válida. Un problema más complejo ocurre cuando varios procesadores se conectan al mismo bus y cada uno de ellos tiene su propia cache local. En tal caso, si se modifica una palabra en una de las caches, podría presumiblemente invalidar una palabra de otras caches.

La técnica más sencilla se denomina *escritura inmediata*. Utilizando esta técnica, todas las operaciones de escritura se hacen, tanto en cache como en memoria principal, asegurando que el contenido de la memoria principal siempre es válido. Cualquier otro módulo procesador-cache puede monitorizar el tráfico a memoria principal para mantener la coherencia en su propia cache. La principal desventaja de esta técnica es que genera un tráfico sustancial a memoria que puede originar un cuello de botella. Una técnica alternativa, conocida como *post-escritura*, minimiza las escrituras en memoria. Con la post-escritura, las actualizaciones se hacen sólo en la cache. Cuando tiene lugar una actualización, se activa un bit ACTUALIZAR asociado a la línea. Después, cuando el bloque es sustituido, es (post-) escrito en memoria principal si, y solo si, el bit ACTUALIZAR está activo. El problema de este esquema es que a veces porciones de memoria principal no son válidas, y los accesos por parte de los módulos de E/S sólo podrían hacerse a través de la cache. Esto complica la circuitería y genera un cuello de botella potencial. La experiencia ha demostrado que el porcentaje de referencias a memoria para escritura es del orden del 15 % [SMIT82].

En una estructura de bus en la que más de un dispositivo (normalmente un procesador) tiene una cache, y la memoria principal es compartida, se tropieza con un nuevo problema. Si se modifican los datos de una cache, se invalida, no solamente la palabra correspondiente de memoria principal, sino también la misma palabra en otras caches (si coincide que otras caches tengan la misma palabra). Incluso, si se utiliza una política de escritura inmediata, las otras caches pueden contener datos no válidos. Un sistema que evite este problema, se dice que mantiene la coherencia de cache. Entre las posibles aproximaciones a la coherencia de cache se incluyen:

- **Vigilancia del bus con escritura inmediata:** Cada controlador de cache monitoriza las líneas de direcciones, para detectar operaciones de escritura en memoria por parte de otros maestros del bus. Si otro maestro escribe en una posición de memoria compartida, que también reside en la memoria cache, el controlador de cache invalida el elemento de la cache. Esta estrategia depende del uso de una política de escritura inmediata por parte de todos los controladores de cache.
- **Transparencia hardware:** Se utiliza hardware adicional para asegurar que todas las actualizaciones de memoria principal, vía cache, quedan reflejadas en todas las caches. Así, si un procesador modifica una palabra de su cache, esta actualización se escribe en memoria principal. Además, de manera similar, se actualizan todas las palabras coincidentes de otras caches.
- **Memoria excluida de cache:** Sólo una porción de memoria principal se comparte por más de un procesador, y ésta se diseña como no transferible a cache. En un sistema de este tipo, todos los accesos a la memoria compartida son fallos de cache, porque la memoria compartida nunca se copia en la cache. La memoria excluida de cache puede ser identificada utilizando lógica de selección de chip, o los bits más significativos de la dirección.

La coherencia de cache es un campo activo de investigación, y será tratado con más detalle en el Capítulo 16.

Tamaño de línea

Otro elemento de diseño es el tamaño de línea. Cuando se recupera y ubica en cache un bloque de datos, se recuperan, no sólo la palabra deseada, sino además algunas palabras adyacentes. A medida que aumenta el tamaño de bloque, la tasa de aciertos primero aumenta debido al principio de localidad, el cual establece que es probable que los datos en la vecindad de una palabra referenciada sean referenciados en un futuro próximo. Al aumentar el tamaño de bloque, más datos útiles son llevados a la cache. Sin embargo, la tasa de aciertos comenzará a decrecer cuando el tamaño de bloque se haga aún mayor y la probabilidad de utilizar la nueva información captada se haga menor que la de reutilizar la información que tiene que reemplazarse. Dos efectos concretos entran en juego:

- Bloques más grandes reducen el número de bloques que caben en la cache. Dado que cada bloque captado se escribe sobre contenidos anteriores de la cache, un número reducido de bloques da lugar a que se sobrescriba sobre datos poco después de haber sido captados.
- A medida que un bloque se hace más grande, cada palabra adicional está más lejos de la requerida, y por tanto es más improbable que sea necesaria a corto plazo.

La relación entre tamaño de bloque y tasa de aciertos es compleja, dependiendo de las características de localidad de cada programa particular, no habiéndose encontrado un valor óptimo definitivo. Un tamaño entre 4 y 8 unidades direccionables (palabras o bytes) parece estar razonablemente próximo al óptimo [SMIT87a, PRZY88, PRZY90, HAND98].

Número de caches

Cuando se introdujeron originalmente las caches, un sistema tenía normalmente sólo una cache. Más recientemente, se ha convertido en una norma el uso de múltiples caches. Hay dos aspectos de diseño relacionados con este tema, que son: el número de niveles de cache y el uso de cache unificada frente a cache partida.

Con el aumento de densidad de integración, ha sido posible tener una cache en el mismo chip del procesador: cache «on-chip». Comparada con la accesible a través de un bus externo, la cache on-chip reduce la actividad del bus externo del procesador y, por tanto, reduce los tiempos de ejecución e incrementa las prestaciones globales del sistema. Cuando la instrucción o dato requeridos se encuentran en la cache on-chip, se elimina el acceso al bus. Debido a que los caminos de datos internos al procesador son muy cortos en comparación con la longitud de los buses, los accesos a la cache on-chip se efectúan apreciablemente más rápidos que los ciclos de bus, incluso en ausencia de estados de espera. Además, durante este período el bus está libre para realizar otras transferencias.

La inclusión de una cache on-chip deja abierta la cuestión de si es además deseable una cache externa u «off-chip». Normalmente la respuesta es afirmativa, y los diseños más actuales incluyen tanto cache on-chip como externa. La estructura resultante se conoce como cache de dos niveles, siendo la cache interna el nivel 1 (L1) y la externa el nivel 2 (L2). La razón por la que se incluye una cache L2 es la siguiente. Si no hay cache L2 y el procesador hace una petición de acceso a una posición de memoria que no está en la cache L1, entonces el procesador debe acceder a la DRAM o la ROM a través del bus. Debido a la lentitud usual del bus y a los tiempos de acceso de las memorias, se obtienen bajas prestaciones. Por otra parte, si se utiliza una cache L2 SRAM, entonces, con frecuencia, la información que falta puede recuperarse fácilmente. Si la SRAM es suficientemente rápida para adecuarse a la velocidad del bus, los datos pueden accederse con cero estados de espera, el tipo de transferencia de bus más rápido.

La mejora potencial del uso de una cache L2 depende de las tasas de aciertos en ambas caches L1 y L2. Varios estudios han demostrado que, en general, el uso de un segundo nivel de cache mejora las prestaciones (véase, por ejemplo, [AZIM92], [NOVI93], [HAND98]).

Cuando hicieron su aparición las caches on-chip, muchos de los diseños contenían una sola cache para almacenar las referencias, tanto a datos como a instrucciones. Más recientemente, se ha hecho normal separar la cache en dos: una dedicada a instrucciones y otra a datos.

Una cache unificada tiene varias ventajas potenciales:

- Para un tamaño dado de cache, una unificada tiene una tasa de aciertos mayor que una partida, ya que nivela automáticamente la carga entre captación de instrucciones y de datos. Es decir, si un patrón de ejecución implica muchas más captaciones de instrucciones que de datos, la cache tenderá a llenarse con instrucciones, y si el patrón de ejecución involucra relativamente más captaciones de datos, ocurrirá lo contrario.
- Sólo se necesita diseñar e implementar una cache.

A pesar de estas ventajas, la tendencia es hacia caches partidas, particularmente para máquinas superescalares, tales como el Pentium II y el PowerPC, en las que se enfatiza la ejecución paralela de instrucciones y la pre-captación de instrucciones futuras previstas. La ventaja clave del diseño de cache partida es que elimina la competición por la cache entre el procesador de instrucciones y la unidad de ejecución. Esto es importante en diseños que cuentan con segmentación de cauce («pipelining») de instrucciones. Normalmente, el procesador captará instrucciones anticipadamente y llenará un buffer con las instrucciones que van a ejecutarse. Supongamos ahora que se tiene una cache unificada de instrucciones/datos. Cuando la unidad de ejecución realiza un acceso a memoria para cargar y almacenar datos, se envía la petición a la cache unificada. Si, al mismo tiempo, el pre-captador de instrucciones emite una petición de lectura de una instrucción a la cache, dicha petición será temporalmente bloqueada para que la cache pueda servir primero a la unidad de ejecución, permitiéndole completar la ejecución de la instrucción en curso. Esta disputa por la cache puede degradar las prestaciones, interfiriendo con el uso eficiente del cauce segmentado de instrucciones. La cache partida supera esta dificultad.

4.4. ORGANIZACIÓN DE LA CACHE EN EL PENTIUM II Y EL PowerPC

ORGANIZACIÓN DE CACHE EN EL PENTIUM II

La evolución de la estructura de la cache se observa claramente en la evolución de los microprocesadores de Intel. El 80386 no tiene cache on-chip. El 80486 incluye una sola cache on-chip de 8 KBytes, utilizando un tamaño de línea de 16 bytes y una organización asociativa por conjuntos de cuatro vías. El Pentium incluye dos caches on-chip, una para datos y otra para instrucciones. Cada cache es de 8 KBytes, utilizando un tamaño de línea de 32 bytes y una organización asociativa por conjuntos de dos vías. El Pentium Pro y el Pentium II incluyen también dos caches L1 on-chip. Las primeras versiones del procesador incluyen una cache de instrucciones de 8 KBytes, asociativa por conjuntos de cuatro vías, y una cache de datos, también de 8 KBytes, asociativa por conjuntos de dos vías. El Pentium Pro y el Pentium II incluyen además una cache L2 que alimenta a las dos caches L1. La cache L2 es asociativa por conjuntos de cuatro vías, y con tamaños que oscilan entre 256 Kbytes y 1 Mbyte.

La Figura 4.23 proporciona una visión simplificada de la estructura del Pentium II, resaltando la ubicación de las tres caches. El núcleo del procesador consta de cuatro componentes principales:

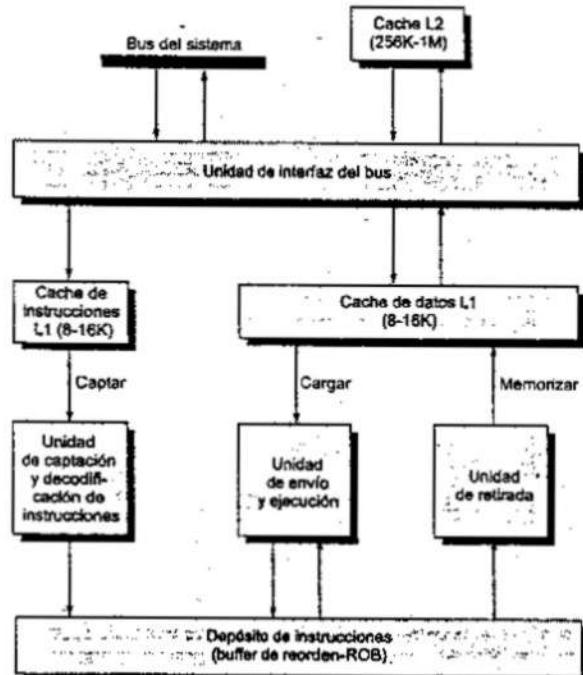


Figura 4.23. Diagrama de bloques del Pentium II.

- **Unidad de captación/decodificación:** Capta instrucciones en su orden de la cache de instrucciones L1, las decodifica en una serie de microoperaciones, y memoriza los resultados en el depósito («pool») de instrucciones.
- **Depósito de instrucciones:** Contiene el conjunto de instrucciones actualmente disponible para ejecución.
- **Unidad de envío/ejecución:** Planifica la ejecución de las micro-operaciones sujetas a dependencias de datos y disponibilidad de recursos, de manera que las micro-operaciones pueden planificarse para su ejecución en un orden distinto al que fueron captadas. Cuando hay tiempo disponible, esta unidad realiza una ejecución especulativa de micro-operaciones que pueden ser necesarias en el futuro. La unidad ejecuta micro-operaciones, captando los datos necesarios de la cache de datos L1, y almacenando los resultados temporalmente en registros.
- **Unidad de retirada:** Determina cuándo los resultados provisionales, especulativos, deben retirarse (unificarse) para establecerse como permanentes en registros o en la cache de datos L1. Esta unidad también elimina instrucciones del depósito tras haber unificado los resultados.

La Figura 4.24 muestra los elementos clave de la cache de datos L1. Los datos en la cache forman 128 conjuntos de dos líneas cada uno. La cache está organizada en dos «vías» de 4 Kbytes. Cada línea tiene asociados una etiqueta y dos bits de estado, con una organización lógica en dos directorios; de manera que hay un elemento de directorio por cada línea de la cache. La etiqueta está formada por los 24 bits más significativos de la dirección de memoria de los datos almacenados en la correspondiente línea. El controlador de cache utiliza un

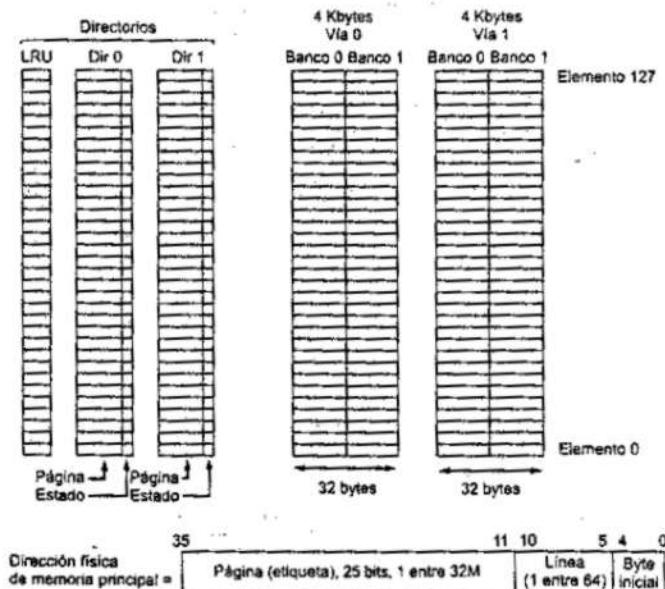


Figura 4.24. Estructura de la cache de datos del Pentium II [ANDE98].

Tabla 4.5. Estados MESI de una línea de cache

	M Modificada	E Exclusiva	S Compartida	I No válida
¿Es válida esta línea de cache?	Sí	Sí	Sí	No
La copia en memoria...	Está desfasada	Es válida	Es válida	—
¿Existen copias en otras caches?	No	No	Puede ser	Puede ser
Una escritura en esta línea...	No va al bus	No va al bus	Va al bus y actualiza la cache	Va directamente al bus

algoritmo de sustitución del menos recientemente utilizado (LRU) y, por tanto, se asocia con cada conjunto de dos líneas un único bit de LRU.

La cache de datos emplea una política de post-escritura: los datos se escriben en memoria principal sólo cuando, habiendo sido actualizados, se eliminan de la cache. El procesador Pentium II puede configurarse dinámicamente para utilizar la política de escritura inmediata.

Coherencia de la cache de datos

Para conseguir coherencia de cache, la cache de datos permite un protocolo conocido como MESI (Modified/Exclusive/Shared/Invalid, «modificada/exclusiva/compartida/no válida»). MESI está diseñado para satisfacer los requisitos de coherencia de cache de un sistema multiprocesador, pero es también útil en una arquitectura Pentium II monoprocesador.

La cache de datos incluye dos bits de estado por cada etiqueta, de manera que cada línea puede estar en uno de estos cuatro estados:

- **Modificada:** la línea de cache ha sido modificada (diferente de memoria principal) y está disponible sólo en esta cache.
- **Exclusiva:** el contenido de la línea de la cache coincide con lo que se tiene en memoria principal, y no está presente en ninguna otra cache.
- **Compartida:** la línea en la cache coincide con memoria principal y puede estar presente en otra cache.
- **No válida:** la línea de la cache no contiene datos válidos.

La Tabla 4.5 resume el significado de los cuatro estados. El protocolo MESI se estudia con detalle en el Capítulo 16.

Tabla 4.6. Modos de funcionamiento de la cache del Pentium II

Bits de control		Modo de operación		
CD	NW	Llenado de cache	Escrituras inmediatas	Invalidez
0	0	Habilitado	Habilitadas	Habilitadas
1	0	Inhabilitado	Habilitadas	Habilitadas
1	1	Inhabilitado	Inhabilitadas	Inhabilitadas

Tabla 4.7. Caches internas en la familia PowerPC

Modelo	Tamaño	Bytes/línea	Organización
PowerPC 601	1 32-KByte	32	Asociativa por conjuntos de 8 vías
PowerPC 603	2 8-KByte	32	Asociativa por conjuntos de 2 vías
PowerPC 604	2 16-KByte	32	Asociativa por conjuntos de 4 vías
PowerPC 620	2 32-KByte	64	Asociativa por conjuntos de 8 vías

Control de cache

La cache interna se controla por dos bits de uno de los registros de control (véase la Tabla 4.6), rotulados CD (cache disable: «inhabilitar cache») y NW (not write through: «no escritura inmediata»). Hay también dos instrucciones del Pentium II que pueden utilizarse para controlar la cache: INVD invalida la memoria cache interna e indica que se invalide la cache externa (si la hay). WBINVD post-escribe e invalida la cache interna, y entonces post-escribe e invalida la externa.

ORGANIZACIÓN DE CACHÉ EN EL PowerPC

La organización de cache del PowerPC ha ido evolucionando paralelamente a la arquitectura global de la familia PowerPC, reflejando la búsqueda continua de mejores prestaciones que es el motor de todos los diseñadores de microprocesadores.

La Tabla 4.7 muestra esta evolución. El modelo original, el 601, incluye una sola cache de código/datos de 32 Kbytes que es asociativa por conjuntos de ocho vías. El 603 emplea un

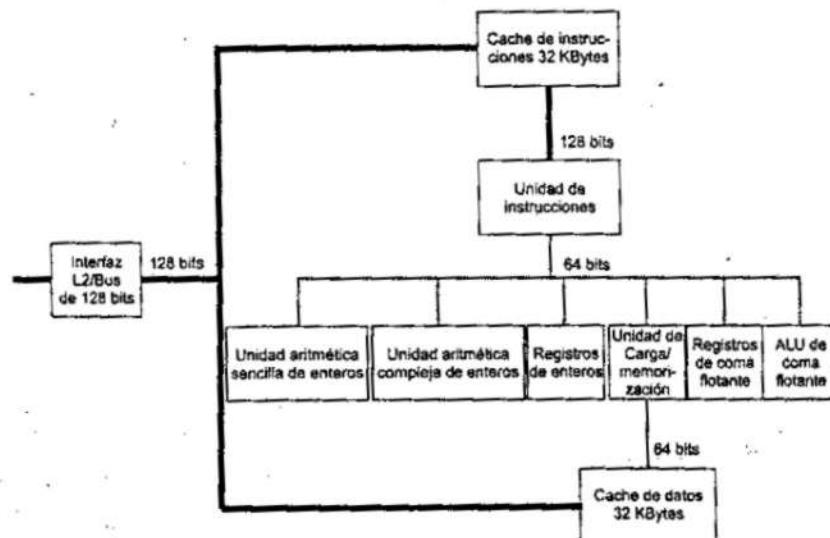


Figura 4.25. Diagrama de bloques del PowerPC G3.

diseño RISC más sofisticado, pero tiene una cache más pequeña: 16 KBytes, divididos en caches separadas de datos y de instrucciones, ambas con una organización asociativa por conjuntos de dos vías. El resultado es que el 603 tiene aproximadamente las mismas prestaciones que el 601, pero con menor coste. En cada modelo posterior, el 604 y el 620, se va duplicando el tamaño de las caches respecto de su predecesor. El modelo actual, G3, tiene el mismo tamaño de caches L1 que el 620.

La Figura 4.25 es un esquema simplificado de la estructura del PowerPC G3, resaltando la ubicación de las dos caches. Los otros miembros de la familia tienen una organización similar. Las unidades de ejecución fundamentales son dos unidades aritméticas y lógicas de enteros, que pueden ejecutar en paralelo, y una unidad de coma flotante con sus propios registros y componentes de multiplicación, suma y división. La cache de datos alimenta, tanto las operaciones con enteros como las de coma flotante, mediante una unidad de carga/memorización. La cache de instrucciones, que es de sólo lectura, alimenta a una unidad de instrucciones, cuyo funcionamiento se discute en el Capítulo 13.

Las caches L1 internas son asociativas por conjuntos de ocho vías, y utilizan una variante del protocolo MESI de coherencia de cache. La cache L2 es asociativa por conjuntos de dos vías, con tamaños de 256 K, 512 K o 1 Mbyte.

ORGANIZACIÓN AVANZADA DE MEMORIAS DRAM

Como se discutió en el Capítulo 2, uno de los cuellos de botella más críticos de un sistema que utiliza procesadores de altas prestaciones es la interfaz con la memoria principal interna. Esta interfaz es el camino más importante en el computador. El bloque básico de construcción de la memoria principal es el chip de DRAM, como lo ha sido durante más de 20 años, y desde principios de la década de los 70 hasta hace poco no ha habido cambios significativos en la arquitectura DRAM. El chip DRAM tradicional está limitado, tanto por su arquitectura interna como por su interfaz con el bus de memoria del procesador.

Hemos visto que una forma de abordar el problema de las prestaciones de la memoria principal DRAM ha sido insertar uno o más niveles de cache SRAM de alta velocidad entre la DRAM y el procesador. Pero la SRAM es mucho más costosa que la DRAM, y ampliar el tamaño de cache más allá de cierta cantidad produce menos beneficios.

En los últimos años, se han explorado diversas versiones mejoradas de la arquitectura básica DRAM, y algunas de ellas están ya comercializadas. No está claro por ahora si alguna de ellas emergerá como estándar de DRAM o si sobrevivirán varias. Esta sección proporciona una visión de estas nuevas tecnologías de DRAM.

DRAM MEJORADA

Tal vez la más simple entre las nuevas arquitecturas de DRAM es la DRAM mejorada (EDRAM, Enhanced DRAM), desarrollada por Ramtron [BOND94]. La EDRAM integra una pequeña cache SRAM en un chip de DRAM genérica.

La Figura 4.26 ilustra una versión de EDRAM de 4 Mbits. La cache SRAM almacena el contenido completo de la última fila leída, que consta de 2.048 bits, o 512 grupos de 4 bits. Un comparador almacena el valor de 11 bits de selección más reciente de dirección de fila. Si el siguiente acceso se realiza a la misma fila, sólo se necesita hacerlo en la rápida cache SRAM.

La EDRAM incluye otras características diversas que mejoran sus prestaciones. Las operaciones de refresco pueden llevarse a cabo en paralelo con operaciones de lectura de la

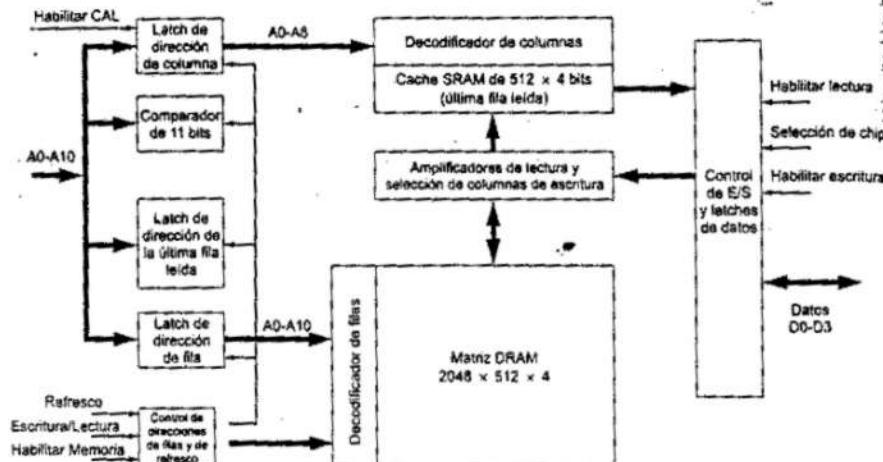


Figura 4.26. RAM dinámica mejorada (EDRAM).

cache, minimizando el tiempo que el chip estaría no disponible por causa del refresco. Obsérvese también que el camino de lectura desde la cache de filas hasta el puerto de salida es independiente del camino de escritura desde el módulo de E/S a los amplificadores de lectura. Esto posibilita que se pueda efectuar una operación de escritura en paralelo con un acceso de lectura a la cache.

Estudios realizados por Ramtron indican que la EDRAM rinde igual o más que una DRAM ordinaria junto con otra cache SRAM externa (a la DRAM) pero más grande.

DRAM CACHE

La DRAM 'cache' (CDRAM), desarrollada por Mitsubishi [HIDA90] es similar a la EDRAM. La CDRAM incluye una cache SRAM más grande que la de una EDRAM (16 en lugar de 2 Kbytes).

La SRAM de la CDRAM puede utilizarse de dos modos. En primer lugar, puede utilizarse como una verdadera cache de un número dado de líneas de 64 bits. En esto difiere de la EDRAM, en la que la cache SRAM contiene sólo un bloque: la fila accedida más recientemente. El modo de cache de la CDRAM es efectivo para accesos aleatorios a memoria usuales.

La SRAM de la CDRAM puede utilizarse también como buffer para proporcionar accesos serie a un bloque de datos. Por ejemplo, para refrescar una pantalla gráfica («bit-mapped»), la CDRAM puede precapturar los datos de la DRAM en el buffer SRAM. Accesos subsiguientes al chip se harán sólo a la SRAM.

DRAM SÍNCRONA

Una aproximación bastante diferente de DRAM con prestaciones mejoradas es la DRAM síncrona (SDRAM), desarrollada conjuntamente por varias compañías [VOGL94].

A diferencia de las DRAM típicas, que son asíncronas, la SDRAM intercambia datos con el procesador de forma sincronizada con una señal de reloj externa, funcionando a la velocidad tope del bus procesador/memoria, sin imponer estados de espera.

En una DRAM típica, el procesador presenta las direcciones y niveles de control a la memoria, indicando que los datos de una posición de memoria concreta deben bien escribirse o leerse. Después de un tiempo, el tiempo de acceso, se escriben o leen los datos en la DRAM. Durante el tiempo de acceso, la DRAM realiza varias operaciones internas, tales como activar las elevadas capacidades de las líneas de fila y de columna, detectar los datos y sacarlos a través de los buffers de salida. El procesador debe simplemente esperar durante este tiempo, haciendo que el sistema baje en prestaciones.

Con el acceso sincrónico, la DRAM introduce y saca datos bajo el control del reloj del sistema. El procesador, u otro maestro, cursa la información de dirección e instrucción, que es retenida por la DRAM. La DRAM responderá después de un cierto número de ciclos de reloj. Entré tanto, el maestro puede realizar sin riesgo otras tareas mientras la SDRAM está procesando la petición.

La Figura 4.27 muestra la lógica interna de una SDRAM. La SDRAM emplea un modo de ráfagas para eliminar los tiempos de establecimiento de dirección y de precarga de las líneas de fila y de columna posteriores al primer acceso. En el modo de ráfagas, se puede secuenciar la salida de una serie de bits de datos una vez que se ha accedido al primero de ellos. Este modo es útil cuando todos los bits a acceder están en secuencia y en la misma fila de la matriz de celdas que el accedido en primer lugar.

Además, la SDRAM tiene una arquitectura interna de banco doble (dos submatrices) que facilita el paralelismo en el propio chip.

El registro de modo y la lógica de control asociada constituyen otra característica clave que diferencia las SDRAM de las DRAM convencionales. Proporciona una manera de particularizar la SRAM para ajustarse a las necesidades concretas del sistema. El registro de modo especifica la longitud de la ráfaga, que es el número de unidades individuales de datos que se entregan sincrónicamente al bus. Este registro también permite al programador ajustar la latencia entre la recepción de una petición de lectura y el comienzo de la transferencia de datos.

La SDRAM funciona mejor cuando transfiere bloques largos de datos en serie, tal como en aplicaciones de procesamiento de textos, hoja de cálculo y multimedia.

DRAM RAMBUS

La RDRAM, desarrollada por Rambus [GARR94, CRIS97], aborda una aproximación más revolucionaria para el problema del ancho de banda de memoria. Los chips RDRAM tienen encapsulados verticales, con todos los terminales en un lateral. El chip intercambia datos con el procesador por medio de 28 hilos de menos de 12 centímetros de longitud. El bus puede direccionar hasta 320 chips de RDRAM a razón de 500 Mbps. Esto contrasta con los aproximadamente 33 Mbps de las DRAM convencionales.

El bus especial de las RDRAM entrega direcciones e información de control, utilizando un protocolo asíncrono orientado a bloques. Tras un tiempo de acceso inicial de 480 ns, se consigue la velocidad de datos de 500 Mbps. Lo que hace posible esta velocidad es el bus en sí, que define muy precisamente las impedancias, la temporización y las señales. En lugar de ser controladas por las señales explícitas RAS, CAS, R/W y CE que se utilizan en DRAM convencionales, las RDRAM obtienen las peticiones de memoria a través de un bus de alta velocidad. Cada petición contiene la dirección deseada, el tipo de operación y el número de bytes en dicha operación.

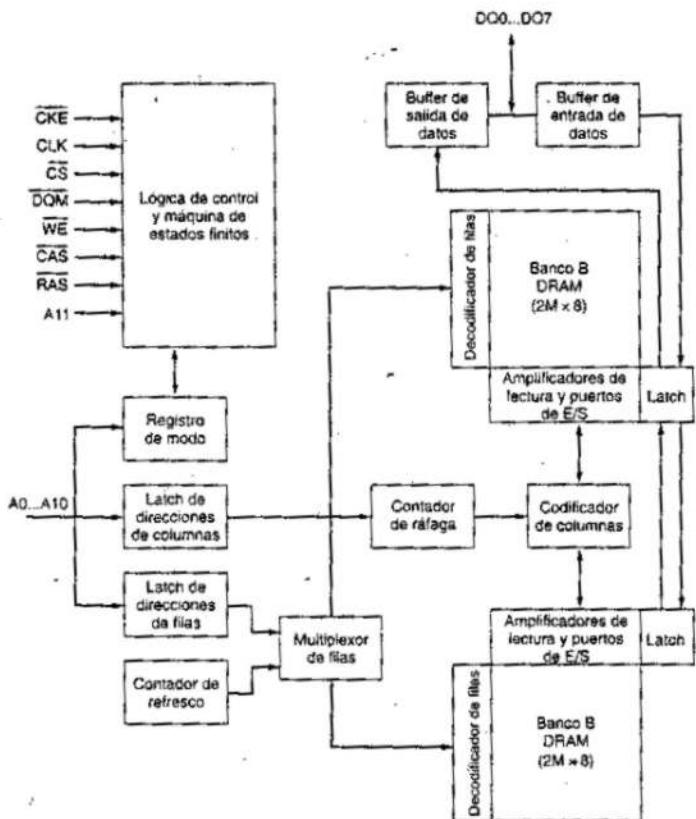


Figura 4.27. RAM dinámica síncrona (SDRAM) (PRZY94).

RAMLINK

El cambio más radical respecto de la DRAM convencional se encuentra en la especificación RamLink [GJES92], desarrollada como parte de una iniciativa de un grupo de trabajo del IEEE denominada Scalable Coherent Interface (SCI). RamLink se centra en la interfaz procesador/memoria, en lugar de en la arquitectura interna de los chips DRAM.

RamLink es una interfaz de memoria con conexiones punto a punto dispuestas en un anillo (Figura 4.28a). El tráfico en el anillo es gestionado por un controlador de memoria, que envía mensajes a los chips de DRAM, los cuales actúan como nodos de la red en anillo. Los datos se intercambian en forma de paquetes (Figura 4.28b).

Los paquetes de petición inicián transacciones u operaciones de memoria. Son enviados por el controlador, y contienen una cabecera de orden, la dirección, una suma de comprobación (check-sum) y, en el caso de órdenes de escritura, los datos a escribir. La cabecera de

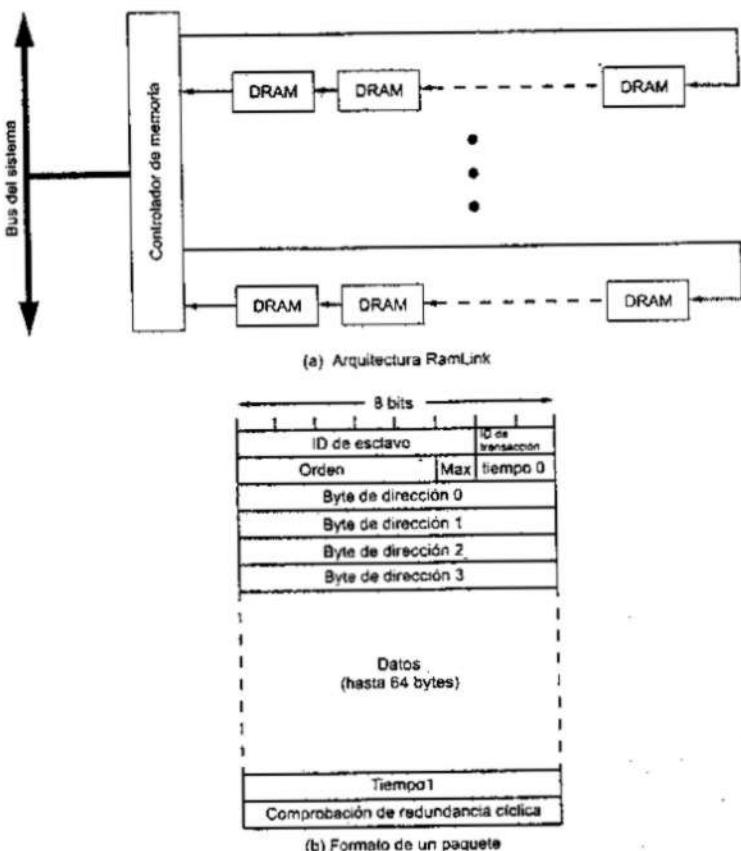


Figura 4.28. RamLink.

orden consta de tipo, tamaño e información de control, y contiene, bien un tiempo de respuesta específico, o bien el tiempo máximo permitido para que el esclavo responda. La información de control incluye un bit que indica si las peticiones subsiguientes serán a posiciones secuenciales. Puede haber hasta cuatro transacciones activas por dispositivo simultáneamente, por lo que cada paquete tiene un identificador (ID) de transacción para casar sin ambigüedad los paquetes de petición y de respuesta.

En una lectura con éxito, la DRAM esclava envía un paquete de respuesta que contiene los datos leídos. En una petición sin éxito, el esclavo cursa un paquete de reintento que indica cuánto tiempo adicional necesita para completar la transacción.

Una de las ventajas de la aproximación RamLink es que proporciona una arquitectura expandible que soporta un número pequeño o grande de DRAM, y no impone una estructura interna de DRAM. La disposición en anillo RamLink está diseñada para coordinar la actividad de muchas DRAM, y dota de una interfaz eficiente al controlador de memoria.

Una mejora introducida más recientemente en la RamLink hace uso de tecnología desarrollada en SDRAM, y es conocida con el nombre de SDRAM [GILL97].

4.6. LECTURAS Y SITIOS WEB RECOMENDADOS

[PRIN91] proporciona un tratamiento amplio de las tecnologías de memorias semiconductoras, incluyendo SRAM, DRAM, y memorias flash. [SHAR97] cubre también los mismos temas, haciendo más hincapié en aspectos relativos a test y seguridad. [PRIN96] se centra en arquitecturas avanzadas de DRAM y SRAM.

[MCEL85] contiene una buena explicación de los códigos de corrección de errores. Para un estudio más profundo merecen la pena los libros [ADAM91] y [BLAH83]. [SHAR97] contiene una buena revisión de los códigos utilizados en memorias actuales.

Un tratamiento en profundidad del diseño de caches se puede encontrar en [HAND98]. Una descripción detallada de la organización de cache del Pentium II pueden encontrarse en [ANDE98], y de la organización de cache del PowerPC en [MOTO97] y [SHAN95]. Un artículo clásico que todavía merece la pena leer es [SMIT82], que revisa los distintos elementos del diseño de cache y presenta los resultados de un amplio conjunto de análisis. En [AGAR89] se presenta un examen detallado de diversos aspectos de diseño de cache relacionados con multiprogramación y multiprocesamiento. [HIGB90] proporciona un conjunto de fórmulas sencillas, que pueden utilizarse para estimar las prestaciones de una cache en función de varios parámetros.

- ADAM91 Adamek, J. *Foundations of Coding*. New York: Wiley, 1991.
- AGAR89 Agarwal, A. *Analysis of Cache Performance for Operating Systems and Multiprogramming*. Boston: Kluwer Academic Publishers, 1989.
- ANDE98 Anderson, D., y Shanley, T. *Pentium Pro and Pentium II System Architecture*. Reading, MA: Addison-Wesley, 1998.
- BLAH83 Blahut, R. *Theory and Practice of Error Control Codes*. Reading, MA: Addison-Wesley, 1983.
- HAND98 Handy, J. *The Cache Memory Book*. San Diego: Academic Press, 1993.
- HIGB90 Higbie, L. «Quick and Easy Cache Performance Analysis.» *Computer Architecture News*, June, 1990.
- MCEL85 McEliece, R. «The Reliability of Computer Memories.» *Scientific American*, January, 1985.
- MOTO97 Motorola, Inc. *PowerPC Microprocessor Family: The Programming Environments for 32-bits Micropocessors*. Denver, CO, 1997.
- PRIN91 Prince, B. *Semiconductor Memories*. New York: Wiley, 1991.
- PRIN96 Prince, B. *High Performance Memories: New Architecture DRAMs and SRAMs. Evolution and Function*. New York: Wiley, 1996.
- SHAN95 Shanley, T. *PowerPC System Architecture*. Reading, MA: Addison-Wesley, 1995.
- SHAR97 Sharma, A. *Semiconductor Memories: Technology, Testing, and Reliability*. New York: IEEE Press, 1997.
- SMIT82 Smith, A. «Cache Memories.» *ACM Computing Surveys*, September, 1992.



SITIO WEB RECOMENDADO:

- **The RAM Guide:** Una buena revisión sobre tecnología RAM y múltiples enlaces de utilidad.

PROBLEMAS

- 4.1. Sugiera razones por las que las RAM han sido tradicionalmente organizadas en sólo un bit por chip, mientras que las ROM están normalmente organizadas en múltiples bits por chip.
- 4.2. Considere una RAM dinámica a la que deba darse un ciclo de refresco 64 veces por milisegundo. Cada operación de refresco requiere 150 ns; un ciclo de memoria requiere 250 ns. ¿Qué porcentaje del tiempo total de funcionamiento de la memoria debe dedicarse a los refrescos?
- 4.3. Diseñe una memoria de 16 bits con una capacidad total de 8.192 bits, utilizando chips de SRAM de tamaño 64×1 bit. Indique la configuración matricial de los chips en la tarjeta de memoria, mostrando todas las señales de entrada y salida necesarias para asignar esta memoria al espacio de direcciones más bajo. El diseño debe permitir accesos, tanto por bytes como por palabras de 16 bits.

Fuente: [ALEX93]

- 4.4. Considere la siguiente palabra de 8 bits almacenada en memoria: 11000010. Utilizando el algoritmo de Hamming, determine qué bits de comprobación se memorizarían junto con la palabra de datos. Muestre cómo ha obtenido el resultado.
- 4.5. Para la palabra de datos de 8 bits 00111001, los bits de comprobación que se memorizan junto con ella serían: 0111. Suponga que al leer la palabra de memoria se calculan los bits de comprobación: 1101. ¿Cuál es la palabra de datos leída de memoria?
- 4.6. ¿Cuántos bits de comprobación se necesitan para utilizar el código de corrección de errores de Hamming para la detección de errores de un solo bit en una palabra de datos de 1.024 bits?
- 4.7. Desarrolle un código SEC para palabras de datos de 16 bits. Genere el código para la palabra de datos 010100000111001. Demuestre que el código identificará correctamente un error en el bit de datos 4.
- 4.8. Una cache asociativa por conjuntos consta de 64 líneas, divididas en conjuntos de 4. La memoria principal contiene 4K bloques de 128 palabras cada uno. Muestre el formato de direcciones de memoria principal.
- 4.9. Para las direcciones hexadecimales de memoria principal: 11111. 666666. BBBBBB; muestre en formato hexadecimal la siguiente información:
 - Los valores de etiqueta, línea y palabra para una cache con correspondencia directa, utilizando el formato de la Figura 4.18.

- b) Los valores de etiqueta y de palabra para una cache asociativa, utilizando el formato de la Figura 4.20.
- c) Los valores de etiqueta, conjunto y palabra para una cache asociativa por conjuntos de dos vías, utilizando el formato de la Figura 4.22.
- 4.10. Considere un microprocesador de 32 bits que tiene una cache on-chip de 16 KBytes asociativa por conjuntos de cuatro vías. Suponga que la cache tiene un tamaño de línea de cuatro palabras de 32 bits. Dibuje un diagrama de bloques de esta cache mostrando su organización y cómo se utilizan los diferentes campos de dirección para determinar un acierto/fallo de cache. ¿Dónde se asigna, dentro de la cache, la palabra de la posición de memoria ABCDE8F8?
- Fuente: [ALEX93].
- 4.11. Dadas las siguientes especificaciones para una memoria cache externa: asociativa por conjuntos de cuatro vías; tamaño de línea de dos palabras de 16 bits; capaz de albergar un total de 4K palabras de 32 bits de la memoria principal; utilizada con un procesador de 16 bits que emite direcciones de 24 bits. Diseñe la estructura de cache con toda la información pertinente, y muestre cómo interpreta las direcciones del procesador.
- Fuente: [ALEX93].
- 4.12. El Intel 80486 tiene una cache unificada on-chip. Esta contiene 8 KBytes y tiene una organización asociativa por conjuntos de cuatro vías y una longitud de bloque de cuatro palabras de 32 bits. La cache está estructurada en 128 conjuntos. Hay un único «bit de línea válida» y tres bits, B0, B1, y B2 (los bits de LRU), por conjunto. En un fallo de cache, el 80486 lee una línea de 16 bytes de memoria principal en una ráfaga de lectura de memoria a través del bus. Dibuje un diagrama simplificado de la cache, y muestre cómo son interpretados los diferentes campos de la dirección.
- Fuente: [ALEX93].
- 4.13. Considere una máquina con una memoria principal de 216 bytes, direccionable por bytes, y un tamaño de bloque de 8 bytes. Suponga que con esta máquina se utiliza una cache de 32 líneas y correspondencia directa.
- ¿Cómo se divide la dirección de memoria de 16 bits entre etiqueta, número de línea y número de byte?
 - En qué líneas se almacenarían los bytes que se encuentran en las siguientes direcciones?

0001 0001 0001 1011
 1100 0011 0011 0100
 1101 0000 0001 1101
 1010 1010 1010 1010
 - Suponga que se almacena en la cache el byte de dirección 0001 1010 0001 1010. ¿Cuáles son las direcciones de los bytes que se almacenan junto con él?
 - ¿Cuántos bytes de memoria pueden almacenarse en total en la cache?
 - ¿Por qué se almacenan también las etiquetas en la cache?

- 4.14. El algoritmo de sustitución del Intel 486 es denominado pseudo-LRU. Asociados con cada uno de los 128 conjuntos de cuatro líneas (etiquetadas L0, L1, L2, L3) hay tres bits, B0, B1 y B2. El algoritmo de sustitución opera así: cuando se debe sustituir una línea, la cache determinará primero si el uso más reciente fue de L0 y L1 o de L2 y L3. Entonces la cache determinará cuál de la pareja de bloques fue utilizado menos recientemente y lo marcará para sustituirlo.
- Especifique cómo se ponen los bits B0, B1 y B2, y cómo se utilizan estos en el algoritmo de sustitución.
 - Muestre cómo el algoritmo del 80486 aproxima a un algoritmo LRU verdadero.
 - Demuestre que un algoritmo LRU verdadero requeriría seis bits por conjunto.
- 4.15. Una cache asociativa por conjuntos tiene un tamaño de bloque de cuatro palabras de 16 bits y un tamaño de conjunto de 2. La cache puede acomodar un total de 4.096 palabras. El tamaño de memoria principal que es transferible a cache es de $64K \times 32$ bits. Diseñe la estructura de cache, y muestre cómo son interpretadas las direcciones del procesador.

Fuente: [ALEX93].

- 4.16. Describa una técnica sencilla para implementar un algoritmo de sustitución LRU en una cache asociativa por conjuntos de cuatro vías.
- 4.17. Considere el siguiente código:
- ```
for (i = 0; i < 20; i++)
 for (j = 0; j < 10; j++)
 a[i] = a[i] * j
```
- Muestre un ejemplo de localidad espacial en el código.
  - Muestre un ejemplo de localidad temporal en el código.
- 4.18. Generalice las ecuaciones (4.1) y (4.2) del Apéndice 4A. a jerarquías de memoria de  $N$  niveles.
- 4.19. Un computador contiene una memoria principal de 32K palabras de 16 bits. Tiene también una cache de 4K palabras dividida en conjuntos de 4 líneas con 64 palabras por línea. Suponga que la cache está inicialmente vacía. El procesador capta palabras de las posiciones 0, 1, 2, ..., 4.351, en ese orden. Entonces repite esta secuencia de captación 9 veces más. La cache es 10 veces más rápida que la memoria principal. Estime la mejora resultante por el uso de la cache. Suponga una política LRU para la sustitución de bloques.

- 4.20. Considere un sistema de memoria con los siguientes parámetros:

$$T_c = 100 \text{ ns} \quad C_c = 0,01 \text{ centavos/bit}$$

$$T_m = 1.200 \text{ ns} \quad C_m = 0,001 \text{ centavos/bit}$$

- ¿Cuál es el coste de una memoria principal de 1 MByte?
- ¿Cuál es el coste de una memoria principal de 1 MByte utilizando la tecnología de la cache?

- c) Si el tiempo de acceso efectivo es un 10 % mayor que el tiempo de acceso de la cache, ¿cuál es la tasa de aciertos  $H$ ?
- 4.21. Un computador dispone de una cache, memoria principal y un disco utilizado para memoria virtual. Cuando se referencia una palabra que está en la cache, se requieren 20 ns para acceder a ella. Si está en memoria principal pero no en la cache, se necesitan 60 ns para cargarla en la cache, y entonces se inicia de nuevo la referencia. Si la palabra no está en memoria principal se necesitan 12 ms para captarla de disco, seguidos de 60 ns para copiarla en la cache, comenzando entonces nuevamente la referencia. La tasa de aciertos de cache es 0.9 y la de memoria principal 0.6. ¿Cuál es, en nanosegundos, el tiempo medio necesario para acceder a una palabra referenciada en este sistema?

#### APÉNDICE 4A. PRESTACIONES DE LAS MEMORIAS DE DOS NIVELES

En este capítulo se ha hecho referencia a la cache que actúa como buffer entre la memoria principal y el procesador, creando una memoria interna de dos niveles. Esta arquitectura de dos niveles proporciona mejores prestaciones que una memoria comparable de un nivel, explotando una propiedad conocida como «localidad», la cual se analiza más adelante en este apéndice.

El mecanismo de cache de la memoria principal es parte de la arquitectura del computador, implementada en hardware, y normalmente invisible para el sistema operativo. Además, hay otros dos ejemplos de memorias de dos niveles que también aprovechan la localidad y que se implementan, al menos parcialmente, en el sistema operativo: la memoria virtual y la cache de disco (Tabla 4.8). La memoria virtual se verá en el Capítulo 7; la cache de disco queda fuera del alcance de este libro pero es examinada en [STAL98]. En este apéndice veremos algunas características sobre las prestaciones de las memorias de dos niveles que son comunes a las tres aproximaciones mencionadas.

#### LOCALIDAD

La base para la mejora de prestaciones de una memoria de dos niveles es un principio conocido como *localidad de las referencias* [DENN68]. Este principio establece que las referencias a memoria tienden a formar agrupaciones («clusters»). A lo largo de un período de tiempo largo, las agrupaciones en uso cambian, pero durante períodos cortos, el procesador trabaja fundamentalmente con agrupaciones fijas de referencias a memoria.

Tabla 4.8. Características de las memorias de dos niveles

|                                                                   | Cache de memoria principal        | Memoria virtual (paginación)                   | Cache de disco                       |
|-------------------------------------------------------------------|-----------------------------------|------------------------------------------------|--------------------------------------|
| Relaciones de tiempos de acceso típicas                           | 5/1                               | 1000/1                                         | 1000/1                               |
| Sistema de gestión de memoria                                     | Implementado en hardware especial | Combinación de hardware y software del sistema | Software del sistema                 |
| Tamaño de bloque típico<br>Acceso del procesador al segundo nivel | 4 a 128 bytes<br>Acceso directo   | 64 a 4.096 bytes<br>Acceso indirecto           | 64 a 4.096 bytes<br>Acceso indirecto |

Intuitivamente, el principio de localidad tiene sentido. Considérense la siguiente secuencia de razonamientos:

1. Excepto para instrucciones de bifurcación y de llamada, la ejecución de un programa es secuencial. Por tanto, en la mayoría de los casos, la siguiente instrucción a captar sigue inmediatamente a la última captada.
2. Es raro tener una secuencia larga ininterrumpida de llamadas a procedimientos seguidas por la correspondiente secuencia de retornos. En su lugar, un programa queda confinado a una ventana bastante estrecha de profundidad o nivel de anidamiento de procedimientos. Así pues, a lo largo de un período de tiempo corto las referencias a instrucciones tienden a localizarse en unos cuantos procedimientos.
3. La mayoría de las construcciones iterativas constan de un número relativamente pequeño de instrucciones repetidas muchas veces. Durante una iteración, el procesamiento está, por tanto, confinado a una pequeña porción contigua del programa.
4. En muchos programas, gran parte del cálculo incumbe al procesamiento de estructuras de datos, tales como matrices o secuencias de registros. En muchos casos, las referencias sucesivas a estas estructuras de datos serán a unidades de datos ubicados próximos entre sí.

Esta secuencia de razonamientos ha sido confirmada en muchos estudios. En relación al punto primero, se han hecho diversos estudios para analizar el comportamiento de programas en lenguajes de alto nivel. La Tabla 4.9 recoge, de los estudios que se indican, resultados clave que miden la aparición de distintos tipos de sentencias durante la ejecución. El primero de los estudios sobre el comportamiento de lenguajes de programación, realizado por Knuth [KNU71], evaluó un conjunto de programas FORTRAN utilizados como ejercicios de estudiantes. Tanenbaum [TANE78] publicó medidas recopiladas de más de 300 procedimientos utilizados en programas de sistemas operativos, y escritos en un lenguaje que soporta programación estructurada (SAL). Patterson y Sequein [PATT82a] analizaron un conjunto de medidas tomadas de compiladores y programas para composición de textos, CAD, ordenación, y comparación de ficheros. Se estudiaron los lenguajes de programación C y Pascal. Huck [HUCK83] analizó cuatro programas ideados para una mezcla de cálculos científicos de uso general, incluyendo la transformada rápida de Fourier y la resolución de sistemas de ecuaciones diferenciales. Hay bastante coincidencia, en los resultados de esta mezcla de lenguajes y de aplicaciones, en que las instrucciones de bifurcación y de llamada representan sólo una fracción de las sentencias ejecutadas durante el tiempo de vida de un programa. Estos estudios confirman pues la afirmación primera anterior.

Con respecto a la segunda afirmación, estudios aportados en [PATT85a] la confirman. Esto se ilustra en la Figura 4.29, que muestra el comportamiento de la pareja llamada/retorno.

Tabla 4.9. Frecuencia dinámica relativa de operaciones en lenguajes de alto nivel

| Estudio<br>Lenguaje<br>Tipo de trabajo | [HUCK83]<br>Pascal<br>Científico | [KNU71]<br>FORTRAN<br>Estudiante | [PATT82]          |              | [TANE78]<br>SAL<br>Sistema |
|----------------------------------------|----------------------------------|----------------------------------|-------------------|--------------|----------------------------|
|                                        |                                  |                                  | Pascal<br>Sistema | C<br>Sistema |                            |
| Assign                                 | 74                               | 67                               | 45                | 38           | 42                         |
| Loop                                   | 4                                | 3                                | 5                 | 3            | 4                          |
| Call                                   | 1                                | 3                                | 15                | 12           | 12                         |
| IF                                     | 20                               | 11                               | 29                | 43           | 36                         |
| GOTO                                   | 2                                | 9                                | —                 | 3            | —                          |
| Otras                                  | —                                | 7                                | 6                 | 1            | 6                          |

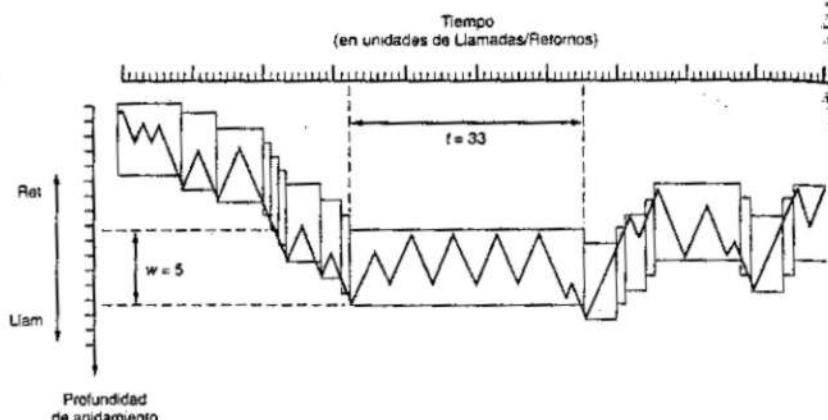


Figura 4.29. Comportamiento de las llamadas/retornos de los programas.

Cada llamada es representada mediante la línea hacia abajo y hacia la derecha, y cada retorno mediante la línea hacia arriba y a la derecha. En la figura se ha definido una *ventana* con profundidad igual a 5. Sólo una secuencia de llamadas y retornos con variación de 6 en cualquier dirección hace que se traslade la ventana. Como puede verse, el programa en ejecución puede mantenerse dentro de una ventana estacionaria por períodos de tiempo bastante largos. Un estudio por los mismos analistas de programas en C y en Pascal mostró que una ventana de profundidad 8 sólo necesitaría desplazarse en menos del 1 % de las llamadas o retornos [TAMI83].

El principio de localidad de las referencias continúa siendo validado en estudios más recientes. Por ejemplo, la Figura 4.30 muestra los resultados de un estudio sobre patrones de acceso a páginas Web en un mismo sitio Web.

En la literatura se distingue entre localidad espacial y temporal. La localidad espacial se refiere a la tendencia durante la ejecución a involucrar múltiples posiciones de memoria que estén agrupadas. La localidad espacial refleja la tendencia del procesador a acceder a las ins-

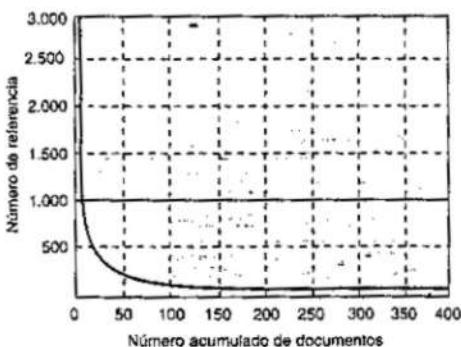


Figura 4.30. Localidad de referencias para páginas Web (BAEN97).

trucciones secuencialmente, y también la tendencia de los programas a acceder a posiciones de datos consecutivas, como, por ejemplo, cuando se procesa una tabla de datos. La localidad temporal hace referencia a la tendencia del procesador a acceder a posiciones de memoria que han sido utilizadas recientemente. Por ejemplo, cuando se ejecutan iteraciones de un bucle, el procesador ejecuta repetidamente el mismo conjunto de instrucciones.

## FUNCIONAMIENTO DE LA MEMORIA DE DOS NIVELES

La propiedad de localidad puede ser aprovechada formando una memoria de dos niveles. La memoria del nivel superior (M1) es más pequeña, más rápida y más costosa (por bit) que la del nivel inferior (M2). M1 se utiliza como almacenamiento temporal para una parte del contenido de la otra más grande, M2. Cuando se hace una referencia a memoria, se intenta acceder al elemento en M1. Si tiene éxito, entonces tiene lugar un acceso rápido. Si no, se copia un bloque de posiciones de memoria de M2 a M1, y el acceso se hace vía M1. Debido a la localidad, una vez que el bloque es llevado a M1, habrá un número de accesos a posiciones de ese bloque, resultando un servicio rápido en su conjunto.

Para expresar el tiempo medio de acceso a un elemento, debemos considerar no sólo las velocidades de los dos niveles de memoria, sino también la probabilidad de que una referencia dada se encuentre en M1. Tenemos:

$$\begin{aligned} T_s &= H \times T_1 + (1 - H) \times (T_1 + T_2) \\ &= T_1 + (1 - H) \times T_2 \end{aligned} \quad (4.1)$$

donde:

$T_s$  = tiempo de acceso medio (del sistema)

$T_1$  = tiempo de acceso de M1 (por ejemplo: cache, cache de disco)

$T_2$  = tiempo de acceso de M2 (por ejemplo: memoria principal, disco)

$H$  = tasa de aciertos (fracción de veces que la referencia es encontrada en M1)

La Figura 4.2 muestra el tiempo de acceso medio en función de la tasa de aciertos. Como puede verse, para un porcentaje de aciertos alto el tiempo de acceso total medio es mucho más próximo al de M1 que al de M2.

## PRESTACIONES

Veamos algunos parámetros relevantes a la hora de evaluar un esquema de memoria de dos niveles. Consideraremos primero el coste. Tenemos:

$$C_s = \frac{C_1 S_1 + C_2 S_2}{S_1 + S_2} \quad (4.2)$$

donde:

$C_s$  = coste medio por bit de la combinación de dos niveles de memoria

$C_1$  = coste medio por bit de la memoria M1 del nivel superior

$C_2$  = coste medio por bit de la memoria M2 del nivel inferior

$S_1$  = tamaño de M1

$S_2$  = tamaño de M2

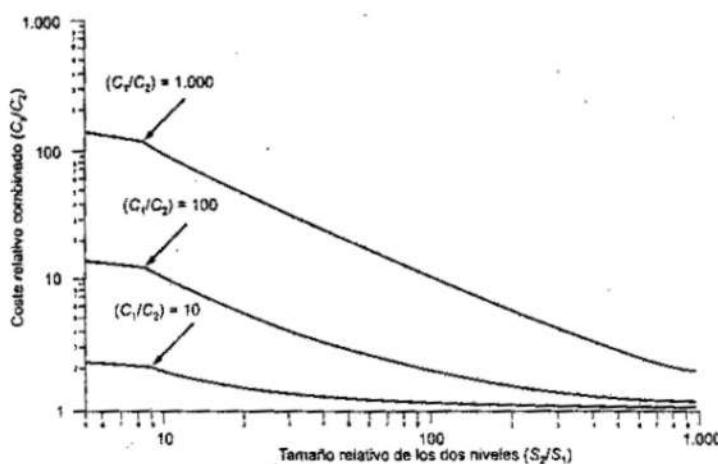


Figura 4.31. Relación entre el coste de memoria medio y el tamaño relativo de las memorias, para una memoria de dos niveles.

Nos gustaría que  $C_1 \approx C_2$ . Dado que  $C_1 \gg C_2$ , ello requiere que  $S_1 \ll S_2$ . La Figura 4.31 muestra dicha relación.

Consideremos ahora el tiempo de acceso. Para que una memoria de dos niveles suponga una mejora de prestaciones significativa, necesitamos tener  $T_1$  aproximadamente igual a  $T_2$  ( $T_1 \approx T_2$ ). Dado que  $T_1$  es mucho menor que  $T_2$  ( $T_1 \ll T_2$ ), se necesita una tasa de aciertos próxima a 1.

Así pues, nos gustaría que M1 fuera pequeña para mantener bajo el coste, y grande para mejorar la tasa de aciertos y en consecuencia las prestaciones. ¿Hay un tamaño de M1 que satisfaga razonablemente ambos requisitos? Esta pregunta la podemos desarrollar en otras:

- ¿Qué valor de tasa de aciertos se necesita para satisfacer los requisitos de prestaciones?
- ¿Qué tamaño de M1 asegurará la tasa aciertos necesaria?
- ¿Satisface dicho tamaño el requisito del coste?

Para responder, consideraremos la cantidad  $T_1/T_p$ , conocida como *eficiencia de acceso*. Es una medida de cuán próximo es el tiempo de acceso medio ( $T_p$ ) al tiempo de acceso de M1 ( $T_1$ ). De la ecuación (4.1) resulta:

$$\frac{T_1}{T_p} = \frac{1}{1 + (1 - H) \frac{T_2}{T_1}} \quad (4.3)$$

En la Figura 4.32 se representa  $T_1/T_p$  en función de la tasa de aciertos  $H$ , con la cantidad  $T_1/T_2$  como parámetro. Normalmente el tiempo de acceso a cache es entre cinco y diez veces menor que el tiempo de acceso a memoria principal (es decir  $T_2/T_1$  está entre 5 y 10), y el acceso a memoria principal es del orden de 1.000 veces más rápido que el acceso a disco ( $T_2/T_1 = 1.000$ ). Por tanto, parece necesaria una tasa de aciertos entre 0,8 y 0,9 para satisfacer el requisito de prestaciones.

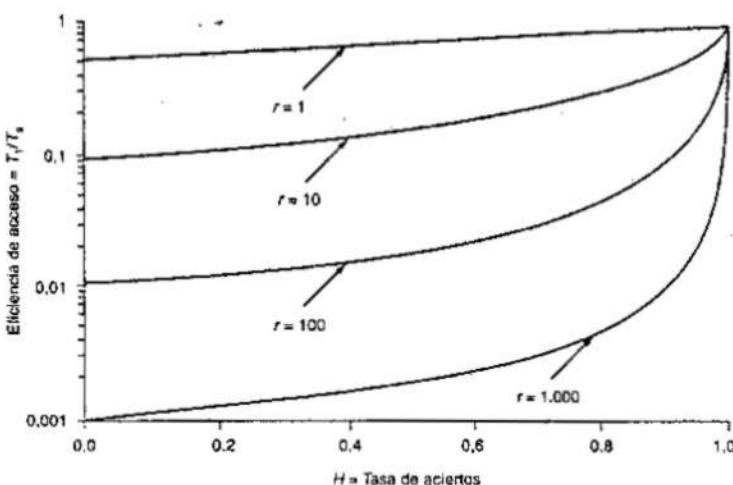


Figura 4.32. Eficiencia de acceso en función de la tasa de acierto ( $r = T_2/T_1$ ).

Ahora podemos plantear con mayor exactitud la cuestión referida al tamaño relativo de las memorias. ¿Es razonable una tasa de aciertos de 0.8 o superior con  $S_1 << S_2$ ? Eso dependerá de diversos de factores, incluida la naturaleza del software que se ejecute y los detalles del diseño de la memoria de dos niveles. Lo más decisivo es, por supuesto, el grado de localidad. La Figura 4.33 sugiere el efecto que tiene la localidad sobre la tasa de aciertos. Clara-

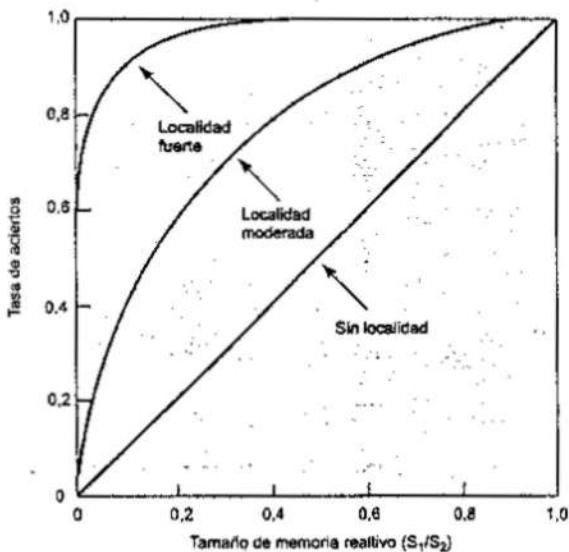


Figura 4.33. Tasa de aciertos en función del tamaño relativo de las memorias.

mente, si M1 tiene el mismo tamaño que M2, la tasa de aciertos será 1.0: todos los contenidos de M2 están también siempre almacenados en M1. Supongamos ahora que no hay localidad; esto es, que las referencias son completamente aleatorias. En este caso la tasa de aciertos sería una función estrictamente lineal del tamaño relativo de las memorias. Por ejemplo, si M1 tiene la mitad de capacidad que M2, en todo momento la mitad de los elementos de M2 están también en M1, y la tasa de aciertos será 0.5. En la práctica, sin embargo, existe cierto grado de localidad en las referencias a memoria. Los efectos de una localidad moderada o fuerte se indican en la figura.

Así pues, si la localidad es fuerte, es posible conseguir una tasa de aciertos alta, incluso con un tamaño relativamente pequeño de la memoria de nivel superior. Por ejemplo, numerosos estudios han mostrado que tamaños de cache bastante pequeños producen tasas de aciertos por encima de 0.75, *con independencia del tamaño de la memoria principal* (consultese por ejemplo [AGAR89], [PRZY88], [STRE83], y [SMIT82]). Generalmente, es adecuada una cache de 1K a 128K palabras, mientras que una memoria principal actual suele tener múltiples megabytes. Cuando consideremos la memoria virtual y la cache de disco, citaremos otros estudios que confirman el mismo fenómeno, es decir, que una M1 relativamente pequeña produce un valor elevado de la tasa de aciertos gracias a la localidad.

Esto nos conduce a la última pregunta antes planteada: ¿satisface el tamaño relativo de las dos memorias el requisito del coste? La respuesta es claramente sí. Si para conseguir buenas prestaciones necesitamos poca capacidad para la memoria de nivel superior, el coste medio por bit de la memoria de dos niveles se aproximará a la del nivel inferior, que es más económica.

## CAPÍTULO 5

# Memoria externa

### 5.1. Discos magnéticos

Organización y formatos de los datos

Características físicas

Parámetros para medir las prestaciones de un disco

### 5.2. RAID

Nivel 0 de RAID

Nivel 1 de RAID

Nivel 2 de RAID

Nivel 3 de RAID

Nivel 4 de RAID

Nivel 5 de RAID

Nivel 6 de RAID

### 5.3. Memoria óptica

CD-ROM

WORM

Disco óptico borrable

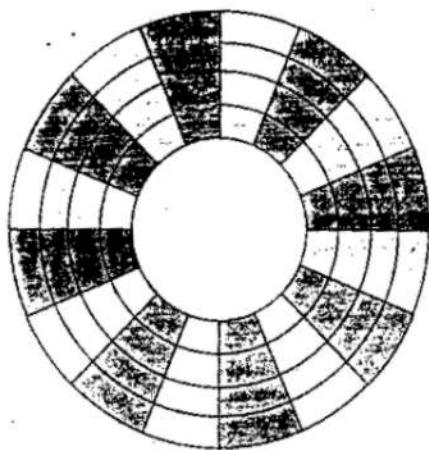
Disco vídeo digital

Discos magnético-ópticos

### 5.4. Cinta magnética

### 5.5. Lecturas y sitios Web recomendados

### 5.6. Problemas



- Los discos magnéticos siguen siendo el componente más importante de la memoria externa. Tanto los extraíbles como los fijos, o duros, los discos se usan tanto en computadores personales, como en computadores grandes y supercomputadores.
  - Para conseguir mayores prestaciones y disponibilidad, un esquema de servidores y sistemas grandes extendido es la tecnología RAID de discos. RAID se refiere a una familia de técnicas para utilizar varios discos como un conjunto de dispositivos de almacenamiento de datos en paralelo, con redundancia para compensar los fallos de disco.
  - Las técnicas de almacenamiento óptico se han convertido en algo cada vez más importante en los computadores. Mientras que el CD-ROM se ha usado ampliamente durante muchos años, tecnologías más recientes, como el CD reescribible y las unidades de almacenamiento magnético-ópticas, están siendo cada vez más importantes.

**E**n este capítulo se examinan distintos sistemas y dispositivos de memoria externa. Comenzamos con el dispositivo más importante, el disco magnético. Los discos magnéticos son la base de las memorias externas en casi todos los computadores. En la siguiente sección se examina el uso de conjuntos de discos para conseguir mayores prestaciones, concretamente la familia conocida como RAID (Redundant Array of Independent Disks, «conjunto redundante de discos independientes»). La memoria óptica externa es un componente cada vez más importante de muchos computadores, y se examinará en la tercera sección. Al final, se describen las cintas magnéticas.

### 5.1 DISCOS MAGNETICOS

Un disco magnético es un plato circular construido con metal o plástico, cubierto por un material magnetizable. Los datos se graban en él y después se recuperan del disco a través de una bobina, llamada *cabeza*. Durante una operación de lectura o escritura, la cabeza permanece quieta mientras el plato rota bajo ella.

El mecanismo de escritura se basa en el campo magnético producido por el flujo eléctrico que atraviesa la bobina. Se envían pulsos a la cabeza, y se graban patrones magnéticos en la superficie bajo ella, con patrones diferentes para corrientes positivas y negativas. El mecanismo de lectura se basa en la corriente eléctrica que atraviesa la bobina, producida por un campo magnético que se mueve respecto a la bobina. Cuando la superficie del disco pasa bajo la cabeza, se genera en ésta una corriente de la misma polaridad que la que produjo la grabación magnética.

### ORGANIZACIÓN Y FORMATO DE LOS DATOS

La cabeza es un dispositivo relativamente pequeño, capaz de leer o escribir en una zona del plato que rota bajo ella. Esto da lugar a que los datos se organicen en un conjunto de anillos concéntricos en el plato, llamados *pistas*. Cada pista es del mismo ancho que la cabeza. Usualmente hay de 500 a 2.000 pistas por superficie.

En la Figura 5.1 se puede ver la disposición de los datos. Las pistas adyacentes están separadas por bandas vacías. Esto previene, o por lo menos minimiza, los errores debidos a desalineamientos de la cabeza o simplemente a interferencias del campo magnético. Para simplificar la electrónica, se suele almacenar el mismo número de bits en cada pista. Entonces, la densidad, en bits por pulgada lineal, aumenta según se avanza de la pista más externa a la más interna.

Como se mencionó anteriormente, los datos se transfieren hacia, y desde, el disco en bloques. Normalmente, el bloque es menor que la capacidad de una pista. De acuerdo con esto, los datos se almacenan en regiones del tamaño de un bloque, conocidas como *sectores* (Figura 5.1). Normalmente hay entre 10 y 100 sectores por pista, y estos pueden ser de longitud fija o variable. Para evitar imposiciones de precisión ilógicas del sistema, los sectores adyacentes se separan con intrapistas (intersectores) vacías.

Se necesita algún medio para identificar las posiciones de los sectores dentro de una pista dada. Claramente, debe haber algún punto de comienzo de la pista, y una manera de identificar el principio y el fin de cada sector. Estos requisitos son gestionados mediante datos de control grabados en el disco. Por tanto, el disco se graba con un formato que contiene algunos datos extra, usados sólo por el controlador del disco y no accesibles al usuario.

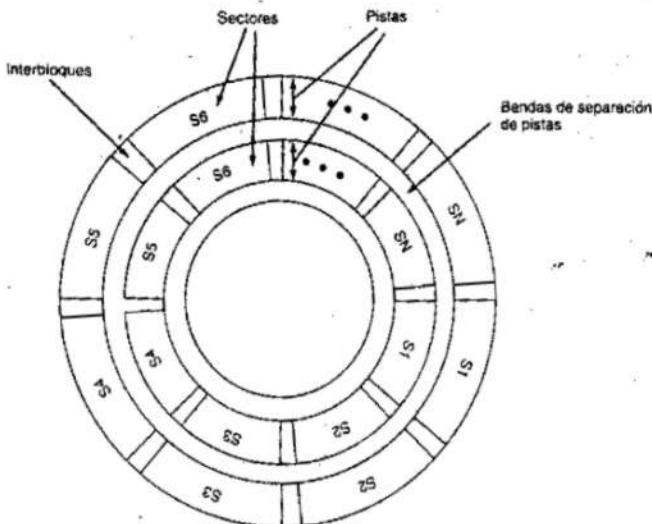


Figura 5.1. Organización de los datos en el disco.

En la Figura 5.2 se muestra un ejemplo del formato de grabación de un disco. En este caso, cada pista contiene 30 sectores de longitud fija, de 600 bytes cada uno. Cada sector contiene 512 bytes de datos, más información de control útil al controlador del disco. El campo ID es un identificador único, o dirección, usado para localizar un sector particular. El byte SINCRO es un patrón de bits especial que delimita el comienzo del campo. El número de pista identifica una pista en una superficie. El número de cabeza identifica una cabeza, si el disco tiene varias superficies (como acabamos de explicar). El ID y los campos de datos contienen, cada uno, un código de detección de errores.

## CARACTERÍSTICAS FÍSICAS

En la Tabla 5.1 se listan las principales características que diferencian los distintos tipos de discos. Primero, las cabezas pueden ser fijas o móviles con respecto a la dirección radial del plato. En un disco de cabeza fija, hay una cabeza de lectura/escritura por pista. Todas las cabezas se montan en un brazo rígido que se extiende a través de todas las pistas (Figura 5.3a). En un disco de cabeza móvil, hay sólo una cabeza de lectura/escritura (Figura 5.3b). Como antes, la cabeza se monta en un brazo. Como la cabeza debe poder posicionarse encima de cualquier pista, el brazo debe extenderse o retraerse para este propósito.

El disco mismo se monta en una unidad de disco, que consta del brazo, un eje que hace rotar el disco, y la electrónica necesaria para la entrada y salida de datos binarios. Un disco no extraíble está permanentemente montado en la unidad de disco. Un disco extraíble puede ser quitado y sustituido por otro disco. La ventaja de este último tipo es que es posible una cantidad de datos ilimitada con un número limitado de unidades de disco. Además, un disco puede ser utilizado en diversos computadores.

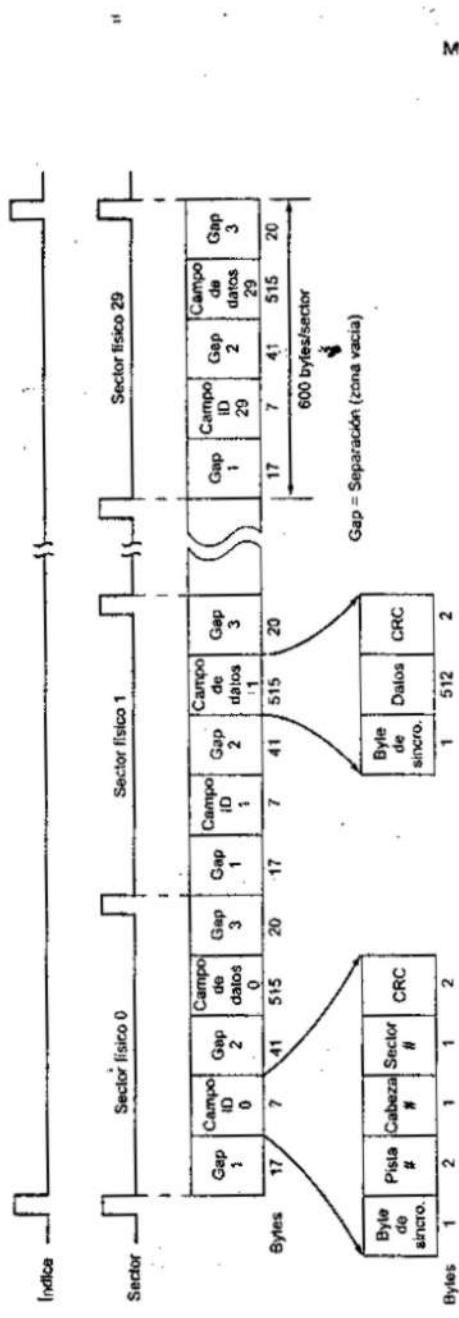


Figura 5.2. Formato de las pistas de un disco Winchester (Seagate ST506).

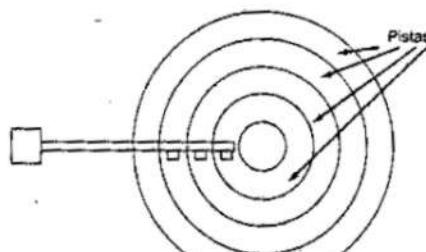
Tabla 5.1. Características de los sistemas de discos

|                                   |                                      |
|-----------------------------------|--------------------------------------|
| <b>Desplazamiento de cabezas</b>  | <b>Platos</b>                        |
| Cabeza fija (una por pista)       | Plato único                          |
| Cabeza móvil (una por superficie) | Múltiples platos                     |
| <b>Transportabilidad de disco</b> | <b>Mecanismo de la cabeza</b>        |
| Disco fijo                        | Contacto (disquete)                  |
| Disco extraíble                   | Separación fija                      |
| <b>Superficies</b>                | Separación aerodinámica (Winchester) |
| Superficie única                  |                                      |
| Superficie doble                  |                                      |

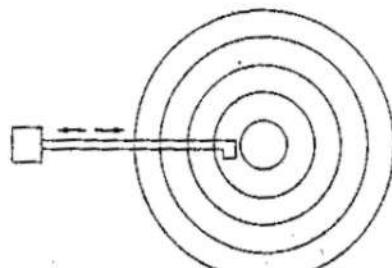
En la mayoría de los discos, la cubierta magnetizable se aplica a ambas caras del plato, denominándose estos discos de doble superficie. Algunos discos, menos caros, son de una sola superficie.

Algunas unidades de disco poseen varios platos, apilados verticalmente y separados por una distancia de alrededor de una pulgada (Figura 5.4). Disponen de varios brazos. Los platos constituyen una unidad llamada paquete de disco.

Finalmente, el mecanismo de la cabeza proporciona una clara clasificación de los discos en tres tipos. Tradicionalmente, la cabeza de lectura/escritura se posiciona a una distancia fija sobre el plato, dejando entre ambos una capa de aire. En el otro extremo está el mecanismo



(a) Cabeza fija



(b) Cabeza móvil

Figura 5.3. Discos con cabeza fija y móvil.

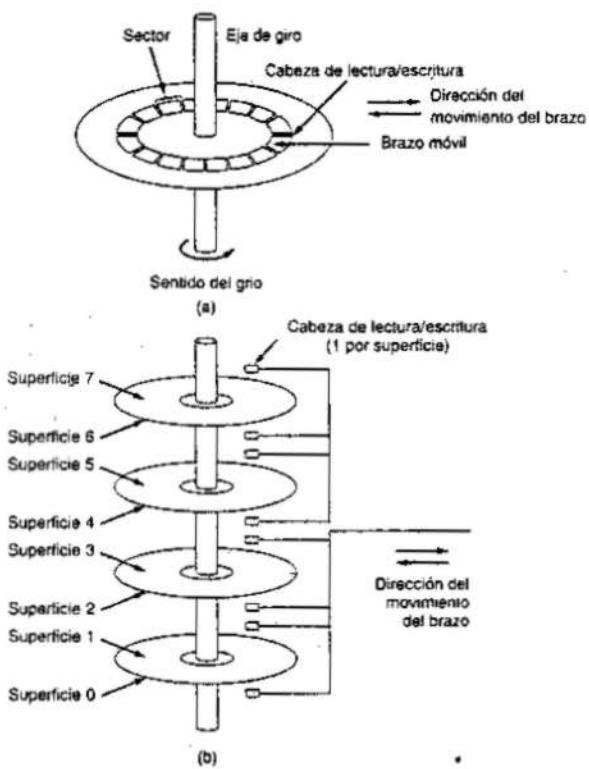


Figura 5.4. Disco con varios platos.

de la cabeza que realmente efectúa un contacto físico con el medio durante la operación de lectura o escritura. Este mecanismo es el que se usa con los disquetes, que son pequeños y de plato flexible, y es el tipo de disco más barato.

Para entender el tercer tipo de disco, necesitamos comentar la relación entre la densidad de datos y la anchura de la capa de aire. La cabeza debe generar o detectar un campo magnético de intensidad suficiente para escribir y leer correctamente. Cuanto más estrecha es la cabeza, más cercana debe estar a la superficie del plato para funcionar. Esto es deseable, ya que una cabeza más estrecha implica pistas más estrechas y, por tanto, mayor densidad de datos. Sin embargo, cuanto más cerca esté la cabeza del disco, mayor será el riesgo de error debido a impurezas o imperfecciones. Los discos Winchester supusieron un avance tecnológico, en este sentido. Las cabezas de los Winchester están montadas en unidades herméticamente cerradas, que están casi libres de contaminación. Fueron diseñados para operar más cerca de la superficie del disco que las cabezas de los discos rígidos anteriores, por tanto permiten una densidad de datos mayor. La cabeza está en el contorno de una hoja de metal aerodinámica, que reposa suavemente sobre la superficie del plato cuando el disco no se mueve. La presión del aire generada por el giro del disco es suficiente para hacer subir la hoja encima de la superficie. El sistema sin contacto resultante puede ser diseñado para usar cabezas más

estrechas que las de los discos rígidos convencionales, operando más cerca de la superficie de los platos<sup>1</sup>.

### PARÁMETROS PARA MEDIR LAS PRESTACIONES DE UN DISCO

Los detalles de las operaciones de E/S de un disco dependen del tipo de computador, del sistema operativo y de la naturaleza de los canales de E/S y del hardware controlador del disco. En la Figura 5.5 se muestra un diagrama de temporización general de las transferencias de E/S del disco.

Cuando la unidad de disco está funcionando, el disco está rotando a una velocidad constante. Para leer o escribir, la cabeza debe posicionarse en la pista deseada y al principio del sector deseado en la pista. La selección de la pista implica un movimiento de la cabeza (en un sistema de cabeza móvil) o una selección electrónica de una cabeza (en un sistema de cabezas fijas). En un sistema de cabeza móvil, el tiempo que tarda la cabeza en posicionarse en la pista se conoce como *tiempo de búsqueda*. En cualquier caso, una vez seleccionada la pista, el controlador del disco espera hasta que el sector apropiado rote hasta alinearse con la cabeza. El tiempo que tarda el sector en alcanzar la cabeza se llama *retardo rotacional*, o latencia rotacional. La suma del tiempo de búsqueda, si lo hay, y el retardo rotacional se denomina *tiempo de acceso*, o tiempo que se tarda en llegar a la posición de lectura o escritura. Una vez posicionada la cabeza, se lleva a cabo la operación de lectura o escritura, desplazándose el sector bajo la cabeza; esta operación conlleva un *tiempo de transferencia de datos*.

Además del tiempo de acceso y de transferencia, hay varios retardos en cola usualmente asociados con operaciones de E/S del disco. Cuando un proceso hace una petición de E/S, primero debe esperar en cola hasta que el dispositivo esté disponible. En ese momento, el dispositivo es asignado al proceso. Si el dispositivo comparte un único canal E/S o un conjunto de canales de E/S con otros discos, entonces puede tener que hacer esperas adicionales para que el canal esté disponible. En este punto, se hace la búsqueda para empezar el acceso al disco.

En algunos computadores grandes, se usa una técnica conocida como detección de posición rotacional (RPS, «rotational positional sensing»). Esta funciona de la siguiente forma: cuando se lleva a cabo una orden de búsqueda, el canal es liberado para atender otras operaciones de E/S. Cuando la búsqueda se ha completado, el dispositivo determina cuándo se rotan los datos bajo la cabeza. Mientras el sector se aproxima a la cabeza, el dispositivo intenta restablecer el camino de comunicación hacia el anfitrión. Si la unidad de control o el canal están ocupados con otra E/S, la conexión puede fallar y el dispositivo debe rotar una vuelta completa antes de que pueda intentar conectarse de nuevo, lo que se denomina una

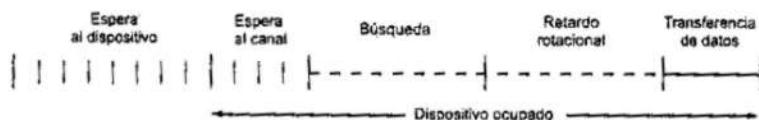


Figura 5.5. Temporización de transferencia de un disco de E/S.

<sup>1</sup> Como información de interés histórico, el término *Winchesier* fue usado originalmente por IBM como nombre preliminar para el modelo de disco 3340. El 3340 era un paquete de discos extraíble con las cabezas integradas en el paquete. El término se aplica ahora a cualquier unidad de disco integrada con un diseño de cabezas aerodinámico. El disco *Winchester* se usa habitualmente en PC y estaciones de trabajo, y se le suele llamar *disco duro*.

pérdida RPS. Esto supone un retardo extra, que se debe añadir a la línea de tiempo de la Figura 5.5.

### Tiempo de búsqueda

El tiempo de búsqueda es el tiempo necesario para desplazar el brazo del disco hasta la pista requerida. Este tiempo resulta difícil de precisar. El tiempo de búsqueda está formado por varios componentes clave: el tiempo inicial de comienzo y el tiempo necesario para atravesar las pistas que tienen que cruzarse una vez que el brazo de acceso esté a la velocidad adecuada. El tiempo transversal no es, desgraciadamente, una función lineal del número de pistas. El tiempo de búsqueda se puede aproximar con la siguiente fórmula lineal:

$$T_s = m \times n + s$$

donde:

$T_s$  = tiempo de búsqueda estimado

$n$  = número de pistas atravesadas

$m$  = constante que depende del disco

$s$  = tiempo de comienzo

Por ejemplo, un disco duro barato de un PC puede tener aproximadamente  $m = 0.3$  ms y  $s = 20$  ms, mientras que otro mayor, más caro, puede tener  $m = 0.1$  ms y  $s = 3$  ms.

### Retardo rotacional

Los discos distintos de los disquetes normalmente rotan a 3.600 rpm, que es una revolución cada 16,7 ms. Por tanto, de media, el retardo rotacional será de unos 8,3 ms. Las disqueteras normalmente rotan entre 300 y 600 rpm. Por tanto, el retardo medio estará entre los 100 y 200 ms.

### Tiempo de transferencia

El tiempo de transferencia hacia o desde el disco depende de la velocidad de rotación del disco de la siguiente forma:

$$T = \frac{b}{rN}$$

donde:

$T$  = tiempo de transferencia

$b$  = número de bytes a transferir

$N$  = número de bytes de una pista

$r$  = velocidad de rotación en revoluciones por segundo

Por tanto, el tiempo de acceso medio total se puede expresar como

$$T_a = T_s + \frac{1}{2r} + \frac{b}{rN}$$

donde  $T_s$  es el tiempo de búsqueda medio.

### Una comparación de tiempos

Con los parámetros definidos anteriormente, veamos dos operaciones de E/S diferentes, que ilustrarán el peligro de fiarse de los valores medios. Considerese un disco típico con un tiempo de búsqueda medio especificado de 20 ms, una velocidad de transferencia de 1 Mbyte/s, y sectores de 512 bytes con 32 sectores por pista. Supóngase que queremos leer un fichero que consta de 256 sectores con un total de 128 kbytes. Queremos estimar el tiempo total de transferencia.

Primero, supongamos que el fichero está almacenado de la forma más compacta posible en el disco. Es decir, el fichero ocupa todos los sectores de ocho pistas adyacentes (8 pistas  $\times$  32 sectores/pista = 256 sectores). Esto se conoce como *organización secuencial*. Ahora, el tiempo para leer la primera pista es el siguiente:

|                    |         |
|--------------------|---------|
| Tiempo de búsqueda | 20,0 ms |
| Retardo rotacional | 8,3 ms  |
| Leer 32 sectores   | 16,7 ms |
|                    | <hr/>   |
|                    | 45 ms   |

Supongamos que el resto de las pistas se puede leer ahora sin prácticamente tiempo de búsqueda. Es decir, la operación de E/S puede mantenerse con un flujo continuo desde el disco. Entonces se necesita considerar, al menos, un retardo rotacional para cada pista leída. Entonces, cada pista siguiente se lee en  $8,3 + 16,7 = 25$  ms. Para leer el fichero entero:

$$\text{Tiempo total} = 45 + 7 \times 25 = 220 \text{ ms} = 0,22 \text{ s}$$

Ahora calculemos el tiempo requerido para leer los mismos datos utilizando acceso aleatorio en vez de secuencial; es decir, los accesos a los sectores se distribuyen aleatoriamente sobre el disco. Para cada sector tenemos

|                    |         |
|--------------------|---------|
| Tiempo de búsqueda | 20,0 ms |
| Retardo rotacional | 8,3 ms  |
| Leer 1 sector      | 0,5 ms  |
|                    | <hr/>   |
|                    | 28,8 ms |

$$\text{Tiempo total} = 256 \times 28,8 = 7.373 \text{ ms} = 7,37 \text{ s}$$

Está claro que el orden en que se lean los sectores desde el disco tiene una repercusión enorme en las prestaciones de E/S. En el caso de acceso a ficheros en los que selean o escriban varios sectores, se tiene un cierto control sobre la forma en la que los sectores o datos se organizan, y debemos decir algo sobre este tema en el siguiente capítulo. Sin embargo, aún en el caso de un acceso a un fichero en un entorno de multiprogramación, habrá peticiones de E/S compitiendo por el mismo disco. Entonces, merece la pena examinar las maneras en las que las prestaciones de E/S del disco mejoran respecto a las llevadas a cabo con accesos al disco puramente aleatorios. Esto conduce a considerar algoritmos de planificación del disco, que son jurisdicción de los sistemas operativos y están fuera del alcance de este libro (véase [STAL98] para más detalles).

**5.2. RAID**

Como se dijo anteriormente, el ritmo de mejora de prestaciones en memoria secundaria ha sido considerablemente menor que en procesadores y en memoria principal. Esta desigualdad ha hecho, quizás, del sistema de memoria de disco el principal foco de optimización en las prestaciones de los computadores.

Como en otras áreas de rendimiento de los computadores, los diseñadores de memorias de disco reconocen que si uno de los componentes sólo se puede llevar a un determinado límite, se puede conseguir una ganancia en prestaciones adicional usando varios de esos componentes en paralelo. En el caso de la memoria de disco, esto conduce al desarrollo de conjuntos de discos que operen independientemente y en paralelo. Con varios discos, las peticiones separadas de E/S se pueden gestionar en paralelo, siempre que los datos requeridos residan en discos separados. Además, se puede ejecutar en paralelo una única petición de E/S si el bloque de datos al que se va a acceder está distribuido a lo largo de varios discos.

Con el uso de varios discos, hay una amplia variedad de formas en las que se pueden organizar los datos y en las que se puede añadir redundancia para mejorar la seguridad. Esto podría dificultar el desarrollo de esquemas de bases de datos que se pueden usar en numerosas plataformas y sistemas operativos. Afortunadamente, la industria está de acuerdo con los esquemas estandarizados para el diseño de bases de datos para discos múltiples, conocidos como RAID (Redundant Array of Independent Disks, «conjunto redundante de discos independientes»). El esquema RAID consta de seis niveles<sup>2</sup> independientes, desde cero hasta cinco. Estos niveles no implican una relación jerárquica, sino que designan métodos diferentes que poseen tres características comunes:

1. RAID es un conjunto de unidades físicas de disco vistas por el sistema operativo como una única unidad lógica.
2. Los datos se distribuyen a través de las unidades físicas del conjunto de unidades.
3. La capacidad de los discos redundantes se usa para almacenar información de paridad que garantice la recuperación de los datos en caso de fallo de disco.

Los detalles de las características segunda y tercera, cambian según los distintos niveles RAID. RAID 0 no soporta la tercera característica.

El término RAID fue originalmente ideado en un artículo de un grupo de investigación de la Universidad de California en Berkeley [PATT88]<sup>3</sup>. El artículo perfilaba varias configuraciones y aplicaciones RAID, e introducía las definiciones de los niveles RAID que todavía se usan. La estrategia RAID reemplaza una unidad de disco de gran capacidad por unidades múltiples de menor capacidad, y distribuye los datos de forma que se puedan habilitar accesos simultáneos a los datos de varias unidades, mejorando, por tanto, las prestaciones de E/S, y permitiendo más fácilmente aumentos en la capacidad.

La única contribución de la propuesta RAID es, efectivamente, hacer hincapié en la necesidad de redundancia. El uso de varios dispositivos, además de permitir que varias cabezas y

<sup>2</sup> Algunos investigadores y compañías han definido niveles adicionales, pero los seis niveles descritos en esta sección son los convenidos universalmente.

<sup>3</sup> En este artículo, el acrónimo RAID significaba «conjunto redundante de discos baratos» (Redundant Array of Inexpensive Disks). El término *barato* se usó para contrastar los discos pequeños de los conjuntos RAID, relativamente baratos, frente a la alternativa de discos únicos, grandes y caros (SLED, single large expensive disk). Hoy, el término SLED está obsoleto, y se usan tecnologías similares, tanto para configuraciones RAID como no RAID. De acuerdo con esto, la industria ha adoptado el término *independiente*, para enfatizar que el conjunto RAID proporciona prestaciones adecuadas y mejoras de seguridad.

Tabla 5.2. Niveles RAID

| Categoría            | Nivel | Descripción                                        | Grado de E/S solicitado (lectura/escritura) | Grado de transferencia de datos (lectura/escritura) | Aplicación típica                                                   |
|----------------------|-------|----------------------------------------------------|---------------------------------------------|-----------------------------------------------------|---------------------------------------------------------------------|
| Estructura en tiras  | 0     | No redundante                                      | Tiras largas: excelente                     | Pequeñas tiras: excelente                           | Aplicaciones que requieren altas prestaciones con datos no críticos |
| Estructura en espejo | 1     | Espejo                                             | Bueno/regular                               | Regular/regular                                     | Controladores de sistemas; ficheros críticos                        |
| Acceso paralelo      | 2     | Redundante con código Hamming                      | Pobre                                       | Excelente                                           |                                                                     |
|                      | 3     | Bit de paridad intercalado                         | Pobre                                       | Excelente                                           | Aplicaciones con muchas E/S, tales como imágenes, CAD               |
| Acceso independiente | 4     | Bloque de paridad intercalado                      | Excelente/regular                           | Excelente/Pobre                                     |                                                                     |
|                      | 5     | Paridad distribuida en bloques intercalados        | Excelente/regular                           | Excelente/pobre                                     | Grado de petición alto, lectura intensiva, consulta de datos        |
|                      | 6     | Paridad distribuida dual en bloques intercambiados | Excelente/regular                           | Excelente/pobre                                     | Aplicaciones que requieren alta disponibilidad                      |

actuadores operen simultáneamente, consiguiendo mayores velocidades de E/S y de transferencia, incrementa la probabilidad de fallo. Para compensar esta disminución de seguridad, RAID utiliza la información de paridad almacenada, que permite la recuperación de datos perdidos debido a un fallo de disco.

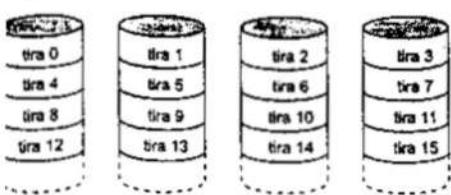
A continuación examinaremos cada nivel de RAID. En la Tabla 5.2 se hace un sumario de los seis niveles. De ellos, los niveles 2 y 4 no se ofrecen comercialmente y no es probable que consigan aceptación industrial. Sin embargo, la descripción de estos niveles ayuda a clasificar las elecciones de diseño en algunos de los otros niveles.

La Figura 5.6 ilustra los seis esquemas RAID para el caso en que se requiere una capacidad de datos de cuatro discos sin redundancia. La figura marca las zonas de datos del usuario y las de datos redundantes, y señala los requisitos relativos de almacenamiento de los distintos niveles. Nos referiremos a esta figura a lo largo de la presente discusión.

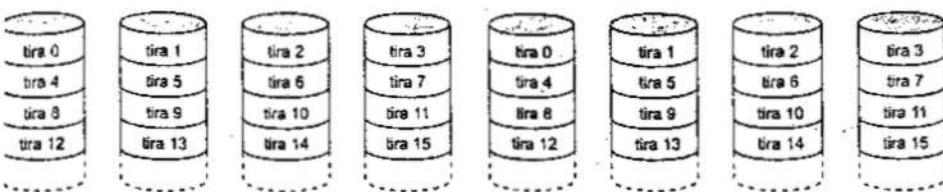
### NIVEL 0 DE RAID

El nivel 0 de RAID no es un verdadero miembro de la familia RAID, porque no incluye redundancia para mejorar las prestaciones. Sin embargo, hay algunas aplicaciones, como algunas ejecuciones en supercomputadores, en las que las prestaciones y la capacidad son la preocupación primaria, y un costo bajo es más importante que mejorar la seguridad.

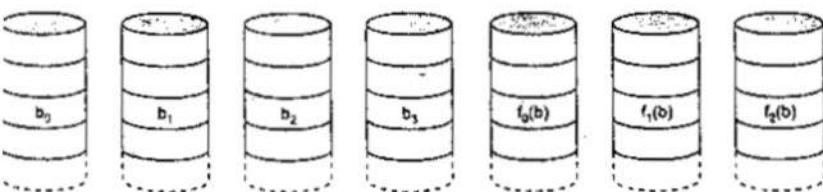
Para el RAID 0, los datos del usuario y del sistema están distribuidos a lo largo de todos los discos del conjunto. Esto tiene una notable ventaja frente al uso de un único y gran disco:



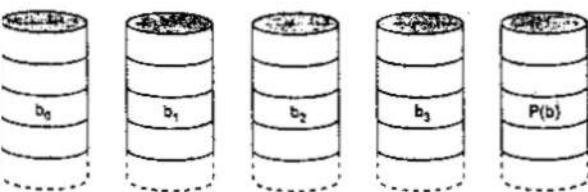
a) RAID 0 (no redundante)



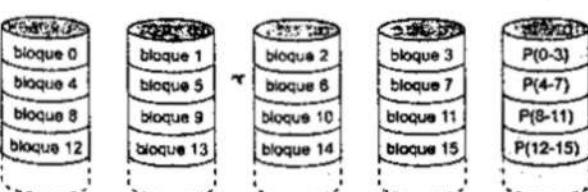
b) RAID 1 (con espejo)



c) RAID 2 (redundante con código Hamming)

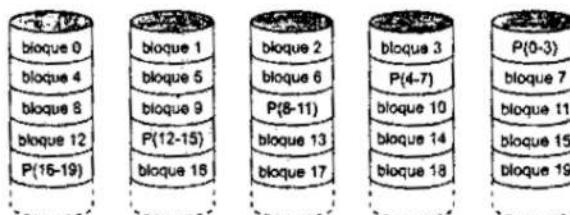


d) RAID 3 (bit de paridad intercalado)

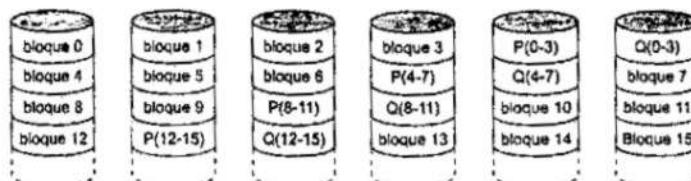


e) RAID 4 (paridad en bloques)

Figura 5.6. Niveles RAID (continúa en página siguiente).



(f) RAID 5 (paridad distribuida a nivel de bloques)



(g) RAID 6 (redundancia doble)

Figura 5.6. Niveles RAID (continuación).

si hay pendientes dos peticiones diferentes de E/S para dos bloques de datos diferentes, entonces es muy probable que los bloques pedidos estén en diferentes discos. Entonces, las dos peticiones se pueden emitir en paralelo, reduciendo el tiempo de cola de E/S.

Pero RAID 0, como todos los niveles RAID, va más lejos que una sencilla distribución de datos a través del conjunto de discos: los datos son *organizados en forma de tiras de datos* a través de los discos disponibles. Esto se entiende mejor considerando la Figura 5.6. Todos los datos del usuario y del sistema se ven como almacenados en un disco lógico. El disco se divide en tiras; estas tiras pueden ser bloques físicos, sectores o alguna otra unidad. Las tiras se proyectan cíclicamente, en miembros consecutivos del conjunto. Un conjunto de tiras lógicamente consecutivas, que se proyectan exactamente sobre una misma tira en cada miembro del conjunto, se denomina «franja». En un conjunto de  $n$  discos, las primeras  $n$  tiras lógicas (una franja) se almacenan físicamente en la primera tira de cada uno de los  $n$  discos, las segundas  $n$  tiras lógicas, se distribuyen en la segunda tira de cada disco, etc. La ventaja de esta disposición es que si una única petición de E/S implica a varias tiras lógicas contiguas, entonces las  $n$  tiras de esta petición se pueden gestionar en paralelo, reduciendo considerablemente el tiempo de transferencia de E/S.

En la Figura 5.7 se indica como el software de gestión de un conjunto proyecta el espacio del disco físico sobre el disco lógico. Este software se puede ejecutar, tanto en el subsistema de disco como en un computador anfitrión.

#### RAID 0 para alta capacidad de transferencia de datos

Las prestaciones de cualquiera de los niveles RAID dependen críticamente de los patrones de petición del sistema anfitrión y de la distribución de los datos. Estas emisiones pueden ser más claramente direccionadas en RAID 0, donde el impacto de la redundancia no interfiere con el análisis. Primero, consideremos el uso de RAID 0 para lograr una velocidad de trans-

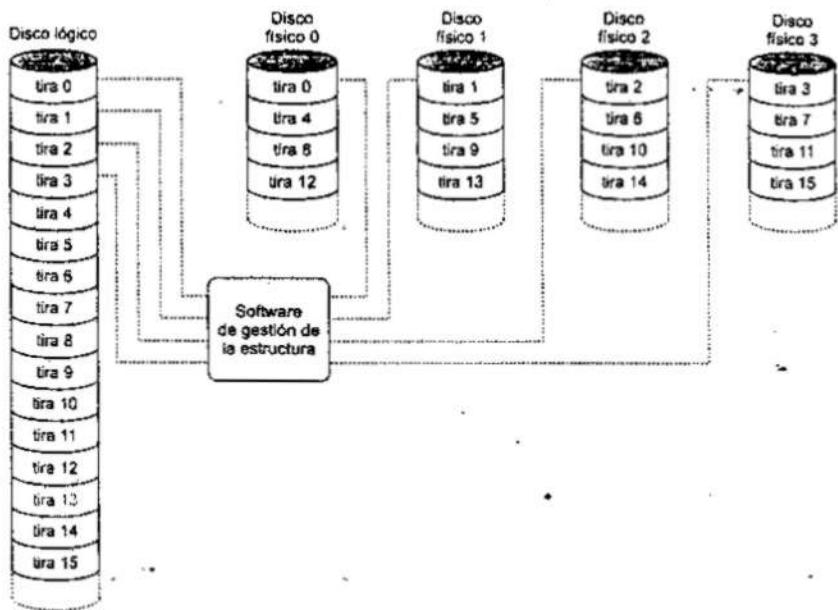


Figura 5.7. Mapa de datos para un conjunto RAID de nivel 0.

ferencia de datos alta. Se deben cumplir dos requisitos para que las aplicaciones tengan una velocidad de transferencia alta. Primero, debe existir una capacidad de transferencia alta en todo el camino entre la memoria del anfitrión y las unidades de disco individuales. Esto incluye controladores de buses internos, buses de E/S del anfitrión, adaptadores de E/S, y buses de memoria del anfitrión.

El segundo requisito es que la aplicación debe hacer peticiones de E/S que se distribuyan eficientemente sobre el conjunto de discos. Esta condición se satisface si la petición típica es de una gran cantidad de datos lógicamente contiguos, comparados con el tamaño de una cinta. En este caso, una única petición de E/S implica la transferencia paralela de datos desde varios discos, aumentando la velocidad efectiva de transferencia en comparación con la de un único disco.

#### RAID 0 para altas frecuencias de petición de E/S

En los entornos orientados a transacciones, el usuario se suele preocupar más del tiempo de respuesta que de la velocidad de transferencia. Para una petición individual de E/S de una pequeña cantidad de datos, el tiempo de E/S está dominado por el movimiento de las cabezas del disco (tiempo de búsqueda) y el movimiento del disco (latencia rotacional).

En un entorno de transacción, puede haber cientos de peticiones de E/S por segundo. Un conjunto de discos puede proporcionar velocidades altas de ejecución de E/S, balanceando la carga de E/S a través de los distintos discos. El balanceo de la carga efectiva se consigue solamente si hay varias peticiones de E/S pendientes. Esto, por turnos, implica que hay varias

aplicaciones independientes, o una única aplicación orientada a transacción que es capaz de generar varias peticiones de E/S asíncronas. Las prestaciones también se verán influidas por el tamaño de la franja. Si la franja es relativamente grande, de forma que una única petición de E/S sólo implique un único acceso a disco, entonces las peticiones de E/S que están esperando pueden ser tratadas en paralelo, reduciendo el tiempo en cola para cada petición.

## NIVEL 1 DE RAID

RAID 1 se diferencia de los niveles 2 a 5 en cómo se consigue la redundancia. En estos otros esquemas RAID, se usan algunas formas de cálculo de paridad para introducir redundancia; en RAID 1, la redundancia se logra con el sencillo recurso de duplicar todos los datos. Según muestra la Figura 5.6b, se hace una distribución de datos, como en el RAID 0. Pero en este caso, cada franja lógica se proyecta en dos discos físicos separados, de forma que cada disco del conjunto tiene un disco espejo que contiene los mismos datos.

En la organización RAID 1 hay una serie de aspectos positivos:

1. Una petición de lectura puede ser servida por cualquiera de los discos que contienen los datos pedidos; cualquiera de ellos implica un tiempo de búsqueda mínimo más la latencia rotacional.
2. Una petición de escritura requiere que las dos tiras correspondientes se actualicen, y esto se puede hacer en paralelo. Entonces, el resultado de la escritura viene determinado por la menos rápida de las dos escrituras (es decir, la que conlleva el mayor tiempo de búsqueda más la latencia rotacional). Sin embargo, en RAID 1 no hay «penalización en la escritura». Los niveles RAID del 2 al 5 implican el uso de bits de paridad. Por tanto, cuando se actualiza una única tira, el software de gestión del conjunto debe calcular y actualizar primero los bits de paridad, así como actualizar la tira en cuestión.
3. La recuperación tras un fallo es sencilla. Cuando una unidad falla, se puede acceder a los datos desde la segunda unidad.

La principal desventaja es el coste: requiere el doble del espacio de disco del disco lógico que puede soportar. Debido a esto, una configuración RAID 1 posiblemente está limitada a unidades que almacenan el software del sistema y los datos, y otros ficheros altamente críticos. En estos casos, RAID proporciona una copia de seguridad en tiempo real de todos los datos, de forma que, en caso de fallo de disco, todos los datos críticos están inmediatamente disponibles.

En un entorno orientado a transacciones, RAID 1 puede conseguir altas velocidades de petición de E/S si la mayor parte de las peticiones son lecturas. En esta situación, las prestaciones de RAID 1 son próximas al doble de las de RAID 0. Sin embargo, si una parte importante de las peticiones de E/S son peticiones de escritura, entonces la ganancia en prestaciones sobre RAID 0 puede no ser significativa. RAID 1 puede también proporcionar una mejora en las prestaciones de RAID 0 en aplicaciones de transferencia intensiva de datos con un alto porcentaje de lecturas. Se produce una mejora, si la aplicación puede dividir cada petición de lectura de forma que ambos miembros del disco participen.

## NIVEL 2 DE RAID

Los niveles 2 y 3 de RAID usan una técnica de acceso paralelo. En un conjunto de acceso paralelo, todos los discos miembros participan en la ejecución de cada petición de E/S. Normalmente, el giro de cada unidad individual está sincronizado de forma que cada cabeza de disco está en la misma posición en cada disco en un instante dado.

Como en los otros esquemas RAID, se usa la descomposición de datos en tiras. En el caso de RAID 2 y 3, las tiras son muy pequeñas; a menudo, tan pequeñas como un único byte o palabra. Con RAID 2, el código de corrección de errores se calcula a partir de los bits de cada disco, y los bits del código se almacenan en las correspondientes posiciones de bit en varios discos de paridad. Normalmente, se usa el código Hamming (ver Capítulo 4), que permite corregir errores en un bit y detectar errores en dos bits.

Aunque RAID 2 requiere menos discos que RAID 1, es todavía bastante caro. El número de discos redundantes es proporcional al logaritmo del número de discos de datos. En una sola lectura se accede a todos los discos simultáneamente. El controlador del conjunto proporciona los datos pedidos y el código de corrección de errores asociado. Si hay un error en un solo bit, el controlador lo puede reconocer y corregir instantáneamente, con lo que el tiempo de acceso a lectura no se ralentiza. En una escritura sencilla, la operación de escritura debe acceder a todos los discos de datos y de paridad.

RAID 2 debería ser solamente una elección efectiva en un entorno en el que haya muchos errores de disco. Si hay una alta seguridad en los discos individuales y en las unidades de disco, RAID 2 es excesivo y no se implementa.

### NIVEL 3 DE RAID

RAID 3 se organiza de manera similar a RAID 2. La diferencia es que RAID 3 requiere sólo un disco redundante, sin importar lo grande que sea el conjunto de discos. RAID 3 utiliza un acceso paralelo, con datos distribuidos en pequeñas tiras. En vez de un código de corrección de errores, se calcula un sencillo bit de paridad para el conjunto de bits individuales que están en la misma posición en todos los discos de datos.

#### Redundancia

En el caso de un fallo en una unidad, se accede a la unidad de paridad y se reconstruyen los datos desde el resto de los dispositivos. Una vez se sustituye la unidad que ha fallado, los datos que faltan se restauran en la nueva unidad y se reanuda la operación.

La reconstrucción de los datos es bastante sencilla. Consideremos un conjunto de cinco discos, de los que de X0 a X3 contienen datos, y X4 es el disco de paridad. La paridad para el *i*-ésimo bit se calcula de la siguiente forma:

$$X4(i) = X3(i) \oplus X2(i) \oplus X1(i) \oplus X0(i)$$

Supongamos que la unidad X1 ha fallado. Si sumamos  $X4(i) \oplus X1(i)$  a ambos miembros de la ecuación, tenemos que:

$$X1(i) = X4(i) \oplus X3(i) \oplus X2(i) \oplus X0(i)$$

Por lo tanto, se puede regenerar el contenido de cualquier tira de datos en X1 a partir del contenido de las correspondientes tiras del resto de los discos del conjunto. Este principio es válido para los niveles 3 a 6 de RAID.

Caso de que un disco falle, todos los datos estarán todavía disponibles en lo que se denomina modo reducido. En este modo, para lecturas, los datos que faltan se recuperan «al vuelo» con la operación exclusive-or. Cuando se escriben datos en un conjunto RAID 3 reducido, se debe mantener la consistencia de la paridad para regeneraciones posteriores.

Volviendo al funcionamiento global, se requiere que el disco que ha fallado se reemplace y se regenere todo su contenido en el nuevo disco.

### Prestaciones

Puesto que los datos se dividen en tiras muy pequeñas, RAID 3 puede conseguir velocidades de transferencia de datos muy altas. Cualquier petición de E/S implicará una transferencia de datos paralela desde todos los discos de datos. Para grandes transferencias, la mejora de prestaciones es especialmente notable. Por otra parte, sólo se puede ejecutar a la vez una petición de E/S. Por tanto, en un entorno orientado a transacciones, el rendimiento sufre.

### NIVEL 4 DE RAID

Los niveles 4 y 5 de RAID usan una técnica de acceso independiente. En un conjunto de acceso independiente, cada disco opera independientemente, de forma que peticiones de E/S separadas se atienden en paralelo. Debido a esto, son más adecuados los conjuntos de acceso independiente para aplicaciones que requieren velocidades de petición de E/S altas, y son menos adecuados para aplicaciones que requieren velocidades altas de transferencia de datos.

Como en otros esquemas RAID, se usan tiras de datos. En el caso de RAID 4 y 5, las tiras son relativamente grandes. Con RAID 4 se calcula una tira de paridad, bit a bit, a partir de las correspondientes tiras de cada disco de datos, y los bits de paridad se almacenan en la correspondiente tira del disco de paridad.

RAID 4 lleva consigo una penalización en la escritura cuando se realiza una petición de escritura de E/S pequeña. Cada vez que se realiza una escritura, el software de gestión del conjunto debe actualizar, no sólo los datos del usuario, sino también los bits de paridad correspondientes. Consideremos un conjunto de cinco unidades en las que de X0 a X3 contienen datos y X4 es el disco de paridad. Supongamos que se realiza una escritura que implica sólo una tira del disco X1. Inicialmente, para cada bit  $i$ , tenemos la siguiente relación:

$$X4(i) = X3(i) \oplus X2(i) \oplus X1(i) \oplus X0(i)$$

Después de la actualización, indicamos con prima los bits que han sido alterados:

$$\begin{aligned} X4'(i) &= X3(i) \oplus X2(i) \oplus X1'(i) \oplus X0(i) \\ &= X3(i) \oplus X2(i) \oplus X1'(i) \oplus X0(i) \oplus X1(i) \oplus X1(i) \\ &= X4(i) \oplus X1(i) \oplus X1'(i) \end{aligned}$$

Para calcular la nueva paridad, el software de gestión del conjunto debe leer la antigua tira del usuario y la antigua tira de paridad. Entonces, se pueden actualizar estas dos tiras con nuevos datos y calcular la nueva paridad. Por tanto, cada escritura de una tira implica dos lecturas y dos escrituras.

En el caso de una escritura de E/S de mayor tamaño, que implique tiras en todas las unidades de disco, la paridad se puede obtener fácilmente con un cálculo, usando solamente los nuevos bits de datos. Por tanto, la unidad de paridad puede ser actualizada en paralelo con las unidades de datos, y no habrá lecturas o escrituras extra.

En cualquier caso, cada operación de escritura implica al disco de paridad que, por consiguiente, se convertirá en un cuello de botella.

## NIVEL 5 DE RAID

RAID 5 está organizado de manera similar a RAID 4. La diferencia es que RAID 5 distribuye las tiras de paridad a lo largo de todos los discos. Una distribución típica es un esquema cíclico, como se muestra en la Figura 5.6f. Para un conjunto de  $n$  discos, la tira de paridad está en diferentes discos para las primeras  $n$  tiras, y este patrón se repite.

La distribución de las tiras de paridad a lo largo de todas las unidades, evita el potencial cuello de botella de E/S encontrado en RAID 4.

## NIVEL 6 DE RAID

El nivel 6 de RAID se introdujo en un artículo de los investigadores de Berkeley [KATZ89]. En el esquema del nivel 6 de RAID, se hacen dos cálculos de paridad distintos, que se almacenan en bloques separados en distintos discos. Por tanto, un conjunto RAID 6 cuyos datos requieran  $N$  discos consta de  $N + 2$  discos.

La figura 5.6g ilustra este esquema. P y Q son dos algoritmos de comprobación de datos distintos. Uno de los dos calcula la exclusive-OR usada en los niveles de 4 y 5 de RAID, pero el otro es un algoritmo de comprobación de datos independiente. Esto hace posible la regeneración de los datos, incluso si dos de los discos que contienen los datos de los usuarios fallan.

La ventaja del RAID 6 es que proporciona una disponibilidad de los datos extremadamente alta. Tendrían que fallar tres discos en el intervalo MTTR (tiempo medio de reparación) para no poder disponer de los datos. Por otra parte, RAID 6 incurre en una penalización de escritura, ya que cada escritura afecta a dos bloques de paridad.

## 5.3. MEMORIA ÓPTICA

En 1983, se introdujo uno de los productos de consumo de más éxito de todos los tiempos: el disco compacto (CD, Compact Disk) digital de audio. El CD es un disco no borrable, que puede almacenar más de 60 minutos de información de audio en una cara. El gran éxito comercial del CD posibilitó el desarrollo de la tecnología de discos de memoria óptica de bajo coste, que revolucionó el almacenamiento de datos en un computador. Se han introducido una gran variedad de discos ópticos (Tabla 5.3). Vamos a ver cada uno de ellos brevemente.

### CD-ROM

Tanto el CD de audio como el CD-ROM (compact disk read-only memory, «memoria de disco compacto de solo lectura») comparten una tecnología similar. La principal diferencia es que los lectores de CD-ROM son más robustos y tienen dispositivos de corrección de errores para asegurar que los datos se transfieren correctamente del disco al computador. Ambos tipos de disco se hacen también de la misma forma. El disco se forma a partir de una resina, como un policloruro de vinilo, y se cubre con una superficie altamente reflectante, normalmente con aluminio. La información grabada digitalmente (ya sea música o datos del computador) se graba como una serie de hoyos microscópicos en la superficie reflectante. Esto se hace, en primer lugar, con un láser de alta intensidad, enfocado con precisión, para crear el disco patrón. El patrón se usa después, para hacer una matriz para estampar copias. La superficie con los hoyos de las copias se protege del polvo y rasguños con una capa final de laca transparente.

Tabla 5.3. Discos ópticos

**CD**

Disco compacto. Un disco no borrable que almacena información de audio digitalizada. El sistema estándar usa discos de 12 cm y puede grabar más de 60 minutos de tiempo de audición ininterrumpido.

**CD-ROM**

Disco compacto de memoria de sólo lectura. Un disco no borrable usado como memoria de datos de un computador. El sistema estándar usa discos de 12 cm y puede guardar más de 600 Mbytes.

**DVD**

Disco de video digital. Una tecnología para producir representación de información de video digitalizada y comprimida, así como grandes cantidades de otros datos digitales.

**WORM**

Una escritura, varias lecturas. Un disco en el que se puede escribir más fácilmente que en un CD-ROM, haciendo que los discos de una sola copia sean comercialmente factibles. Como en los CD-ROM, tras una operación de escritura, el disco es de sólo lectura. El tamaño más común es de 5,25 pulgadas, que puede guardar entre 200 y 800 Mbytes de datos.

**Disco óptico borrable**

Un disco que usa tecnología óptica, pero que se puede borrar y reescribir fácilmente. Hay discos de 3,25 y 5,25 pulgadas. La capacidad típica es de 650 Mbytes.

**Disco magnético-óptico**

Un disco que usa tecnología óptica para leer técnicas de grabación magnéticas con ayuda de focalización óptica. Se usan discos tanto de 3,25 como de 5,25 pulgadas. Son usuales capacidades de alrededor de 1 Gbyte.

La información del CD o CD-ROM se recupera con un láser de baja potencia situado en un lector o unidad de disco óptico. El láser pasa a través de la capa protectora transparente mientras un motor hace girar el disco sobre el láser. La intensidad de la luz reflejada cambia si se encuentra un hoyo. Un fotosensor detecta este cambio que es convertido en una señal digital.

Un hoyo cerca del centro del disco que rota, pasa por delante de un punto fijo (como un haz de láser) más despacio que un hoyo que esté en el exterior, y de alguna forma hay que compensar la variación de velocidad, de forma que el láser pueda leer todos los hoyos a la misma velocidad. Esto se puede hacer (como en los discos magnéticos) incrementando el espacio lineal entre bits de información grabados en los segmentos más externos del disco. La información se puede explorar a la misma velocidad, girando el disco a una velocidad fija, conocida como velocidad angular constante (CAV). La Figura 5.8a muestra la organización de un disco usando CAV. El disco se divide en una serie de sectores, como trozos de tarta, y en una serie de pistas concéntricas. La ventaja de usar CAV es que los bloques individuales de datos se pueden direccionar directamente a partir de la pista y el sector. Para mover la cabeza de su posición habitual a una dirección específica, se hace sólo un pequeño movimiento de la cabeza a la pista especificada, y se espera un poco a que el sector se posicione bajo la cabeza. La desventaja de CAV es que la cantidad de datos que se pueden almacenar en las pistas exteriores, más grandes, es la misma que la que se puede almacenar en las pistas interiores, más pequeñas.

Puesto que poner menos información en la parte más exterior del disco malgasta espacio, el método CAV no se usa en CD y CD-ROM. En cambio, la información se empaquetá con densidad uniforme a lo largo del disco en segmentos del mismo tamaño, y se explora a la misma velocidad, rotando el disco a una velocidad variable. El láser, por tanto, lee los hoyos a una velocidad lineal constante (CLV). El disco rota más despacio en accesos cercanos al borde exterior que en accesos cerca del centro. Por tanto, la capacidad de una pista y el tiempo de rotación se incrementan en posiciones de la pista cercanas al borde exterior del disco.

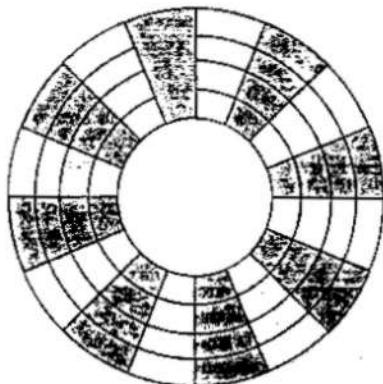
Se producen CD-ROM de varias densidades. El ejemplo más típico es el espaciado entre pistas de  $1.6 \mu\text{m}$  ( $1.6 \times 10^{-6} \text{ m}$ ). El ancho grabable de un CD-ROM a lo largo de su radio, es de 32.55 mm, así que el número total de pistas aparentes es de 32.550  $\mu\text{m}$ , dividido por el espaciado entre pistas, o sea 20.344 pistas. De hecho, hay una única pista en espiral, y podemos calcular la longitud de una pista multiplicando la circunferencia media por el número de vueltas en la espiral; este cálculo debe ser de unos 5.27 Km aproximadamente. La velocidad lineal constante del CD-ROM es de 1,2 m/s, lo que da un tiempo total de 4.391 segundos, o sea 73.2 minutos, lo que está cercano al estándar máximo del tiempo de lectura de un disco compacto de audio. Puesto que los datos salen del disco a 176.4 Kbytes/segundo, la capacidad de almacenamiento de un CD-ROM es 774.57 MBytes. Esto equivale a más de 550 discos de 3.25 pulgadas.

Los datos de un CD-ROM se organizan en una secuencia de bloques. En la Figura 5.8 se muestra un formato típico de un bloque. Este consta de los siguientes campos:

- **Sincronización:** El campo de sincronización identifica el principio de un bloque. Consta de un byte de 0s, 10 bytes de 1s, y un byte de 0s.
- **Cabecera:** La cabecera contiene la dirección del bloque y el byte de modo. El modo 0 especifica un campo de datos en blanco; el modo 1 especifica el uso de un código de corrección de errores y 2.048 bytes de datos; el modo 2 especifica 2.336 bytes de datos del usuario sin código de corrección de errores.
- **Datos:** Datos del usuario.
- **Auxiliar:** Datos del usuario adicionales, en modo 2. En modo 1, es un código de corrección de errores de 288 bytes.

En la Figura 5.8b se indica la organización usada para CD y CD-ROM. Como mencionamos, los datos se sitúan secuencialmente a lo largo de la pista en espiral. Con el uso de CLV, el acceso aleatorio es más difícil. Situarse en una dirección específica implica el movimiento de la cabeza al área general, ajustar la velocidad de rotación y leer la dirección y, entonces, se hacen pequeños ajustes para encontrar y acceder al sector específico.

Los CD-ROM son apropiados para la distribución de grandes cantidades de datos a un gran número de usuarios. Debido al gasto del proceso inicial de escritura, no es adecuado



(a) Velocidad angular constante



(b) Velocidad lineal constante

Figura 5.8. Comparación entre los métodos de organización.

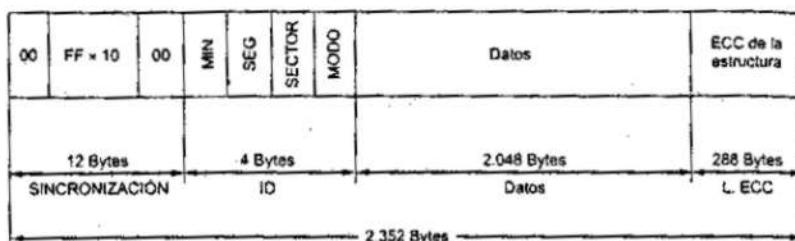


Figura 5.9. Formato de los bloques de un CD-ROM.

para aplicaciones individuales. Comparado con los discos magnéticos tradicionales, el CD-ROM tiene tres ventajas principales:

- La capacidad de almacenamiento de información es mucho mayor en los CD-ROM.
- El CD-ROM, junto con la información almacenada en él, se puede replicar en grandes cantidades de forma barata (a diferencia de los discos magnéticos). Las bases de datos en un disco magnético se reproducen copiando uno a uno, usando dos unidades de disco.
- El CD-ROM es extraíble, permitiendo usar el disco mismo como memoria de archivo. La mayoría de los discos magnéticos no son extraíbles. La información que contiene tiene que copiarse en una cinta antes de que se pueda usar la unidad de disco/disco para almacenar nueva información.

Las desventajas del CD-ROM son:

- Es sólo de lectura y no se puede actualizar.
- El tiempo de acceso es mayor que el de una unidad de disco magnético, tanto como medio segundo.

## WORM

Se ha desarrollado el CD de una-escritura-varias-lecturas, para adaptarse a aplicaciones en las que sólo se necesitan unas pocas copias de un conjunto de datos. Para hacer un WORM, se construye un disco preparado para poder ser escrito una vez con un haz láser de intensidad modesta. De esta forma, con algún controlador de disco especial, más caro que para CD-ROM, el cliente puede escribir una vez, además de leer el disco. Para proporcionar un acceso más rápido, el WORM usa velocidad angular constante, sacrificando parte de su capacidad.

Una técnica típica para fabricar el disco es usar un láser de alta potencia para producir una serie de ampollas en el disco. Cuando el medio preformatado se sitúa en una unidad de WORM, un láser de baja potencia puede producir calor suficiente para reventar las ampollas pregrabadas. Durante la operación de lectura, el láser de la unidad WORM ilumina la superficie del disco. Como las ampollas reventadas proporcionan mayor contraste que el área de alrededor, una electrónica sencilla lo reconoce fácilmente.

El disco óptico WORM es atractivo para almacenar archivos de documentos y ficheros. Proporciona una grabación permanente de grandes cantidades de datos del usuario.

## DISCO ÓPTICO BORRABLE

El desarrollo más reciente en discos ópticos es el disco óptico borrable. Este disco se puede escribir y reescribir repetidamente, como cualquier disco magnético. Aunque se han probado muchas aproximaciones, la única tecnología que los ha hecho comercialmente factibles es el sistema magnético-óptico. En este sistema, se utiliza la energía de un haz de láser, junto con un campo magnético, para grabar y borrar información, invirtiendo los polos magnéticos en una pequeña área del disco cubierta con un material magnético. El haz láser calienta un determinado punto del medio, y un campo magnético puede cambiar la orientación magnética (polo) de este punto, mientras se eleva su temperatura. Como el proceso de polarización no causa un cambio físico en el disco, el proceso se puede repetir varias veces. Para leer, el polo de magnetización se puede detectar con un haz láser polarizado. La luz polarizada reflejada por un punto particular cambiará su grado de rotación dependiendo de la orientación, norte o sur, del campo magnético.

El disco óptico borrable tiene la ventaja obvia, sobre el CD-ROM y el WORM, de que puede ser reescrito, y por tanto usado, como una verdadera memoria secundaria. Por tanto, compite con el disco magnético. Las principales ventajas de los discos ópticos borrables, comparadas con los discos magnéticos son:

- **Alta capacidad:** Un disco óptico de 5,25 pulgadas puede almacenar alrededor de 650 Mbytes de datos. Los discos Winchester más avanzados puede memorizar menos de la mitad de los datos anteriores.
- **Intercambiabilidad:** Los discos ópticos se pueden extraer de la unidad de disco.
- **Seguridad:** Las tolerancias de construcción para los discos ópticos son mucho menos severas que para los discos magnéticos de alta capacidad. Por tanto, muestra mayor seguridad y vida.

Como los WORM, los discos ópticos borrables usan una velocidad angular constante.

## DISCO VÍDEO DIGITAL

Con la gran capacidad de almacenamiento de los discos video digitales (DVD, Digital Video Disk), la industria de la electrónica ha encontrado por fin un sustituto razonable de las cintas VHS de vídeo analógicas. El DVD sustituirá a las cintas de vídeo usadas en los reproductores de vídeo (VCR) y, lo que es más importante para este texto, sustituirá al CD-ROM en los PC y servidores. El DVD lleva al vídeo a la edad digital. Proporciona películas con una calidad de imagen impresionante, y se puede acceder a ellos aleatoriamente como en los CD de audio, que pueden también leer los DVD. En un disco se puede grabar un gran volumen de datos, en la actualidad siete veces más que en un CD-ROM. Con esta gran capacidad de almacenamiento y alta calidad de los DVD, los juegos para PC serán más reales, y el software educativo incorporará más vídeo. Como consecuencia de estos desarrollos habrá un nuevo pico en el tráfico en Internet y en las intranets corporativas, ya que este material se incorporará a los sitios Web.

He aquí algunas claves que destacar de los DVD que los diferencia de los CD-ROM:

- Un DVD estándar almacena 4,7 Gbytes por capa, y los de doble capa y una cara almacenan 8,5 Gbytes.
- El DVD utiliza un formato de compresión conocido como MPEG para imágenes de pantalla completa de alta calidad.
- Un DVD de una capa puede almacenar una película de dos horas y media, mientras que uno dual puede contener una película de más de cuatro horas.

## DISCOS MAGNÉTICO-ÓPTICOS

Una unidad de disco magnético-óptica (MO, Magneto Optical) usa un láser óptico para aumentar las posibilidades de una unidad de disco magnético convencional. La tecnología de grabación es fundamentalmente magnética. Sin embargo, se usa un láser óptico para focalizar la cabeza grabadora magnética, para poder conseguir una gran capacidad. En este esquema, el disco se recubre con un material, cuya polaridad se puede alterar sólo a altas temperaturas. La información se escribe en el disco, usando un láser para calentar un pequeño punto de su superficie, y luego aplicando un campo magnético. Cuando el punto se enfria, adopta la polaridad norte-sur. Como el proceso de polarización no causa un cambio físico en el disco, este proceso se puede repetir varias veces.

La operación de lectura es puramente óptica. La dirección de la magnetización se puede detectar con un haz de láser polarizado (con una potencia inferior a la de escritura). La luz polarizada reflejada por un punto en concreto cambiará su grado de rotación dependiendo de la orientación del campo magnético.

La principal ventaja de los MO frente a los puramente ópticos CD es la longevidad de los discos. Repetidas escrituras de los datos en un disco óptico produce una degradación gradual del medio. En las unidades MO no se produce esta degradación y, por tanto, sigue siendo utilizable durante más operaciones de escritura. Otra ventaja de la tecnología MO es que ofrece un coste por megabyte considerablemente inferior que en el almacenamiento magnético.

## 5.4. CINTA MAGNÉTICA

Los sistemas de cinta usan las mismas técnicas de lectura y grabación que los discos. El medio es una cinta de plástico («mylar») flexible, cubierta por un óxido magnético. La cinta y la unidad de cinta son análogas a una cinta de grabación doméstica.

El medio de la cinta se estructura en un pequeño número de pistas paralelas. Los primeros sistemas de cintas usaban 9 pistas. Esto hace posible almacenar datos de un byte en un instante dado, con un bit de paridad adicional, en la novena pista. Los nuevos sistemas de cintas usan 18 o 36 pistas, que corresponden a una palabra o a una doble palabra digital. Como con el disco, los datos se leen y escriben en bloques contiguos, llamados *registros físicos* de cinta. Los bloques de la cinta están separados por bandas vacías, llamadas bandas *inter-registros*. En la Figura 5.10 se muestra la estructura de una cinta de 9 pistas. Como en el disco, la cinta tiene un formato para facilitar la localización de los registros físicos.

Una unidad de cinta es un dispositivo de *acceso secuencial*. Si la cabeza de la cinta se posiciona en el registro 1, entonces, para leer el registro  $N$ , es necesario leer los registros físicos del 1 al  $N - 1$ , uno a uno. Si la cabeza está actualmente situada más allá del registro

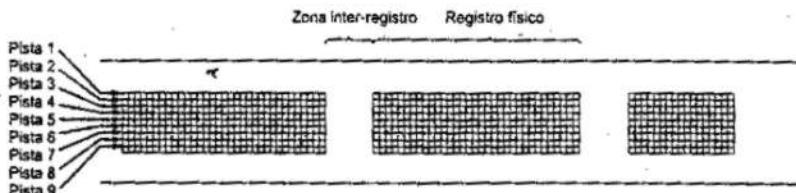


Figura 5.10. Formato de un disco magnético de nueve pistas.

deseado, es necesario rebobinar la cinta un cierto trecho y empezar a leer hacia adelante. A diferencia del disco, la cinta está en movimiento solamente durante las operaciones de lectura o escritura.

En contraste con las cintas, a la unidad de disco se le llama dispositivo de *acceso directo*. Una unidad de disco no necesita leer todos los sectores de un disco secuencialmente para llegar al sector deseado. Sólo debe esperar a los sectores dentro de una pista, y puede acceder sucesivamente a cualquier pista.

Las cintas magnéticas fueron el primer tipo de memorias secundarias. Se usan todavía ampliamente, como los miembros de la jerarquía de memoria de menor coste y de menor velocidad.

### LECTURAS Y SITIOS WEB RECOMENDADOS

La referencia [MEE96a] ofrece un buen resumen de la tecnología de grabación subyacente de los discos y cintas. [MEE96b] se centra en las técnicas de almacenamiento de datos en discos y cintas.

Un excelente estudio sobre la tecnología RAID, escrita por los inventores del concepto RAID, está en [CHEN94]. Una discusión más detallada está publicada en RAID Advisory Board, una asociación de suministradores y usuarios de productos relacionados con RAID [MASS97]. Un buen artículo reciente es [FRIE96].

[MARC90] da una excelente visión del campo de las memorias ópticas. Un buen examen de las tecnologías subyacentes de grabación y de lectura es [MANS97].

Finalmente, [ROSC97] proporciona una visión comprensible de todos los tipos de memorias externas, con una modesta cantidad de detalles técnicos de cada uno.

CHEN94 Chen, P.; Lee, E.; Gibson, G.; Katz, R.; y Patterson, D. «RAID: High Performance Reliable Secondary Storage.» *ACM Computing Surveys*. Junio, 1994.

FRIE96 Friedman, M. «RAID Keeps Going and Going and...» *IEEE Spectrum*. April, 1996.

MANS97 Mansuripur, M., y Sincerbox, G. «Principles and Techniques of Optical Data Storage.» *Proceedings of the IEEE*. November, 1997.

MARC90 Marchant, A. *Optical Recording*. Reading, MA: Addison-Wesley, 1990.

MASS97 Massiglia, P. (editor). *The RAID book: A Storage System Technology*. St. Peter, MN: The RAID Advisory Board, 1997.

MEE96a Mee, C., y Daniel, E., eds. *Magnetic Recording Technology*. New York: McGraw-Hill, 1996.

MEE96b Mee, C., y Daniel, E., eds. *Magnetic Storage Handbook*. New York: McGraw-Hill, 1996.

ROSC97 Rosch, W. *The Winn L. Rosch Hardware Bible*. Indianapolis, IN: Sams, 1997.



#### SITIO WEB RECOMENDADO:

- Tarjeta de consulta RAID: industria RAID

**5.6. PROBLEMAS**

- 5.1. Se define lo siguiente para un disco:

$t_s$  = tiempo de búsqueda; tiempo medio para posicionar la cabeza sobre una pista.

$r$  = velocidad de rotación del disco en revoluciones por segundo.

$n$  = número de bits por sector.

$N$  = capacidad de una pista en bits.

$t_A$  = tiempo de acceso a un sector.

Desarrollar una fórmula para  $t_A$  en función del resto de los parámetros.

- 5.2. Sea una configuración formada por 10 unidades RAID. Rellenar la siguiente matriz, que compara los distintos niveles RAID:

| Nivel RAID | Densidad de almacenamiento | Prestaciones del ancho de banda | Prestaciones de transacción |
|------------|----------------------------|---------------------------------|-----------------------------|
| 0          | 1                          |                                 | 1                           |
| 1          |                            |                                 |                             |
| 2          |                            |                                 |                             |
| 3          |                            | 1                               |                             |
| 4          |                            |                                 |                             |
| 5          |                            |                                 |                             |

Cada parámetro se normaliza al nivel RAID que ofrece las mejores prestaciones. La densidad de almacenamiento se refiere a la fracción de memoria de disco disponible para datos del usuario. Las prestaciones de ancho de banda reflejan la rapidez a la que se pueden transferir datos fuera de un conjunto. Las prestaciones de transacción miden cuántas operaciones de E/S por segundo puede realizar un conjunto.

- 5.3. Debería quedar claro que la organización en tiras de los datos de un disco puede mejorar la velocidad de transferencia de datos cuando el tamaño de una tira es pequeño comparado con el tamaño de la petición de E/S. Debería quedar también claro, que RAID 0 proporciona mejoras en las prestaciones con respecto a un único y gran disco, ya que se pueden manejar en paralelo varias peticiones de E/S. En este último caso, ¿es necesaria la partición en tiras del disco? Es decir, ¿las tiras mejoran las prestaciones en frecuencia de peticiones de E/S, comparado con un conjunto de discos sin tiras?
- 5.4. ¿Cuál es la velocidad de transferencia de una cinta magnética de 9 pistas cuya velocidad es 120 pulgadas por segundo y cuya densidad es 1.600 bits lineales por pulgada?
- 5.5. Supongamos una cinta de carrete de 2.400 pies; un interbloque de 0,6 pulgadas, donde la cinta se para a mitad de camino entre dos lecturas, la velocidad de la cinta se incrementa/decremente linealmente y las otras características de la cinta son las mismas que las del Problema 5.4. Los datos se organizan en la cinta en registros físicos, donde cada

registro físico contiene un número fijo de unidades definidas por el usuario, llamados «registros lógicos».

- a) ¿Cuánto tiempo se tardará en leer una cinta completa, si un registro físico se compone de 10 registros lógicos de 120 bytes?
  - b) ¿Cuánto tiempo se tardará en leer una cinta completa, si un registro físico son 30 registros lógicos de 120 bytes?
  - c) ¿Cuántos registros lógicos podrá contener una cinta en cada uno de los casos anteriores?
  - d) ¿Cuál es la velocidad de transferencia total efectiva para cada uno de los casos anteriores?
  - e) ¿Cuál es la capacidad de la cinta?
- 5.6. Calcular el espacio del disco (en sectores, pistas y superficies) que se necesitaría para almacenar los registros lógicos descritos en el Problema 5.5a, si el disco tiene sectores fijos de 512 bytes/sector, con 96 sectores/pista, 110 pistas por superficie y 8 superficies utilizables. Ignorar cualquier registro(s) de cabecera del fichero e índices de pistas, y asumir que los registros no pueden abarcar más de un sector.

.....

## CAPÍTULO 6

# Entrada/salida

### 6.1. Dispositivos externos

Teclado/monitor

Controlador de disco («Disk Drive»)

### 6.2. Módulos de E/S

Funciones de un módulo

Estructura de un módulo de E/S

### 6.3. E/S programada

Résumen

Órdenes de E/S

Instrucciones de E/S

### 6.4. E/S mediante interrupciones

Procesamiento de la interrupción

Cuestiones de diseño

Controlador de interrupciones Intel 82C59A

La interfaz programable de periféricos Intel 82C55A

### 6.5. Acceso directo a memoria

Inconvenientes de la E/S programada y con interrupciones

Funcionamiento del DMA

### 6.6. Canales y procesadores de E/S

La evolución del funcionamiento de las E/S

Características de los canales de E/S

### 6.7. La interfaz externa: SCSI y FireWire

Tipos de interfaces

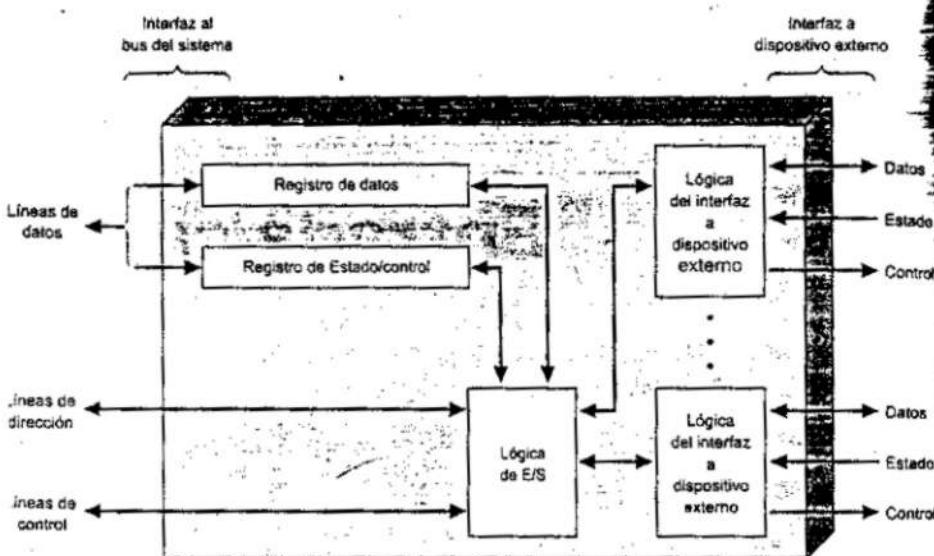
Configuraciones punto-a-punto y multipunto

Interfaz SCSI (Small Computer System Interface)

Bus serie FireWire

### 6.8. Lecturas y sitios Web recomendados

### 6.9. Problemas



- La arquitectura de E/S del computador es su interfaz con el exterior. Esta arquitectura se diseña de manera que permita una forma sistemática de controlar las interacciones con el mundo exterior y proporcione al sistema operativo la información que necesita para gestionar eficazmente la actividad de E/S.
- Hay tres técnicas de E/S principales: E/S programada, en la que la E/S se produce bajo el control directo y continuo del programa que solicita la operación de E/S; E/S mediante interrupciones, en la que el programa genera una orden de E/S y después continúa ejecutándose hasta que el hardware de E/S lo interrumpe para indicar que la operación de E/S ha concluido; y acceso directo a memoria (DMA, Direct Memory Access), en el que un procesador de E/S específico toma el control de la operación de E/S para transferir un gran bloque de datos.
- Dos ejemplos importantes de interfaces de E/S externas son SCSI y FireWire. SCSI es una interfaz paralela para dispositivos externos, mientras que la nueva FireWire es una interfaz en serie de alta velocidad.

Junto con el procesador y el conjunto de módulos de memoria, el tercer elemento clave de un computador es un conjunto de módulos de E/S. Cada módulo se conecta al bus del sistema o a un commutador central, y controla uno o más dispositivos periféricos. Un módulo de E/S no es únicamente un conector mecánico que permite encajar el dispositivo al bus del sistema, sino que, además, está dotado de cierta «inteligencia», es decir, contiene la lógica necesaria para permitir la comunicación entre el periférico y el bus.

El lector podría preguntarse por qué los periféricos no se conectan directamente al bus del sistema. Las razones son:

- Hay una amplia variedad de periféricos con formas de funcionamiento diferentes. Podría ser imposible incorporar la lógica necesaria dentro del procesador para controlar tal diversidad de dispositivos.
- A menudo, la velocidad de transferencia de datos de los periféricos es mucho menor que la de la memoria o el procesador. Así, no es práctico utilizar un bus del sistema de alta velocidad para comunicarse directamente con un periférico.
- Con frecuencia, los periféricos utilizan datos con formatos y tamaños de palabra diferentes de los del computador a los que se conectan.

En consecuencia, se necesita un módulo de E/S. Este módulo tiene dos funciones principales (Figura 6.1):

- Realizar la interfaz entre el procesador y la memoria a través del bus del sistema o un commutador central.
- Realizar la interfaz entre uno o más dispositivos periféricos mediante enlaces de datos específicos.

Comenzamos este capítulo con una breve discusión acerca de los dispositivos externos, seguida de una revisión de la estructura y el funcionamiento de un módulo de E/S. Después, consideraremos las diferentes formas de realizar la función de E/S en cooperación con el procesador y la memoria: la interfaz de E/S interna. Por último, se examina la interfaz de E/S externa, entre el módulo de E/S y el mundo exterior.

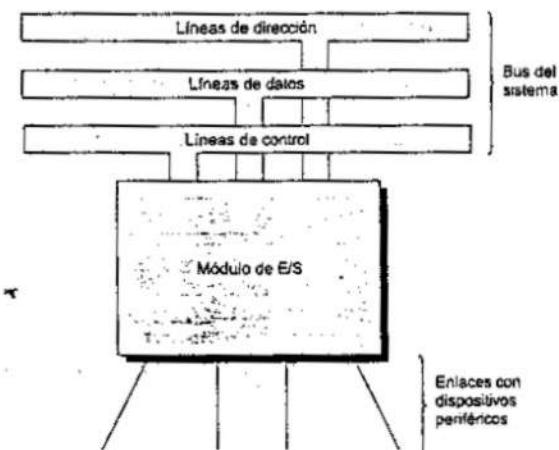


Figura 6.1. Modelo genérico de un módulo de E/S.

**6.1. DISPOSITIVOS EXTERNOS**

Las operaciones de E/S se realizan a través de una amplia gama de dispositivos, que proporcionan una forma de intercambiar datos entre el exterior y el computador. Un dispositivo externo se conecta al computador mediante un enlace a un módulo de E/S (Figura 6.1). El enlace se utiliza para intercambiar señales de control, estado y datos entre el módulo de E/S y el dispositivo externo. Un dispositivo externo conectado a un módulo de E/S frecuentemente se denomina *dispositivo periférico* o, simplemente, *periférico*.

En sentido amplio, los dispositivos externos se pueden clasificar en tres categorías:

- De interacción con humanos: permiten la comunicación con el usuario del computador.
- De interacción con máquinas: permiten la comunicación con elementos del equipo.
- De comunicación: permiten la comunicación con dispositivos remotos.

Ejemplos de dispositivos de interacción con humanos son los terminales de video (VDT, Video Display Terminals) y las impresoras. Ejemplos de dispositivos de interacción con máquinas son los discos magnéticos y los sistemas de cinta, y los sensores y actuadores, tales como los que se usan en aplicaciones de robótica. Obsérvese que los discos y los sistemas de cinta se están considerando como dispositivos de E/S en este capítulo, mientras que en el Capítulo 5 los consideramos como dispositivos de memoria. Desde el punto de vista de su función, estos dispositivos son parte de la jerarquía de memoria y, propiamente, su uso se discute en el Capítulo 5. Desde un punto de vista estructural, estos dispositivos se controlan mediante módulos de E/S y, consecuentemente, deben ser considerados en este capítulo.

Los dispositivos de comunicación permiten que el computador intercambie datos con un dispositivo remoto, que puede ser un dispositivo de interacción con humanos, como por ejemplo un terminal, un dispositivo de interacción con máquinas o, incluso, otro computador.

En términos muy generales, la forma de un dispositivo externo se indica en la Figura 6.2. La conexión con el módulo de E/S se realiza a través de señales de control, estado, y datos. Los datos se intercambian en forma de un conjunto de bits que son enviados a, o recibidos

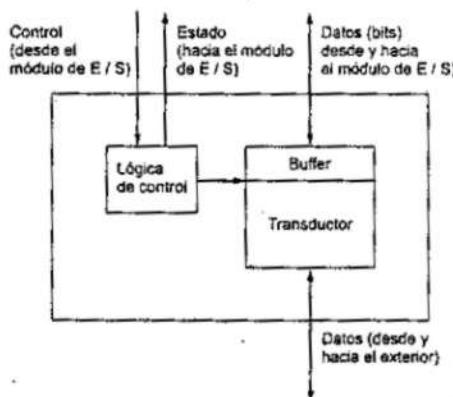


Figura 6.2. Dispositivo externo.

desde el módulo de E/S. Las señales de control determinan la función que debe realizar el dispositivo, tal como enviar datos al módulo de E/S, ENTRADA («INPUT») o LECTURA («READ»), aceptar datos desde el módulo de E/S, SALIDA («OUTPUT») o ESCRITURA («WRITE»), indicar el estado, o realizar alguna función de control particular del dispositivo (por ejemplo, situar una cabeza del disco). Las señales de estado indican el estado del dispositivo. Como ejemplos están la señal LISTO/NO LISTO («READY/NOT-READY»), que indica si el dispositivo está preparado para la transferencia de datos.

La lógica de control asociada al dispositivo controla su operación en respuesta a las indicaciones del módulo de E/S. El transductor convierte las señales eléctricas asociadas al dato a otra forma de energía en el caso de una salida, y viceversa en el caso de una entrada. Usualmente, existe un buffer asociado al transductor para almacenar temporalmente el dato que se está transfiriendo entre el módulo de E/S y el exterior; es común un tamaño de buffer de 8 a 16 bits.

La interfaz entre el módulo de E/S y el dispositivo externo se examinará en la Sección 6.7. La interfaz entre el dispositivo externo y el entorno está fuera del enfoque de este libro, pero se darán algunos ejemplos breves.

## TECLADO/MONITOR

La forma más común de interacción computador/usuario se produce a través de la combinación teclado/monitor. El usuario proporciona la entrada a través del teclado. A continuación esta entrada se transmite al computador y puede verse en el monitor. Además, el monitor muestra los datos que proporciona el computador.

La unidad básica de intercambio es el carácter. Asociado con cada carácter hay un código, usualmente de 7 u 8 bits de longitud. El código más comúnmente usado es un código de 7 bits, conocido como ASCII (American Standard Code for Information Interchange) en los Estados Unidos de América, e internacionalmente como Alfabeto de Referencia Internacional ITU-T. Cada carácter de este código se representa mediante un único número binario de 7 bits; en consecuencia, se pueden representar 128 caracteres<sup>1</sup>. La Tabla 6.1 enumera los valores del código. En la tabla, los bits de cada carácter se designan desde b<sub>7</sub>, que es el bit más significativo, a b<sub>1</sub>, el menos significativo. Los caracteres son de dos tipos: imprimibles y de control. Los caracteres imprimibles son alfábéticos, numéricos, y especiales (estos últimos pueden imprimirse en papel o visualizarse en una pantalla). Por ejemplo, la representación binaria del carácter «K» es 1001011. Algunos de los caracteres de control se utilizan para controlar la impresora o la visualización de los caracteres; un ejemplo es el retorno de carro. Otros caracteres de control están relacionados con los procedimientos de comunicación.

Para la entrada desde teclado, cuando el usuario pulsa una tecla se genera una señal eléctrica, interpretada por el transductor del teclado, que la traduce al patrón binario del correspondiente código ASCII. Entonces, este patrón binario se transmite al módulo de E/S del computador. En el computador el texto se puede almacenar utilizando el mismo código ASCII. En la salida, los códigos ASCII se transmiten al dispositivo externo desde el módulo de E/S. El transductor del dispositivo interpreta este código y envía las señales eléctricas precisas para que muestre en pantalla el carácter indicado o realice la función de control solicitada.

<sup>1</sup> Los caracteres ASCII casi siempre se almacenan y transmiten utilizando 8 bits por carácter (un bloque de 8 bits se llama octeto o byte). El octeto bit es un bit de paridad para la detección de errores. El bit de paridad es el que se encuentra en la posición más significativa, y, por consiguiente, se designa como «b<sub>8</sub>».

Tabla 6.1. Código ASCII (American Standard Code for Information Interchange)

Posición del bit

| b <sub>7</sub> | b <sub>6</sub> | b <sub>5</sub> | b <sub>4</sub> | b <sub>3</sub> | b <sub>2</sub> | b <sub>1</sub> | 0  | 0 | 0 | 0 | 1 | 1   | 1 | 1 |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----|---|---|---|---|-----|---|---|
|                |                |                |                |                |                |                | 0  | 0 | 1 | 0 | 1 | 0   | 1 | 0 |
|                |                |                |                |                |                |                | 0  | 1 | 0 | 1 | 0 | 1   | 0 | 1 |
| 0              | 0              | 0              | 0              | 0              | NUL            | DLE            | SP | 0 | @ | P | ' | P   |   |   |
| 0              | 0              | 0              | 0              | 1              | SOH            | DC1            | !  | 1 | A | Q | a | q   |   |   |
| 0              | 0              | 1              | 0              | 0              | STX            | DC2            | "  | 2 | B | R | b | r   |   |   |
| 0              | 0              | 1              | 1              | 0              | ETX            | DC3            | *  | 3 | C | S | c | s   |   |   |
| 0              | 1              | 0              | 0              | 0              | EOT            | DC4            | \$ | 4 | D | T | d | t   |   |   |
| 0              | -1             | 0              | 1              | 0              | ENQ            | NAK            | %  | 5 | E | U | e | u   |   |   |
| 0              | 1              | 1              | 0              | 0              | ACK            | SYN            | &  | 6 | F | V | f | v   |   |   |
| 0              | 1              | 1              | 1              | 0              | BEL            | ETB            | '  | 7 | G | W | g | w   |   |   |
| 1              | 0              | 0              | 0              | 0              | BS             | CAN            | {  | 8 | H | X | h | x   |   |   |
| 1              | 0              | 0              | 1              | 0              | HT             | EM             | )  | 9 | I | Y | i | y   |   |   |
| 1              | 0              | 1              | 0              | 0              | LF             | SUB            | *  | : | J | Z | j | z   |   |   |
| 1              | 0              | 1              | 1              | 0              | VT             | ESC            | +  | : | K | I | k | i   |   |   |
| 1              | 1              | 0              | 0              | 0              | FF             | FS             | .  | < | L | \ | l | l   |   |   |
| 1              | 1              | 0              | 1              | 0              | CR             | GS             | -  | = | M | I | m | i   |   |   |
| 1              | 1              | 1              | 0              | 0              | SO             | RS             | .  | > | N | ^ | n | ~   |   |   |
| 1              | 1              | 1              | 1              | 1              | SI             | US             | /  | ? | O | - | o | DEL |   |   |

Esta es la versión para EE.UU. del Alfabeto de Referencia Internacional ITU-T (T.50). Los caracteres de control se explican en la Tabla 6.2.

## CONTROLADOR DE DISCO (DISK DRIVE)

Un controlador de disco contiene la electrónica necesaria para intercambiar señales de datos, control y estado con un módulo de E/S, más la electrónica para controlar el mecanismo de lectura/escritura del disco. En un disco de cabeza fija, el transductor hace la conversión entre los patrones magnéticos de la superficie del disco en movimiento y los bits del buffer del dispositivo (Figura 6.2). Un disco de cabeza móvil debe además ser capaz de mover radialmente el brazo del disco hacia dentro y hacia fuera sobre la superficie del disco.

Tabla 6.2. Caracteres de control ASCII

|                                                                                                                                                            |                                                                                                                                                                                     | Control de formato                                                                                                                                                              |                                                                                                                                                                                                                                                                                                      |
|------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| BS                                                                                                                                                         | (Backspace, «espacio atrás»): indica movimiento de un espacio hacia atrás del mecanismo de impresión o del cursor de la pantalla.                                                   | VT                                                                                                                                                                              | (Vertical Tab, «tabulación vertical»): indica movimiento del mecanismo de impresión o del cursor de pantalla hasta la siguiente de una serie de líneas impresas.                                                                                                                                     |
| HT                                                                                                                                                         | (Horizontal Tab, «tabulación horizontal»): indica movimiento del mecanismo de impresión o del cursor de pantalla hasta el siguiente «tabulador» asignado o a la posición de parada. | FF                                                                                                                                                                              | (Form Feed, «avance de páginas»): indica movimiento del mecanismo de impresión o del cursor de pantalla hasta el comienzo de la siguiente página o imagen de pantalla.                                                                                                                               |
| LF                                                                                                                                                         | (Line Feed, «avance de línea»): indica movimiento del mecanismo de impresión o del cursor de pantalla hasta el comienzo de la línea siguiente.                                      | CR                                                                                                                                                                              | (Carriage Return, «retorno de carro»): indica movimiento del mecanismo de impresión o del cursor de pantalla hasta el comienzo de la línea en curso.                                                                                                                                                 |
| Control de transmisión                                                                                                                                     |                                                                                                                                                                                     |                                                                                                                                                                                 |                                                                                                                                                                                                                                                                                                      |
| SOH                                                                                                                                                        | (Start of Heading, «comienzo de cabecera»): usado para indicar el comienzo de una cabecera que puede contener información de dirección o enrutamiento.                              |                                                                                                                                                                                 | respuesta positiva a los mensajes de sondeo («polling»).                                                                                                                                                                                                                                             |
| STX                                                                                                                                                        | (Start of Text, «comienzo de texto»): usado para indicar el comienzo de texto, y también para indicar el final de la cabecera.                                                      | NAK                                                                                                                                                                             | (Negative Acknowledgement, «reconocimiento negativo»): carácter transmitido por un dispositivo receptor como respuesta negativa al emisor. Se utiliza como respuesta negativa a los mensajes de sondeo.                                                                                              |
| ETX                                                                                                                                                        | (End of Text, «final de texto»): usado como fin del texto que se inició con STX.                                                                                                    | SYN                                                                                                                                                                             | (Synchronous/Idle, «síncrono/pagado»): utilizado por un sistema con transmisión síncrona para conseguir la sincronización. Cuando no se envía ningún dato, un sistema con transmisión síncrona envía caracteres SYN continuamente.                                                                   |
| EOT                                                                                                                                                        | (End of Transmission, «final de transmisión»): indica el final de la transmisión, que puede incluir uno o más textos con sus cabeceras.                                             | ETB                                                                                                                                                                             | (End of Transmission Block, «final del bloque transmitido»): se utiliza en el contexto de las comunicaciones para indicar el final de un bloque de datos. Permite organizar los datos en bloques, donde la estructura del bloque no está necesariamente relacionada con el formato de procesamiento. |
| ENQ                                                                                                                                                        | (Enquiry, «interrogación»): petición de respuesta desde una estación remota. Se puede utilizar como una petición de identificación («WHO ARE YOU?») para la estación.               |                                                                                                                                                                                 |                                                                                                                                                                                                                                                                                                      |
| ACK                                                                                                                                                        | (Acknowledge, «reconocimiento»): carácter transmitido por un dispositivo receptor como respuesta afirmativa al emisor. Se utiliza como                                              |                                                                                                                                                                                 |                                                                                                                                                                                                                                                                                                      |
| Separadores de información                                                                                                                                 |                                                                                                                                                                                     |                                                                                                                                                                                 |                                                                                                                                                                                                                                                                                                      |
| Los separadores de información se utilizan de manera opcional, si bien están ordenados jerárquicamente desde el FS (el más inclusivo) hasta US (el menos). | GS                                                                                                                                                                                  | (Group Separator, «separador de grupo»).                                                                                                                                        |                                                                                                                                                                                                                                                                                                      |
| FS (File Separator, «separador de ficheros»).                                                                                                              | RS                                                                                                                                                                                  | (Record Separator, «separador de registro»).                                                                                                                                    |                                                                                                                                                                                                                                                                                                      |
|                                                                                                                                                            | US                                                                                                                                                                                  | (United Separator, «separador unido»).                                                                                                                                          |                                                                                                                                                                                                                                                                                                      |
| Misceláneos                                                                                                                                                |                                                                                                                                                                                     |                                                                                                                                                                                 |                                                                                                                                                                                                                                                                                                      |
| NUL                                                                                                                                                        | (Null, «nulo»): carácter nulo. Se utiliza para consumir tiempo u ocupar espacio en una cinta cuando no hay datos.                                                                   |                                                                                                                                                                                 | Puede proporcionar caracteres de control suplementarios, o permitir el envío de caracteres de datos con cualquier combinación de bits.                                                                                                                                                               |
| BEL                                                                                                                                                        | (Bell, «pitido»): utilizado para llamar la atención humana. Puede controlar una alarma o dispositivos que requieren llamar la atención.                                             | DC1, DC2, DC3, DC4 (Device Controls, «controles de dispositivo»): caracteres para el control de ciertos dispositivos auxiliares o características especiales de los terminales. |                                                                                                                                                                                                                                                                                                      |
| SO                                                                                                                                                         | (Shift out, «fuera de código»): indica que los caracteres que siguen no deben interpretarse utilizando el estándar, hasta que llegue un carácter SI.                                | CAN                                                                                                                                                                             | (Cancel, «cancelar»): indica que el dato que le precede en el mensaje o en el bloque debe descartarse (normalmente porque se ha detectado un error).                                                                                                                                                 |
| SI                                                                                                                                                         | (Shift In, «dentro de código»): indica que los caracteres que siguen deben interpretarse de acuerdo con el estándar.                                                                | EM                                                                                                                                                                              | (End of Medium, «fin del medio»): indica el final físico de una tarjeta, una cinta, u otro medio, o el final de la parte del medio solicitada o utilizada.                                                                                                                                           |
| DEL                                                                                                                                                        | (Delete, «borrar»): utilizado para borrar caracteres (por ejemplo, en una cinta de papel perforando cada posición).                                                                 | SUB                                                                                                                                                                             | (Substitute, «sustituir»): sustituir un carácter que se ha detectado como erróneo o no válido.                                                                                                                                                                                                       |
| SP.                                                                                                                                                        | (Space, «espacio»): carácter no imprimible, utilizado para separar palabras, para mover el mecanismo de impresión o para adelantar el cursor una posición.                          | ESC                                                                                                                                                                             | (Escape, «salir»): carácter pensado para proporcionar una ampliación del código, de forma que permita que un número especificado de caracteres contiguos que lo siguen tengan un significado alternativo.                                                                                            |
| DLE                                                                                                                                                        | (Data Link Escape, «salir del enlace de datos»): carácter que puede cambiar el significado de uno o más caracteres consecutivos que lo siguen.                                      |                                                                                                                                                                                 |                                                                                                                                                                                                                                                                                                      |

**S.Z. MÓDULOS DE E/S****FUNCIONES DE UN MÓDULO**

Las principales funciones y requisitos de un módulo de E/S se encuentran dentro de las siguientes categorías:

- Control y temporización
- Comunicación con el procesador
- Comunicación con los dispositivos
- Almacenamiento temporal de datos
- Detección de errores

En cualquier momento, el procesador puede comunicarse con uno o más dispositivos externos en cualquier orden, según las necesidades de E/S del programa. Los recursos internos, tales como la memoria principal y el bus del sistema, deben compartirse entre distintas actividades, incluyendo la E/S de datos. Así, la función de E/S incluye ciertos requisitos de control y temporización, para coordinar el tráfico entre los recursos internos y los dispositivos externos. Por ejemplo, el control de la transferencia de datos desde un dispositivo externo al procesador podría implicar la siguiente secuencia de pasos:

1. El procesador interroga al módulo de E/S para comprobar el estado del dispositivo conectado al mismo.
2. El módulo de E/S devuelve el estado del dispositivo.
3. Si el dispositivo está operativo y preparado para transmitir, el procesador solicita la transferencia del dato mediante una orden al módulo de E/S.
4. El módulo de E/S obtiene un dato (por ejemplo, de 8 ó 16 bits) del dispositivo externo.
5. Los datos se transfieren desde el módulo de E/S al procesador.

Si el sistema utiliza un bus, entonces cada una de las interacciones entre el procesador y el módulo de E/S implica uno o más arbitrajes del bus.

Además, el esquema simplificado previo muestra que el módulo de E/S debe tener la capacidad de establecer comunicación con el procesador y con el dispositivo externo. La comunicación con el procesador implica:

- Decodificación de órdenes: El módulo de E/S acepta órdenes del procesador. Estas órdenes se envían generalmente utilizando líneas del bus de control. Por ejemplo, un módulo de E/S para un controlador de disco podría recibir las siguientes órdenes: LEER SECTOR (READ SECTOR), ESCRIBIR SECTOR (WRITE SECTOR), BUSCAR número de pista (SEEK track number), y EXPLORAR IDentificador de registro (SCAN record ID). Cada una de las dos últimas órdenes incluye un parámetro que es enviado a través del bus de datos.
- Datos: El procesador y el módulo de E/S intercambian datos a través del bus de datos.
- Información de estado: Puesto que los periféricos son lentos, es importante conocer el estado del módulo de E/S. Por ejemplo, si se solicita a un módulo de E/S que envíe datos al procesador (lectura), puede que no esté preparado, por encontrarse todavía respondiendo a una orden de E/S previa. Esta situación puede indicarse con una señal de estado. Señales de estado usuales son BUSY (ocupado) y READY (preparado). También puede haber señales para informar de ciertas situaciones de error.

- **Reconocimiento de dirección:** Igual que cada palabra de memoria tiene una dirección, cada dispositivo de E/S tiene otra. Así, un módulo de E/S puede reconocer una única dirección para cada uno de los periféricos que controla.

Por otra parte, el módulo de E/S debe ser capaz de comunicarse con el dispositivo. Esta comunicación implica intercambiar órdenes, información del estado y datos (Figura 6.2).

Una tarea esencial para un módulo de E/S es el almacenamiento temporal de datos (data buffering). La necesidad de esta función es clara, si se considera la Figura 6.3. Mientras que la velocidad de transferencia desde y hacia la memoria principal o el procesador es bastante alta, dicha velocidad puede ser varios órdenes de magnitud menor para la mayoría de los dispositivos periféricos. Los datos provenientes de la memoria se envían al módulo de E/S en ráfagas rápidas. Los datos se almacenan temporalmente en el módulo de E/S y después se envían al periférico a la velocidad de éste. En el sentido contrario, los datos se almacenan para no mantener a la memoria ocupada en una operación de transferencia lenta. Así, el módulo de E/S debe ser capaz de operar a las velocidades, tanto del dispositivo como de la memoria.

Por último, un módulo de E/S a menudo es responsable de la detección de errores y de informar de estos errores al procesador. Una clase de errores son los defectos mecánicos y eléctricos en el funcionamiento del dispositivo (por ejemplo: papel atascado o pista de disco en mal estado). Otra clase está constituida por los cambios accidentales en los bits al transmitirse desde el dispositivo al módulo de E/S. Para detectar estos errores de transmisión, frecuentemente se utiliza algún tipo de código de detección de errores. Un ejemplo típico es el uso de un bit de paridad en cada carácter de datos. Por ejemplo, un carácter ASCII utiliza 7 de los bits de un byte. El octavo bit se asigna de manera que el número total de «unos» en el byte sea par (paridad par) o impar (paridad impar). Cuando se recibe un byte, el módulo de E/S comprueba la paridad para determinar si se ha producido un error.

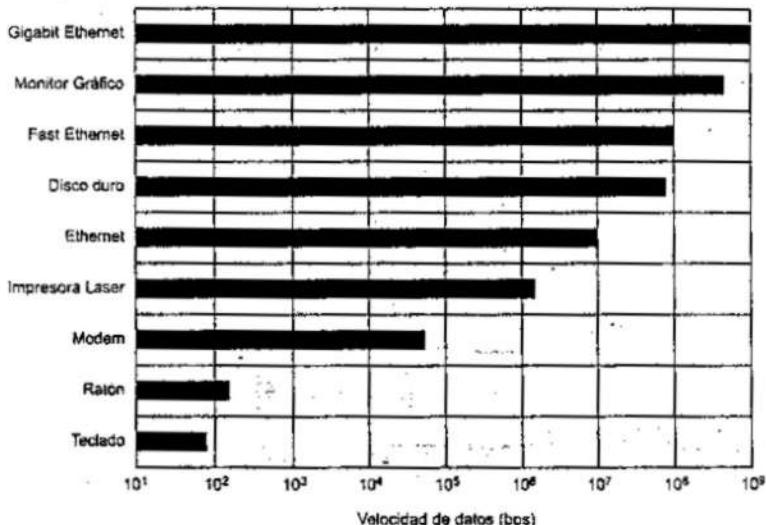


Figura 6.3. Velocidades típicas de transferencia en dispositivos de E/S.

### ESTRUCTURA DE UN MÓDULO DE E/S

La complejidad de los módulos de E/S y el número de dispositivos externos que controlan varían considerablemente. Aquí, únicamente pretendemos realizar una descripción muy general. (Un dispositivo específico, el Intel 82C55A, se describe en la Sección 6.4.) La Figura 6.4 muestra un diagrama de bloques de un módulo de E/S. El módulo se conecta al resto del computador a través de un conjunto de líneas (por ejemplo, líneas del bus del sistema). Los datos que se transfieren a y desde el módulo se almacenan temporalmente en uno o más registros de datos. Además, puede haber uno o más registros de estado que proporcionan información del estado presente. Un registro de estado también puede funcionar como un registro de control, para recibir información de control del procesador. La lógica que hay en el módulo interactúa con el procesador a través de una serie de líneas de control. Estas son las que utilizan el procesador para proporcionar las órdenes al módulo de E/S. Algunas de las líneas de control pueden ser utilizadas por el módulo de E/S (por ejemplo, para las señales de arbitraje y estado). El módulo también debe ser capaz de reconocer y generar las direcciones asociadas a los dispositivos que controla. Cada módulo de E/S tiene una dirección única o, si controla más de un dispositivo externo, un conjunto único de direcciones. Por último, el módulo de E/S posee la lógica específica para la interfaz con cada uno de los dispositivos que controla.

El funcionamiento de un módulo de E/S permite que el procesador vea a una amplia gama de dispositivos de una forma simplificada. Ante el espectro de posibilidades que pueden darse, el módulo de E/S debe ocultar los detalles de temporización, formatos y electromecánica de los dispositivos externos, para que el procesador pueda funcionar únicamente en términos de órdenes de lectura y escritura y, posiblemente, órdenes de abrir y cerrar ficheros. En su forma más sencilla, el módulo de E/S puede, no obstante, dejar al procesador parte del trabajo de control del dispositivo (por ejemplo, rebobinar una cinta).

Un módulo de E/S que se encarga de la mayoría de los detalles del procesamiento, presentando al procesador una interfaz de alto nivel, se denomina generalmente *canal de E/S o procesador de E/S*. Un módulo que sea bastante simple y requiera un control detallado, normalmente se denomina *controlador de E/S o controlador de dispositivo*. Los controladores de E/S aparecen usualmente en microcomputadores, mientras que los canales de E/S se utilizan en grandes computadoras centrales («mainframes»).

En lo que sigue, haremos uso del término genérico *módulo de E/S* cuando no haya posibilidad de confusión, y utilizaremos los términos específicos cuando sea preciso.

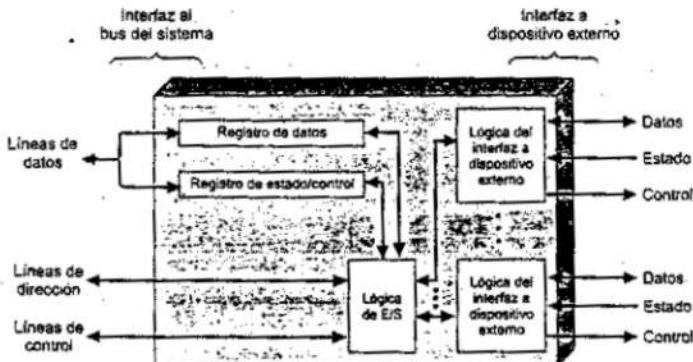


Figura 6.4. Diagrama de bloques de un módulo de E/S.

### 6.3. E/S PROGRAMADA

Son posibles tres técnicas para las operaciones de E/S. Con la *E/S programada*, los datos se intercambian entre el procesador y el módulo de E/S. El procesador ejecuta un programa que controla directamente la operación de E/S, incluyendo la comprobación del estado del dispositivo, el envío de una orden de lectura o escritura y la transferencia del dato. Cuando el procesador envía una orden al módulo de E/S, debe esperar hasta que la operación de E/S concluya. Si el procesador es más rápido que el módulo de E/S, el procesador desperdicia este tiempo. Con la *E/S mediante interrupciones*, el procesador proporciona la orden de E/S, continua ejecutando otras instrucciones, y es interrumpido por el módulo de E/S cuando éste ha terminado su trabajo. Tanto con E/S programada como con interrupciones, el procesador es responsable de extraer los datos de la memoria principal en una salida, y de almacenar los datos en la memoria principal en una entrada. La alternativa se conoce como *acceso directo a memoria* (DMA). En este caso, el módulo de E/S y la memoria principal intercambian datos directamente, sin la intervención del procesador.

La Tabla 6.3 indica la relación entre estas tres técnicas. En esta sección estudiamos la E/S programada. La E/S mediante interrupciones y el DMA se consideran, respectivamente, en las dos próximas secciones.

### RESUMEN

Cuando el procesador está ejecutando un programa y encuentra una instrucción relacionada con una E/S, ejecuta dicha instrucción mandando una orden al módulo de E/S apropiado. Con E/S programada, el módulo de E/S realiza la acción solicitada, y después activa los bits apropiados en el registro de estado de E/S (Figura 6.4). El módulo de E/S no realiza ninguna otra acción para avisar al procesador. En concreto, no interrumpe al procesador. De esta forma, el procesador es responsable de comprobar periódicamente el estado del módulo de E/S hasta que encuentra que la operación ha terminado.

Para explicar la técnica de la E/S programada, la consideraremos primero desde el punto de vista de las órdenes de E/S que envía la CPU al módulo de E/S, y después desde el punto de vista de las instrucciones de E/S que ejecuta la CPU.

### ÓRDENES DE E/S

Al ejecutar una instrucción relacionada con una E/S, el procesador proporciona una dirección, especificando el módulo de E/S particular y el dispositivo externo, y una orden de E/S. Hay cuatro tipos de órdenes de E/S que puede recibir un módulo de E/S cuando es direccionado por el procesador:

Tabla 6.3. Técnicas de E/S

|                                                   | Sin interrupciones | Usando interrupciones          |
|---------------------------------------------------|--------------------|--------------------------------|
| Transferencia de E/S a memoria a través de la CPU | E/S programada     | E/S mediante interrupciones    |
| Transferencia directa de E/S a memoria            |                    | Acceso directo a memoria (DMA) |

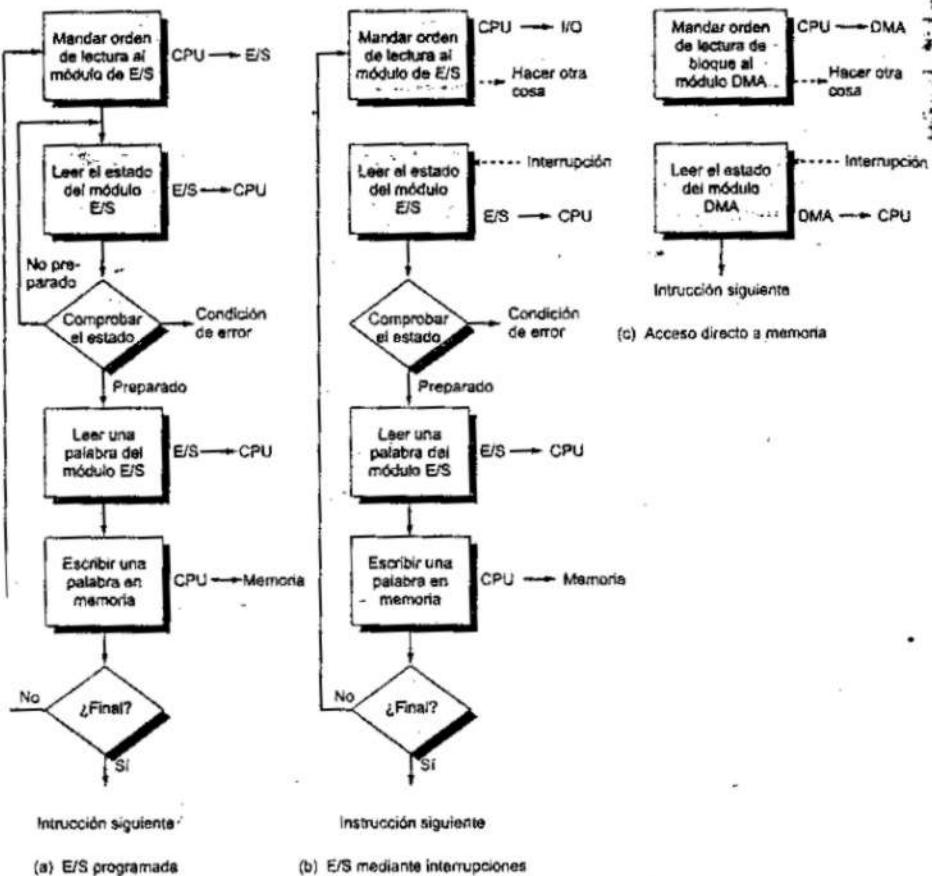


Figura 6.5. Tres técnicas para la entrada de un bloque de datos.

- **Control:** Se utiliza para activar el periférico e indicarle qué hacer. Por ejemplo, puede indicarse a una unidad de cinta magnética que se rebobine o que avance al registro siguiente. Estas órdenes son específicas del tipo particular de periférico.
- **Test:** Se utiliza para comprobar diversas condiciones de estado asociadas con el módulo de E/S y sus periféricos. El procesador podrá comprobar si el periférico en cuestión está conectado y disponible para su uso. También podrá saber si la operación de E/S más reciente ha terminado y si se ha producido algún error.
- **Lectura:** Hace que el módulo de E/S capte un dato de un periférico y lo sitúe en un buffer interno (representado como un registro de datos en la Figura 6.4). Después, el procesador puede obtener el dato solicitando que el módulo de E/S lo ponga en el bus de datos.

- **Ecritura:** Hace que el módulo de E/S capte un dato (byte o palabra) del bus de datos y posteriormente lo transmita al periférico.

La Figura 6.5a proporciona un ejemplo del uso de la E/S programada para leer un bloque de datos desde un dispositivo periférico (por ejemplo, un registro de una cinta) y almacenarlo en memoria. Los datos se leen palabra por palabra (16 bits, por ejemplo). Por cada palabra leída, el procesador debe permanecer en un ciclo de comprobación de estado hasta que determine que la palabra está disponible en el registro de datos del módulo de E/S. Este diagrama de flujo resalta la principal desventaja de esta técnica: es un proceso que consume tiempo y mantiene al procesador innecesariamente ocupado.

## INSTRUCCIONES DE E/S

En la E/S programada, hay una estrecha correspondencia entre las instrucciones de E/S que el procesador capta de memoria y las órdenes de E/S que la CPU envía a un módulo de E/S al ejecutar las instrucciones. Es decir, las instrucciones se pueden hacer corresponder fácilmente con las órdenes de E/S y, a menudo, hay una simple relación de uno a uno. La forma de la instrucción depende de la manera de direccionar los dispositivos externos.

Normalmente, habrá muchos dispositivos de E/S conectados al sistema a través de los módulos de E/S. Cada dispositivo tiene asociado un identificador único o dirección. Cuando el procesador envía una orden de E/S, la orden contiene la dirección del dispositivo deseado. Así, cada módulo de E/S debe interpretar las líneas de dirección para determinar si la orden es para él.

Cuando el procesador, la memoria principal y las E/S comparten un bus común, son posibles dos modos de direccionamiento: asignado en memoria (memory-mapped) y aislado. Con las E/S asignadas en memoria, existe un único espacio de direcciones para las posiciones de memoria y los dispositivos de E/S. La CPU considera los registros de estado y de datos de los módulos de E/S como posiciones de memoria, y utiliza las mismas instrucciones máquina para acceder tanto a memoria como a los dispositivos de E/S. Así por ejemplo, con 10 líneas de dirección se puede acceder a un total de 1.024 posiciones de memoria y direcciones de E/S, en cualquier combinación.

Con las E/S asignadas en memoria, se necesita una sola línea de lectura y una sola línea de escritura en el bus. Alternativamente, el bus puede disponer de líneas de lectura y escritura en memoria junto con líneas para órdenes de entrada y salida. En este caso, las líneas de órdenes especifican si la dirección se refiere a una posición de memoria o a un dispositivo de E/S. El rango completo de direcciones está disponible para ambos. De nuevo, con 10 líneas de dirección, el sistema puede soportar ahora 1.024 posiciones de memoria y 1.024 direcciones de E/S. Puesto que el espacio de direcciones de E/S está aislado del de memoria, éste se conoce con el nombre de *E/S aislada*.

En la Figura 6.6 se contrastan estas dos técnicas de E/S programada. La Figura 6.6a muestra cómo podría ver el programador la interfaz con un dispositivo de entrada sencillo tal como un teclado, cuando se utiliza E/S asignada en memoria. Se asumen direcciones de 10 bits, con una memoria de 512 palabras (posiciones 0-512) y hasta 512 direcciones de E/S (posiciones 512-1.023). Se dedican dos direcciones a la entrada de teclado desde un terminal concreto. La dirección 516 se refiere al registro de datos, y la dirección 517 al registro de estado, que además funciona como registro de control para recibir las órdenes del procesador. El programa que se muestra lee un byte de datos desde el teclado y lo escribe en el registro acumulador del procesador. Obsérvese cómo el procesador ejecuta un bucle hasta que el byte de datos está disponible.



Figura 6.6. E/S mapeada en memoria y mapa de E/S aislado.

Con E/S aislada (Figura 6.6b), los puertos de E/S sólo son accesibles mediante una orden específica de E/S, que activa las líneas de órdenes de E/S del bus.

La mayor parte de los procesadores disponen de un conjunto relativamente grande de instrucciones distintas para acceder a memoria. Si se utiliza E/S aislada, sólo existen unas pocas instrucciones de E/S. Por eso, una ventaja de la E/S asignada en memoria es que se puede utilizar este amplio repertorio de instrucciones, permitiendo una programación más eficiente. Una desventaja es que se utiliza parte del valioso espacio de direcciones de memoria. Tanto la E/S asignada en memoria como la aislada se usan comúnmente.

#### 6.4. E/S MEDIANTE INTERRUPCIONES

El problema con la E/S programada es que el procesador tiene que esperar un tiempo considerable a que el módulo de E/S en cuestión esté preparado para recibir o transmitir los datos. El procesador, mientras espera, debe comprobar repetidamente el estado del módulo de E/S. Como consecuencia, se degrada el nivel de prestaciones de todo el sistema.

Una alternativa consiste en que el procesador, tras enviar una orden de E/S a un módulo, continúe realizando algún trabajo útil. Después, el módulo de E/S interrumpirá al procesador para solicitar su servicio cuando esté preparado para intercambiar datos con él. El procesador ejecuta entonces la transferencia de datos como antes, y después continúa con el procesamiento previo.

Estudiemos como funciona, primero desde el punto de vista del módulo de E/S. Para una entrada, el módulo de E/S recibe una orden READ del procesador. Entonces, el módulo de E/S procede a leer el dato desde el periférico asociado. Una vez que el dato está en el registro de datos del módulo, el módulo envía una interrupción al procesador a través de una línea de control. Después, el módulo espera hasta que el procesador solicite su dato. Cuando ha recibido la solicitud, el módulo sitúa su dato en el bus de datos y pasa a estar preparado para otra operación de E/S.

Desde el punto de vista del procesador, las acciones para una entrada son las que siguen: El procesador envía una orden READ de lectura. Entonces, pasa a realizar otro trabajo (es decir, el procesador puede estar ejecutando programas distintos al mismo tiempo). Al final de cada ciclo de instrucción, el procesador comprueba las interrupciones (Figura 3.9). Cuando se pide la interrupción desde el módulo de E/S, el procesador guarda el contexto (es decir, el contador de programa y los registros del procesador) del programa en curso, y procesa la interrupción. En este caso, el procesador lee la palabra de datos del módulo de E/S y la almacena en memoria. Después, recupera el contexto del programa que estaba ejecutando (o de otro programa) y continúa su ejecución.

La Figura 6.5b muestra el uso de E/S con interrupciones para leer un bloque de datos. Comparese con la Figura 6.5a. La E/S con interrupciones es más eficiente que la E/S programada, porque elimina las esperas innecesarias. No obstante, las E/S con interrupciones consumen gran cantidad del tiempo del procesador, puesto que cada palabra de datos que va desde la memoria al módulo de E/S, o viceversa, debe pasar a través del procesador.

## PROCESAMIENTO DE LA INTERRUPCIÓN

Consideremos con más detalle el papel del procesador en las E/S. Cuando se produce una interrupción, se disparan una serie de eventos en el procesador, tanto a nivel hardware como software. La Figura 6.7 muestra una secuencia típica. Cuando un dispositivo de E/S termina una operación de E/S, se produce la siguiente secuencia de eventos en el hardware:

1. El dispositivo envía una señal de interrupción al procesador.
2. El procesador termina la ejecución de la instrucción en curso antes de responder a la interrupción, como indica la Figura 3.9.
3. El procesador comprueba si hay interrupciones, determina que hay una, y envía una señal de reconocimiento al dispositivo que originó la interrupción. La señal de reconocimiento hace que el dispositivo desactive su señal de interrupción.
4. Ahora, el procesador necesita prepararse para transferir el control a la rutina de interrupción. Para empezar, debe guardar la información necesaria para continuar el programa en curso en el punto en que se interrumpió. La información mínima que se precisa es (a) el estado del procesador, que se almacena en un registro llamado «palabra de estado del programa» (PSW, Program Status Word), y (b) la posición de la siguiente instrucción a ejecutar, que está contenida en el contador de programa. Estos registros se pueden introducir en la pila de control del sistema<sup>2</sup>.
5. Despues, el procesador carga el contador de programa con la posición de inicio del programa de gestión de la interrupción solicitada. Según sea la arquitectura del computador y el diseño del sistema operativo, puede haber un sólo programa, uno por cada tipo de interrupción o uno por cada dispositivo y cada tipo de interrupción. Si hay más de una rutina de gestión de interrupción, el procesador debe determinar qué

<sup>2</sup> Véase el Apéndice 9A para una discusión sobre el funcionamiento de la pila.

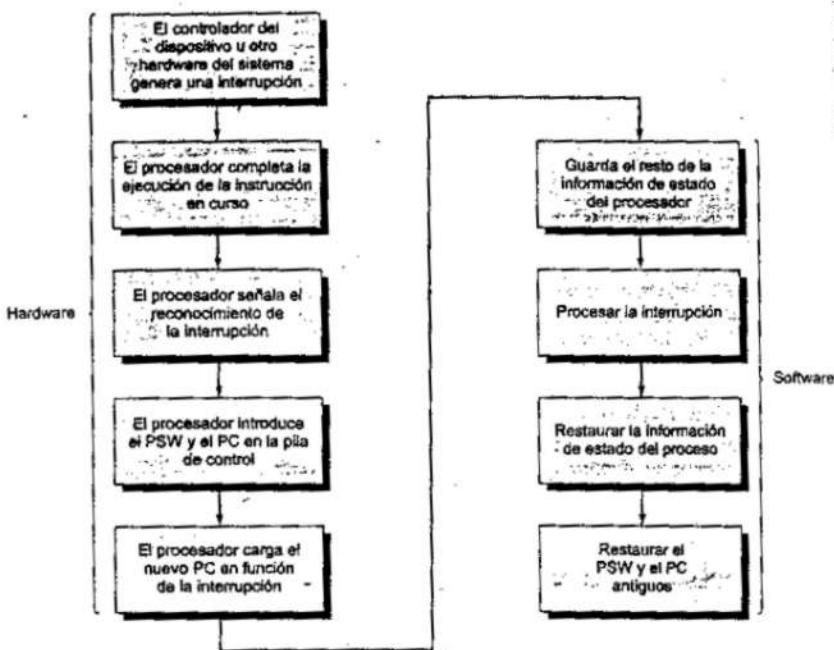
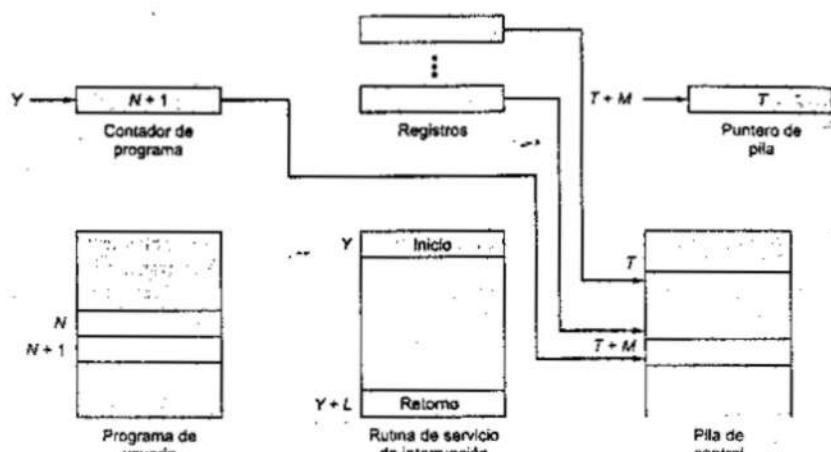
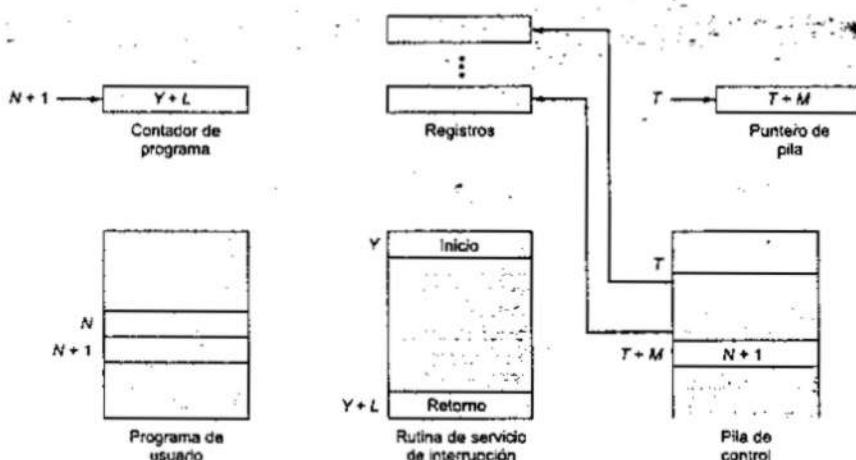


Figura 6.7. Procesamiento en memoria y mapa de E/S aislado.

programa llamar. Esta información puede haber sido incluida en la señal de interrupción original, o bien el procesador puede tener que enviar una solicitud al dispositivo que originó la interrupción para que éste responda con la información que se precise.

Una vez que el contador de programa se ha cargado, el procesador continúa con el ciclo de instrucción siguiente, que empieza con la captación de instrucción. Puesto que la instrucción a captar viene determinada por el contenido del contador de programa, el control se transfiere al programa de gestión de interrupción. La ejecución de este programa da lugar a las siguientes operaciones:

6. Hasta este momento, se han almacenado en la pila del sistema el contador de programa y el PSW del programa interrumpido. Sin embargo, hay otra información que se considera parte del «estado» de un programa en ejecución. En concreto, se deben guardar los contenidos de los registros del procesador, puesto que estos registros pueden ser utilizados por la rutina de interrupción. Usualmente, la rutina de gestión de interrupción empezará almacenando en la pila los contenidos de todos los registros. La Figura 6.8a muestra un ejemplo sencillo. En este caso, un programa de usuario es interrumpido después de la instrucción de la posición  $N$ . Los contenidos de todos los registros, junto con la dirección de la siguiente instrucción ( $N + 1$ ), se introducen en la pila. El puntero de la pila se actualiza para que apunte a la nueva cabecera de la pila, y el contador de programa se actualiza para que apunte al comienzo de la rutina de servicio de interrupción.

(a) Interrupción después de la instrucción de la Posición  $N$ 

(b) Retorno al final de la Interrupción

Figura 6.8. Cambios en memoria y en los registros debido a una interrupción.

7. La rutina de gestión de la interrupción puede continuar ahora procesando la interrupción. Esto incluirá el examen de la información de estado relativa a la operación de E/S, o a cualquier otro evento que causara la interrupción. También puede implicar el envío al dispositivo de E/S de órdenes o señales de reconocimiento adicionales.

8. Cuando el procesamiento de la interrupción ha terminado, los valores de los registros almacenados se recuperan de la pila y se vuelven a almacenar en los registros (como ejemplo, véase la Figura 6.8b).
9. El paso final es recuperar los valores del PSW y del contador de programa desde la pila. Como resultado de esto, la siguiente instrucción que se ejecute pertenecerá al programa previamente interrumpido.

Obsérvese que es importante almacenar toda la información del estado del programa interrumpido para que éste pueda reanudarse. Esto se debe a que la interrupción no es una llamada a una rutina realizada desde el programa. En cambio, la interrupción puede producirse en cualquier momento y, por consiguiente, en cualquier punto de la ejecución del programa de usuario. Una interrupción es impredecible. De hecho, como se verá en el siguiente capítulo, los dos programas pueden no tener nada en común, y pueden pertenecer a distintos usuarios.

## CUESTIONES DE DISEÑO

En la implementación de las E/S mediante interrupciones, aparecen dos cuestiones. Primero, puesto que casi invariablemente habrá múltiples módulos de E/S, ¿cómo determina el procesador qué dispositivo ha provocado la interrupción? Y, por otro lado, si se han producido varias interrupciones, ¿cómo decide el procesador la que debe atender?

Consideremos en primer lugar la identificación del dispositivo. Hay cuatro tipos de técnicas que se utilizan comúnmente:

- Múltiples líneas de interrupción
- Consulta software (software polling)
- Conexión en cadena (Daisy chain), (consulta hardware, vectorizada)
- Arbitraje de bus (vectorizada)

La aproximación más directa al problema consiste en proporcionar *varias líneas de interrupción* entre el procesador y los módulos de E/S. Sin embargo, no resulta práctico dedicar más de unas pocas líneas del bus o terminales del procesador a líneas de interrupción. En consecuencia, incluso si se utilizan varias líneas, es probable que a cada una se conecten varios módulos de E/S. Por eso, se debe utilizar alguna de las otras tres técnicas en cada línea.

Una alternativa es la *consulta software*. Cuando el procesador detecta una interrupción, se produce una bifurcación a una rutina de servicio de interrupción que se encarga de consultar a cada módulo de E/S para determinar el módulo que ha provocado la interrupción. La consulta podría realizarse mediante una línea específica (por ejemplo, TEST\_E/S). En este caso, el procesador activa TEST\_E/S y sitúa la dirección de un módulo de E/S en las líneas de dirección. El módulo de E/S responde positivamente si solicitó la interrupción. Como alternativa, cada módulo de E/S podría disponer de un registro de estado direccional. Entonces, el procesador lee el estado del registro de cada módulo de E/S para identificar el módulo que solicitó la interrupción. Una vez identificado el módulo, se produce una bifurcación para que el procesador ejecute la rutina de servicio específica para ese dispositivo.

La desventaja de la consulta software está en el tiempo que consume. Una técnica más eficiente consiste en utilizar la *conexión en cadena* (Daisy Chain) de los módulos de E/S, que proporciona, de hecho, una consulta hardware. Un ejemplo de configuración que utiliza esta conexión en cadena se muestra en la Figura 3.25. Todos los módulos de E/S comparten una línea común para solicitar interrupciones. La línea de reconocimiento de interrupción se conecta encadenando los módulos uno tras otro. Cuando el procesador recibe una interrupción, activa el reconocimiento de interrupción. Esta señal se propaga a través de la secuencia

de módulos de E/S hasta que alcanza un módulo que solicitó interrupción. Normalmente, este módulo responde colocando una palabra en las líneas de datos. Esta palabra se denomina *vector* y es la dirección del módulo de E/S o algún otro tipo de identificador específico. En cualquier caso, el procesador utiliza el vector como un puntero a la rutina de servicio de dispositivo apropiada. Así se evita tener que ejecutar una rutina de servicio general en primer lugar. Esta técnica se conoce con el nombre de *interrupciones vectorizadas*.

Hay otra técnica que hace uso de las interrupciones vectorizadas; se trata del *arbitraje de bus*. Con el arbitraje de bus, un módulo de E/S debe, en primer lugar, disponer del control del bus antes de poder activar la línea de petición de interrupción. Así, sólo un módulo puede activar la línea en un instante. Cuando el procesador detecta la interrupción, responde mediante la línea de reconocimiento de interrupción. Después, el módulo que solicitó la interrupción sitúa su vector en las líneas de datos.

Las técnicas enumeradas arriba sirven para identificar el módulo de E/S que solicita interrupción. Además, proporcionan una forma de asignar prioridades cuando más de un dispositivo está pidiendo que se sirva su interrupción. Con varias líneas de interrupción, el procesador simplemente selecciona la línea con más prioridad. Con la consulta software, el orden en el que se consultan los módulos determina su prioridad. De igual forma, el orden de los módulos en la conexión en cadena (Daisy Chain) determina su prioridad. Finalmente, el arbitraje de bus puede emplear un esquema de prioridad como el discutido en la Sección 3.4.

Ahora pasamos a considerar dos ejemplos de estructuras de interrupción.

## CÓNTROLADOR DE INTERRUPCIONES INTEL 82C59A

El 80386 de Intel posee una sola línea de petición de interrupción (INTR, Interrupt request) y una sola línea de reconocimiento de interrupción (INTA, Interrupt Acknowledge). Para que el 80386 pueda manejar flexiblemente cierta variedad de dispositivos y estructuras de prioridad, normalmente se configura con un árbito de interrupciones externo, el 82C59A. Los dispositivos externos se conectan al 82C59A, que a su vez se conecta al 80386.

La Figura 6.9 ilustra el uso del 82C59A para conectar varios módulos de E/S con el 80386. Un único 82C59A puede manejar hasta 8 módulos. Si se precisa controlar más de 8 módulos, se pueden disponer en cascada para manejar hasta 64 módulos.

La única responsabilidad del 82C59A es la gestión de interrupciones. Acepta las solicitudes de interrupción de los dispositivos conectados a él, determina qué interrupción tiene la prioridad más alta, y se lo indica entonces al procesador activando la señal INTR. El procesador reconoce la solicitud mediante la línea INTA. Esto hace que el 82C59A sitúe el vector apropiado en el bus de datos. Entonces, el procesador puede iniciar el procesamiento de la interrupción y comunicarse directamente con el módulo de E/S para leer o escribir datos.

El 82C59A es programable. El 80386 determina el esquema de prioridad que se va a utilizar, cargando una palabra de control en el 82C59A. Son posibles los siguientes modos de interrupción.

- **Completamente anidado:** Las solicitudes de interrupción se ordenan según un nivel de prioridad desde 0 (IR0) hasta 7 (IR7).
- **Rotatorio:** En algunas aplicaciones hay varios dispositivos con igual prioridad de interrupción. En este modo, un dispositivo pasa a tener la menor prioridad del grupo después de ser servido.
- **Con máscara especial:** Se permite que el procesador pueda inhibir selectivamente las interrupciones desde ciertos dispositivos.

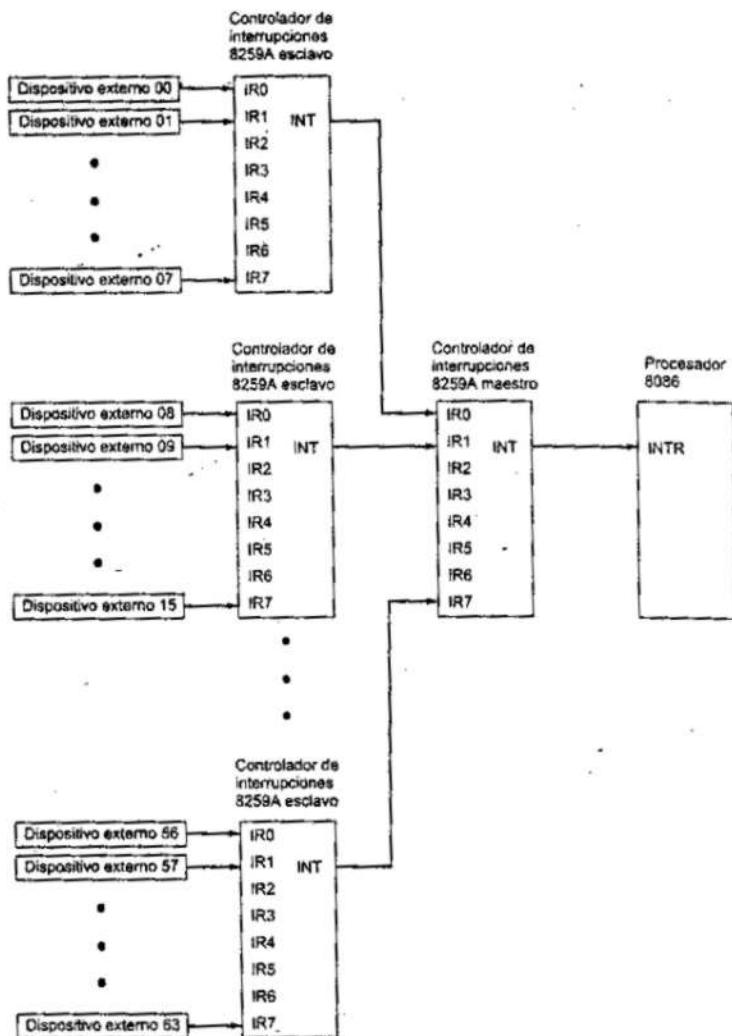


Figura 6.9. Uso del controlador de interrupciones 82C59A.

### LA INTERFAZ PROGRAMABLE DE PERIFÉRICOS INTEL 82C55A

Como ejemplo de un módulo de E/S utilizado para la E/S programada y para la E/S mediante interrupciones, consideraremos la interfaz programable de periféricos Intel 82C55A. El 82C55A es un módulo de E/S de propósito general integrado en un sólo chip y diseñado para

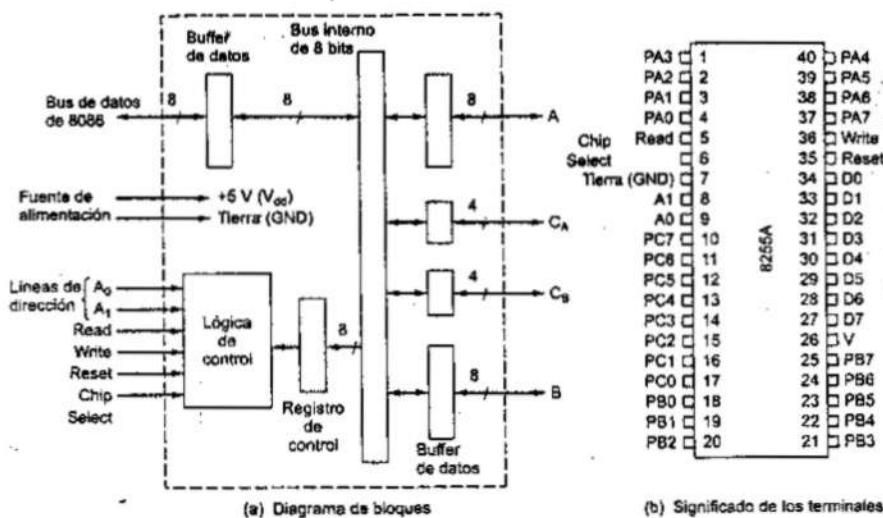


Figura 6.10. Interfaz programable de periféricos 82C55A de Intel.

ser usado con el procesador Intel 8086. La Figura 6.10 muestra el diagrama general de bloques junto con la asignación de terminales para el encapsulado de 40 terminales que lo contiene.

El lado derecho del diagrama de bloques es la interfaz externa del 82C55A. Las 24 líneas de E/S son programables por el 80386 mediante un registro de control. El 80386 puede fijar el valor del registro de control para especificar los diversos modos de operación y configuraciones. Las 24 líneas se dividen en tres grupos de 8 bits (A, B, C). Cada grupo puede funcionar como un puerto de E/S de 8 bits. Además, el grupo C se subdivide en grupos de 4 bits ( $C_A$  y  $C_B$ ), que pueden usarse conjuntamente con los puertos de E/S A y B. Configurado de esta forma, esos grupos contienen las señales de control y estado.

El lado izquierdo del diagrama de bloques es la interfaz interna con el bus del 80386. Ésta incluye un bus de datos bidireccional de 8 bits (D0 a D7), usado para transferir datos a, y desde, los puertos de E/S, y para transferir la información al registro de control. Las dos líneas de direcciones especifican uno de los tres puertos de E/S o el registro de control. Una transferencia se producirá cuando la línea de selección de chip (CHIP SELECT) se activa junto con la línea de lectura (READ) o escritura (WRITE). La línea RESET se utiliza para iniciar el módulo.

El procesador escribe en el registro de control para seleccionar el modo de operación y para definir las señales, en su caso. En el Modo 0 de operación, los tres grupos de 8 líneas externas funcionan como tres puertos de E/S de 8 bits. Cada puerto puede ser designado como de entrada o de salida. En caso contrario, los grupos A y B funcionan como puertos de E/S, y las líneas del grupo C sirven de líneas de control para A y B. Las líneas de control tienen dos funciones principales: la sincronización mediante conformidad de señales (*handshaking*) y la petición de interrupciones. La conformidad es un mecanismo sencillo de temporización. El emisor utiliza una línea de control como línea de datos listos (DATA READY) para indicar que hay un dato en las líneas de datos de E/S. El receptor utiliza otra

línea como reconocimiento (ACKNOWLEDGE), para indicar que el dato se ha leído y que las líneas de datos se pueden liberar. Se puede designar otra línea como línea de petición de interrupción (INTERRUPT REQUEST) en el bus del sistema.

Como el 82C55A es programable a través del registro de control, puede utilizarse para controlar diversos dispositivos periféricos simples. La Figura 6.11 ilustra su uso para controlar un terminal con teclado y pantalla. El teclado proporciona 8 bits de entrada. Dos de estos bits, SHIFT y CONTROL, tienen un significado especial para el programa de gestión de teclado que ejecuta el procesador. Sin embargo, este significado es transparente para el 82C55A,

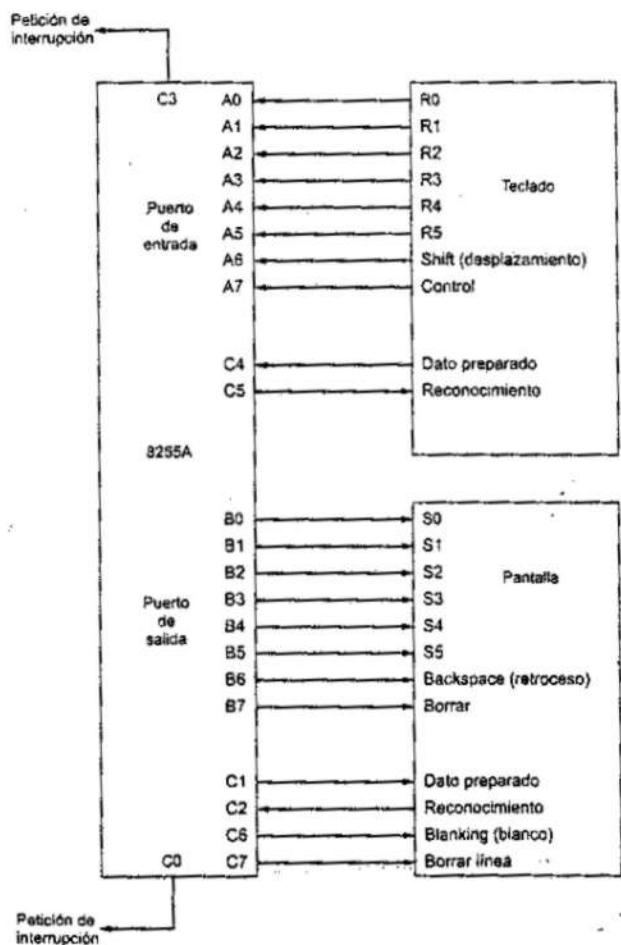


Figura 6.11. Interfaz teclado/pantalla en el 82C55A.

que simplemente acepta los 8 bits de datos y los pone en el bus de datos del sistema. Existen dos líneas para la sincronización del teclado mediante conformidad («handshaking»).

La pantalla también está conectada a un puerto de datos de 8 bits. De nuevo, dos de los bits tienen un significado específico que es transparente para el 82C55A. Junto a las dos líneas para la sincronización mediante conformidad, hay dos líneas más para funciones de control adicionales.

## 6.5. ACCESO DIRECTO A MEMORIA

### INCONVENIENTES DE LA E/S PROGRAMADA Y CON INTERRUPCIONES

La E/S con interrupciones, aunque más eficiente que la sencilla E/S programada, también requiere la intervención activa del procesador para transferir datos entre la memoria y el módulo de E/S, y cualquier transferencia de datos debe seguir un camino a través del procesador. Por tanto, ambas formas de E/S presentan dos inconvenientes inherentes:

1. La velocidad de transferencia de E/S está limitada por la velocidad a la cual el procesador puede comprobar y dar servicio a un dispositivo.
2. El procesador debe dedicarse a la gestión de las transferencias de E/S; se debe ejecutar cierto número de instrucciones por cada transferencia de E/S (véase Figura 6.5).

Existe un cierto compromiso entre estos dos inconvenientes. Considerese una transferencia de un bloque de datos. Utilizando E/S programada, el procesador se dedica a la tarea de la E/S y puede transferir datos a alta velocidad al precio de no hacer nada más. La E/S con interrupciones libera en parte al procesador a expensas de reducir la velocidad de E/S. No obstante, ambos métodos tienen un impacto negativo, tanto en la actividad del procesador como en la velocidad de transferencia de E/S.

Cuando hay que transferir grandes volúmenes de datos, se requiere una técnica más eficiente: el acceso directo a memoria (DMA).

### FUNCIONAMIENTO DEL DMA

El DMA requiere un módulo adicional en el bus del sistema. El módulo de DMA (Figura 6.12) es capaz de imitar al procesador y, de hecho, es capaz de recibir el control del sistema cedido por el procesador. Necesita dicho control para transferir datos a y desde memoria a través del bus del sistema. Para hacerlo, el módulo de DMA debe utilizar el bus sólo cuando el procesador no lo necesita, o debe forzar al procesador a que suspenda temporalmente su funcionamiento. Esta última técnica es la más común y se denomina *robo de ciclo* (cycle stealing), puesto que, en efecto, el módulo de DMA roba un ciclo de bus.

Cuando el procesador desea leer o escribir un bloque de datos, envía una orden al módulo de DMA, incluyendo la siguiente información:

- Si se solicita una lectura o una escritura, utilizando la línea de control de lectura o escritura entre el procesador y el módulo de DMA.
- La dirección del dispositivo de E/S en cuestión, indicada a través de las líneas de datos.
- La posición inicial de memoria a partir de donde se lee o se escribe, indicada a través de las líneas de datos y almacenada por el módulo de DMA en su registro de direcciones.

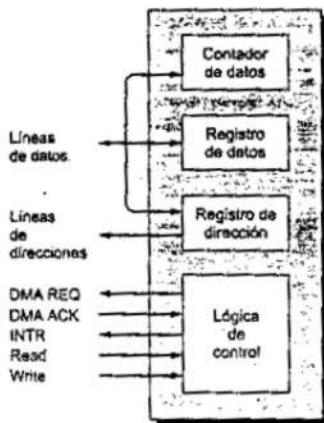


Figura 6.12. Diagrama de bloques típico de un módulo de DMA.

- El número de palabras a leer o escribir, también indicado a través de las líneas de datos y almacenado en el registro de cuenta de datos.

Después, el procesador continúa con otro trabajo. Ha delegado la operación de E/S al módulo de DMA, que se encargará de ella. El módulo de DMA transfiere el bloque completo de datos, palabra a palabra, directamente desde, o hacia, la memoria, sin que tenga que pasar a través del procesador. Cuando la transferencia se ha terminado, el módulo de DMA envía una señal de interrupción al procesador. Así pues, el procesador sólo interviene al comienzo y al final de la transferencia (Figura 6.5c).

La Figura 6.13 muestra en qué momento del ciclo de instrucción puede detenerse el procesador. En cada caso, el procesador se detiene justo antes de necesitar el bus. Después, el módulo de DMA transfiere una palabra y devuelve el control al procesador. Obsérvese que no se trata de una interrupción: el procesador no guarda el contexto ni hace nada más. En cambio, el procesador espera durante un ciclo de bus. El efecto resultante es que el procesa-

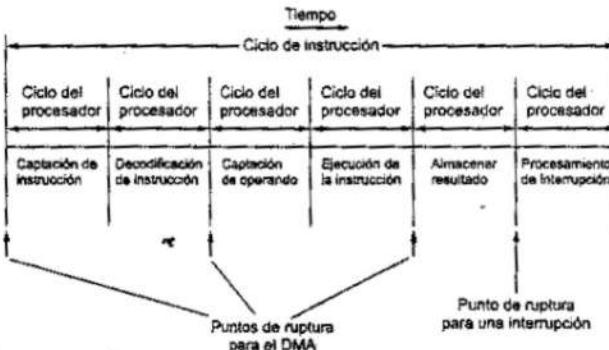


Figura 6.13. Puntos de ruptura para el DMA y las interrupciones en un ciclo de instrucción.

dor es más lento ejecutando los programas. No obstante, para una transferencia de E/S de varias palabras el DMA es mucho más eficiente que la E/S mediante interrupciones o la programada.

El mecanismo de DMA puede configurarse de diversas formas. La Figura 6.14 muestra algunas posibilidades. En el primer ejemplo, todos los módulos comparten el mismo bus del sistema. El módulo de DMA, actuando como un procesador suplementario, utiliza E/S programada para intercambiar datos entre la memoria y un módulo de E/S a través del módulo de DMA. Esta configuración, si bien es la más económica, es claramente ineficiente. Igual que con la E/S programada controlada por el procesador, la transferencia de cada palabra consume dos ciclos de bus.

El número de ciclos de bus necesarios puede reducirse sustancialmente si se integran las funciones de DMA y de E/S. Como indica la Figura 6.14b, esto significa que existe un camino entre el módulo de DMA y uno o más módulos de E/S que no incluye al bus del sistema. La lógica de DMA puede ser parte de un módulo de E/S o puede ser un módulo separado que controla a uno o más módulos de E/S. Este concepto se puede llevar algo más lejos conectando los módulos de E/S a un módulo de DMA mediante un bus de E/S (Figura 6.14c). Esto reduce a uno el número de interfaces de E/S en el módulo de DMA, y permite una configuración fácilmente ampliable. En todos estos casos (Figuras 6.14b y c), el bus del sistema, que el módulo de DMA comparte con el procesador y la memoria, es usado por el módulo de DMA sólo para intercambiar datos con la memoria. El intercambio de datos entre los módulos de DMA y E/S se produce fuera del bus del sistema.

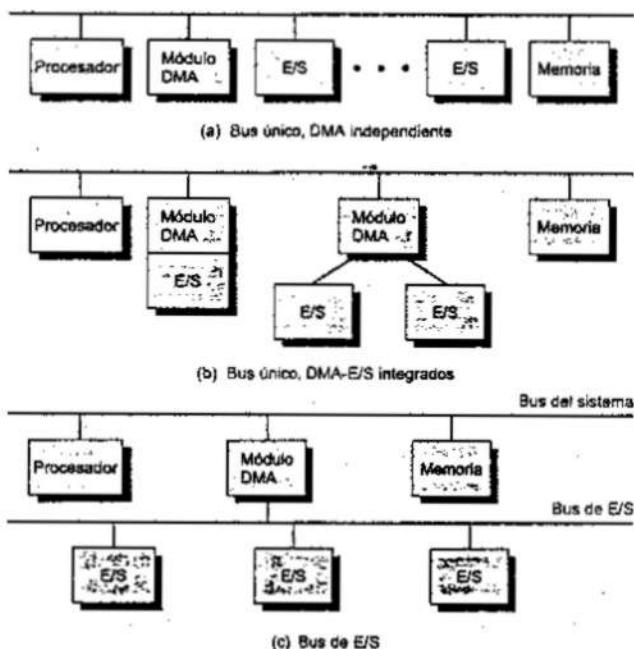


Figura 6.14. Configuraciones alternativas para el DMA.

**6.6. CANALES Y PROCESADORES DE E/S****LA EVOLUCIÓN DEL FUNCIONAMIENTO DE LAS E/S**

A medida que los computadores han evolucionado, la complejidad y sofisticación de sus componentes se ha incrementado. En ningún lugar se hace más evidente que en el funcionamiento de las E/S. Ya se ha considerado parte de esta evolución. Sus etapas se pueden resumir como sigue:

1. La CPU controla directamente al periférico. Esta situación se observa en los dispositivos simples controlados por microprocesadores.
2. Se añade un controlador o módulo de E/S. La CPU utiliza E/S programada sin interrupciones. De esta forma, la CPU se independiza de los detalles específicos de las interfaces de los dispositivos externos.
3. Se utiliza la misma configuración del paso 2, pero ahora se emplean interrupciones. La CPU no necesita esperar a que se realice la operación de E/S, incrementándose la eficiencia.
4. El módulo de E/S tiene acceso directo a la memoria a través del DMA. Ahora se puede transferir un bloque de datos a, o desde, la memoria sin implicar a la CPU, excepto al comienzo y al final de la transferencia.
5. El módulo de E/S se mejora, haciendo que se comporte como un procesador en sí mismo, con un repertorio especializado de instrucciones orientado a las E/S. La CPU hace que el procesador de E/S ejecute un programa de E/S en memoria. El procesador de E/S capta y ejecuta sus instrucciones sin intervención de la CPU. Esto permite que la CPU pueda especificar una secuencia de actividades de E/S y ser interrumpida cuando se haya completado la secuencia entera.
6. El módulo de E/S tiene una memoria local propia y es, de hecho, un computador en sí mismo. Con esta arquitectura se puede controlar un conjunto grande de dispositivos de E/S con la mínima intervención de la CPU. Un uso común de este tipo de arquitectura ha sido la comunicación con terminales interactivos. El procesador de E/S se ocupa de la mayoría de las tareas correspondientes al control de los terminales.

Siguiendo el camino marcado por esta evolución, cada vez más y más funciones de E/S se realizan sin la intervención de la CPU. La CPU es revelada de las tareas relacionadas con las tareas de E/S, mejorando las prestaciones. Con las dos últimas etapas, (5-6), se ha producido un cambio importante, al introducir el concepto de un módulo de E/S capaz de ejecutar un programa. En el caso de la etapa 5, el módulo normalmente se denomina *canal de E/S*. En el paso 6, se utiliza usualmente el término *procesador de E/S*. Sin embargo, ambos términos se aplican ocasionalmente a ambas situaciones. En lo que sigue, utilizaremos el término *canal de E/S*.

**CARACTERÍSTICAS DE LOS CANALES DE E/S**

El canal de E/S representa una ampliación del concepto de DMA. Un canal de E/S puede ejecutar instrucciones de E/S, lo que le confiere un control completo sobre las operaciones de E/S. En un computador con tales dispositivos, la CPU no ejecuta instrucciones de E/S. Dichas instrucciones se almacenan en memoria principal para ser ejecutadas por un procesador de uso específico contenido en el propio canal de E/S. De esta forma, la CPU inicia una transferencia de E/S, indicando al canal de E/S que debe ejecutar un programa de la memo-

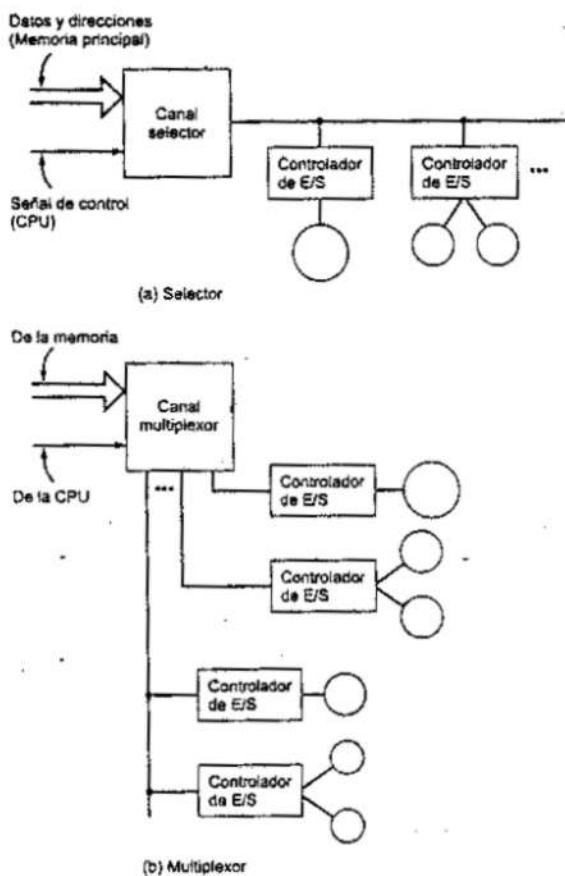


Figura 6.15. Arquitectura de un canal de E/S.

ria. El programa especifica el dispositivo o dispositivos, el área o áreas de memoria para almacenamiento, la prioridad y las acciones a realizar en ciertas situaciones de error. El canal de E/S sigue estas instrucciones y controla la transferencia de datos.

Como ilustra la Figura 6.15, son comunes dos tipos de canales de E/S. Un *canal selector* controla varios dispositivos de velocidad elevada y, en un instante dado, se dedica a transferir datos a uno de esos dispositivos. Es decir, el canal de E/S selecciona un dispositivo y efectúa la transferencia de datos. Cada dispositivo o pequeño grupo de dispositivos es manejado por un *controlador*, o módulo de E/S, que es similar a los módulos de E/S de los que se ha discutido. Así, el canal de E/S se utiliza en lugar de la CPU para controlar estos controladores de E/S. Un *canal multiplexor* puede manejar las E/S de varios dispositivos al mismo tiempo. Para dispositivos de velocidad reducida, un *multiplexor de byte* acepta o transmite caracteres tan rápido como es posible a varios dispositivos. Por ejemplo, la cadena de caracteres

resultante a partir de tres dispositivos con diferentes velocidades y cadenas individuales  $A_1A_2A_3A_4\dots$ ,  $B_1B_2B_3B_4\dots$  y  $C_1C_2C_3C_4\dots$ , podría ser  $A_1B_1C_1A_2C_2A_3B_2C_3A_4\dots$ , y así sucesivamente. Para dispositivos de velocidad elevada, un *multiplexor de bloque* entrelaza bloques de datos de los distintos dispositivos.

## 6.7. LA INTERFAZ EXTERNA SCSI Y FireWire

### TIPOS DE INTERFACES

La interfaz entre el periférico y el módulo de E/S debe ajustarse a la naturaleza y la forma de funcionar del periférico. Una de las principales características de la interfaz es si es serie o paralela (Figura 6.16). En una *interfaz paralela*, hay varias líneas que conectan el módulo de E/S y el periférico, y se transfieren varios bits simultáneamente a través del bus de datos. En una *interfaz serie*, hay sólo una línea para transmitir los datos, y los bits deben transmitirse uno a uno. Las interfaces paralelas se utilizan usualmente para los dispositivos de alta velocidad, como una cinta o un disco. Las interfaz serie son más propias de impresoras y terminales.

En cualquier caso, el módulo de E/S debe establecer un diálogo con el periférico. En términos generales, el diálogo para una operación de escritura es como sigue:

1. El módulo de E/S envía una señal de control solicitando permiso para enviar datos.
2. El periférico reconoce la solicitud.
3. El módulo de E/S transfiere los datos (una palabra o un bloque, según el periférico).
4. El periférico reconoce la recepción de los datos.

Una operación de lectura se realiza de forma similar.

Para el funcionamiento del módulo de E/S, es clave disponer de un registro de acople (buffer) interno que pueda almacenar los datos a transferir entre el periférico y el resto del sistema. Este buffer permite que el módulo de E/S pueda compensar las diferencias de velocidad entre el bus del sistema y sus líneas externas.

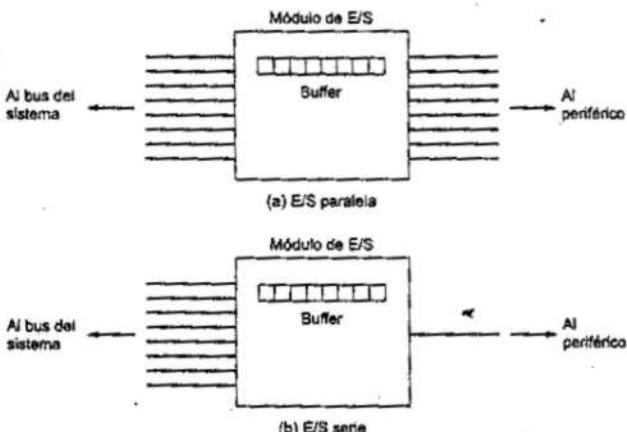


Figura 6.16. E/S paralela y serie.

## CONFIGURACIONES PUNTO-A-PUNTO Y MULTIPUNTO

La conexión entre un módulo de E/S del computador y los dispositivos externos puede ser punto-a-punto o multipunto. Una interfaz punto-a-punto proporciona una línea específica entre el módulo de E/S y el dispositivo externo. En los sistemas pequeños (PC, estaciones de trabajo), existen usualmente enlaces punto-a-punto para el teclado, la impresora y el modem externo. Un ejemplo típico de este tipo de interfaz es la especificación EIA-232 (para su descripción, véase [STAL97]).

Las interfaces externas multipunto, utilizadas para soportar dispositivos de almacenamiento masivo (discos y cintas) y dispositivos multimedia (CD-ROM, equipos de video y audio), tienen una importancia creciente. Estas interfaces multipunto son de hecho buses externos, y poseen el mismo tipo de lógica que los buses que se discutieron en el Capítulo 3. En esta sección, consideraremos dos ejemplos clave: SCSI y FireWire.

## INTERFAZ SCSI (SMALL COMPUTER SYSTEM INTERFACE)

Un buen ejemplo de una interfaz para dispositivos periféricos externos es la SCSI. Popularizada primero en el Macintosh en 1984, la SCSI se utiliza ahora ampliamente en sistemas compatibles, tanto Macintosh como IBM-PC, y también en muchas estaciones de trabajo. SCSI es la interfaz estándar para los dispositivos de CD-ROM, equipos de audio y dispositivos de almacenamiento masivo. La SCSI utiliza una interfaz paralela, con 8, 16 o 32 líneas de datos.

La configuración SCSI se considera usualmente como un bus, aunque en realidad los dispositivos se conectan juntos de forma encadenada («daisy chain»). Cada dispositivo SCSI tiene dos conectores: uno de entrada y otro de salida. Todos los dispositivos se conectan en cadena, con un extremo de la cadena conectado al computador. Todos los dispositivos funcionan independientemente, y pueden intercambiar datos entre ellos, igual que con el computador anfitrión («host»). Por ejemplo, un disco duro puede guardar su contenido en una cinta sin que tenga que intervenir el procesador anfitrión. Como se describe más adelante, los datos se transfieren mediante paquetes, que constituyen un mensaje.

### Versión de SCSI

La especificación original SCSI, llamada ahora SCSI-1, se desarrolló al iniciarse los años 80. SCSI-1 utiliza 8 líneas de datos y opera a una frecuencia de reloj de 5 MHz, o a una velocidad de datos de 5 MBytes/s. La SCSI-1 permite que se conecten al computador hasta siete dispositivos conectados en cadena.

En 1991, se revisó la especificación y se introdujo la SCSI-2. Los cambios más notables fueron la expansión opcional a 16 o 32 líneas de datos y el aumento de la frecuencia de reloj a 10 MHz. Como resultado se tiene una velocidad de datos máxima de 20 o 40 Mbytes/s.

Actualmente se está trabajando en la especificación SCSI-3, que permitirá mayores velocidades.

### Señales y fases

Todos los intercambios en el bus SCSI se producen entre un dispositivo iniciador y un dispositivo seleccionado. Usualmente, el computador anfitrión es el iniciador y un controlador periférico es el dispositivo seleccionado, pero algunos dispositivos pueden asumir los dos pa-

peles. En cualquier caso, toda la actividad del bus se produce en una secuencia de fases. Las fases son las siguientes:

- **Bus Libre:** Indica que ningún dispositivo está utilizando el bus, y que éste está disponible.
- **Arbitraje:** Permite que un dispositivo tome el control del bus, de manera que pueda iniciar o reanudar un proceso de E/S.
- **Selección:** El iniciador selecciona un dispositivo para realizar una operación, tal como una orden de lectura o escritura.
- **Reselección:** Permite que el dispositivo seleccionado se vuelva a conectar al iniciador para reanudar una operación que se inició previamente, pero que fue suspendida por el dispositivo.
- **Orden:** El dispositivo puede solicitar una orden de información al iniciador.
- **Datos:** El dispositivo puede solicitar la transferencia de un dato desde el dispositivo hacia el iniciador (entrada de datos, Data In) o viceversa (salida de datos, Data Out).
- **Estado:** Permite que el dispositivo solicite que se envíe la información de estado desde el dispositivo al iniciador.
- **Mensaje:** Permite que el dispositivo solicite la transferencia de uno o más mensajes desde el dispositivo al iniciador (entrada de mensaje, Message In) o viceversa (salida de mensaje, Message Out).

La Figura 6.17 ilustra el orden en que se producen las fases del bus SCSI. Despues de conectarse el sistema, o despues de un reinicio, el bus pasa a la fase de bus libre. Esta está seguida por la fase de arbitraje, que usualmente da lugar a que un dispositivo tome el control. Si el arbitraje falla, el bus vuelve a la fase de bus libre. Cuando el arbitraje termina con éxito, el bus pasa a una fase de selección o reselección, en la que se designa un iniciador y un dispositivo seleccionado para realizar la transferencia. Despues de que se hayan determinado los dos dispositivos, se producirá una o más fases de transferencia de informacion entre los dos dispositivos (fases de orden, datos, estado y mensaje). La fase final de transferencia de informacion es usualmente la fase de entrada de mensaje, en la que un mensaje de desconexión o de orden completa se transfiere al iniciador, seguida de una fase de bus libre.

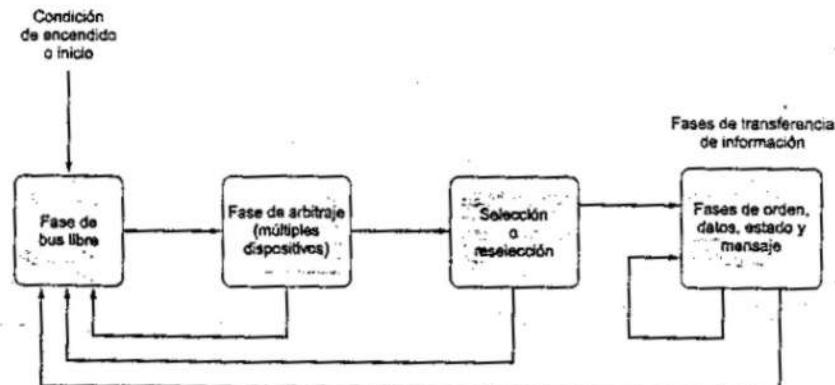


Figura 6.17. Fases en un bus SCSI.

Una característica importante del bus SCSI es la capacidad de reselección. Si una orden enviada necesita algún tiempo para completarse, el dispositivo seleccionado puede liberar el bus y volverse a conectar al iniciador más tarde. Por ejemplo, el computador anfitrión puede enviar una orden a un dispositivo de disco para que dé formato al disco, y el dispositivo realiza la operación sin que se necesite acceder al bus.

La especificación SCSI-1 define un cable formado por 18 líneas de señal, 9 de control y 9 para datos (8 líneas de datos y una de paridad). Las líneas de control son:

- **BSY:** Activado por el iniciador o el dispositivo seleccionado para indicar que el bus está ocupado.
- **SEL:** Utilizada por el iniciador para seleccionar el dispositivo que debe responder a una orden, o por un dispositivo para reseleccionar el iniciador del que se había desconectado. La señal I/O distingue entre selección y reselección.
- **C/D:** Utilizada por el dispositivo seleccionado para indicar si los datos del bus de datos corresponden a información de control (orden, estado o mensaje) o de datos.
- **I/O:** Utilizada por el dispositivo seleccionado para controlar la dirección del movimiento de los datos en el bus de datos.
- **MSG:** Utilizado por el dispositivo seleccionado para indicar al iniciador que la información que está siendo transferida es un mensaje.
- **REQ:** Utilizada por el dispositivo seleccionado para solicitar una transferencia de información de datos. Como respuesta a la señal REQ, el iniciador acepta datos del bus durante la fase de entrada de datos o coloca información en el bus durante la fase de salida de datos.
- **ACK:** Utilizada por el iniciador para reconocer la señal REQ del dispositivo seleccionado. La señal ACK indica que el iniciador ha situado información en el bus durante una fase de salida de datos o ha aceptado datos del bus durante una fase de entrada de datos.
- **ATN:** Utilizada por el iniciador para informar al dispositivo de que hay un mensaje disponible para su transferencia. El iniciador puede activar la señal durante la fase de selección, o en cualquier momento después de que el dispositivo haya asumido el control del bus.
- **RST:** Utilizada para iniciar el bus (reset).

La Tabla 6.4 muestra las relaciones entre las señales del bus y las fases del bus. Para la SCSI-2, hay líneas de datos adicionales y líneas de paridad, además de un par de líneas REQ/ACK adicionales.

### Temporización de la SCSI

La Figura 6.18 ilustra una temporización SCSI típica, que puede utilizarse para explicar las diversas fases y señales del bus. Este ejemplo corresponde a una orden de lectura, que transfiere datos desde el dispositivo al iniciador.

La secuencia comienza en la fase de bus libre, con todas las líneas desactivadas. A continuación, hay una fase de arbitraje, en la que uno o más dispositivos compiten por el control del bus. Cada uno de los dispositivos activa la señal BSY y una de las líneas de datos. Cada uno de los ocho dispositivos (el anfitrión y hasta siete dispositivos) tiene un único identificador, ID, de 0 a 7, y cada dispositivo activa una de las líneas de datos correspondiente a su propio ID. Cada ID tiene una prioridad, correspondiendo a 7 la

Tabla 6.4. Señales de un bus SCSI

| Fase del bus       | Señales del cable A |      |                             |             |                   |      | Señales del cable B |                                           |      |
|--------------------|---------------------|------|-----------------------------|-------------|-------------------|------|---------------------|-------------------------------------------|------|
|                    | BSY                 | SEL  | C/D,<br>I/O,<br>MSG,<br>REQ | ACK,<br>ATN | DB(7-0),<br>DB(P) | REQP | ACKB                | DB(31-8),<br>DB(P1),<br>DB(P2),<br>DB(P3) |      |
| Bus libre          | N                   | N    | N                           | N           | N                 | N    | N                   | N                                         | N    |
| Arbitraje          | Todos               | Gana | N                           | N           | S ID              | N    | N                   | N                                         | N    |
| Selección          | I&D                 | In   | N                           | In          | In                | N    | N                   | N                                         | N    |
| Reselección        | I&D                 | Disp | Disp                        | In          | Disp              | N    | N                   | N                                         | N    |
| Orden              | Disp                | N    | Disp                        | In          | In                | N    | N                   | N                                         | N    |
| Entrada de datos   | Disp                | N    | Disp                        | In          | Disp              | Disp | In                  | In                                        | Disp |
| Salida de datos    | Disp                | N    | Disp                        | In          | In                | Disp | In                  | In                                        |      |
| Estado             | Disp                | N    | Disp                        | In          | Disp              | N    | N                   | N                                         |      |
| Entrada de mensaje | Disp                | N    | Disp                        | In          | Disp              | N    | N                   | N                                         |      |
| Salida de mensaje  | Disp                | N    | Disp                        | In          | In                | N    | N                   | N                                         |      |

Todos: La señal es activada por todos los dispositivos SCSI que participa en el arbitraje.

S ID: Bit de datos específicos (ID de SCSI) activado por cada dispositivo que participa en el arbitraje.

I&D: La señal es activada por el iniciador, el dispositivo seleccionado o ambos, tal y como se especifica en la fase de selección y reselección.

In: Si está activa, sólo la activa el iniciador.

N: La señal no está activa.

Gana: La señal es activada por el ganador en el proceso de arbitraje.

Disp: Si está activa, sólo la activa el dispositivo seleccionado.

más alta y a 0 la más baja. Si más de un dispositivo activa su ID durante la fase de arbitraje, el dispositivo con más prioridad gana. Los otros dispositivos reconocen esto al observar las líneas de datos y ceden el control.

Una vez que un dispositivo ha ganado en el arbitraje se convierte en iniciador. Indica la entrada en la fase de selección activando la señal SEL. Durante esta fase, activa las dos líneas de datos correspondientes tanto a su propio ID como al ID del dispositivo seleccionado. Después de un retardo, niega la señal BSY. Cuando el dispositivo seleccionado detecta que SEL está activa, BSY e I/O a 0, y reconoce su ID, activa la señal BSY. Cuando el iniciador detecta BSY, libera el bus de datos y niega SEL.

A continuación, el dispositivo seleccionado indica que ha pasado a la fase de orden, activando la línea C/D; esta línea permanecerá activada durante esta fase. Después activa REQ para solicitar el primer byte de la orden del iniciador. El iniciador sitúa el primer byte de la orden en el bus de datos y activa ACK. Después de que el dispositivo haya leído el byte, niega REQ, y el iniciador niega ACK. El primer byte de la orden contiene el código de operación de la orden, que indica cuantos bytes quedan por transferir. Estos bytes adicionales se transfieren con el mismo intercambio de señales REQ/ACK antes y después de cada transferencia de byte.

Después de que el dispositivo seleccionado ha recibido e interpretado la orden, hace pasar al bus a la fase de entrada de datos, negando la línea C/D (indicando que el bus de datos contiene datos) y activando la línea de E/S (indicando que la dirección de la transferencia es desde el dispositivo al iniciador). El dispositivo seleccionado sitúa el primer byte del dato solicitado en el bus de datos y activa la señal de la línea REQ. El iniciador activa la línea ACK después de haber leído el byte. Los bytes de datos adicionales se transfieren con el mismo intercambio de señales REQ/ACK, antes y después de cada transferencia de un byte.

Después de transferirse todos los datos solicitados, el dispositivo seleccionado hace pasar al bus a la fase de estado, y transfiere un byte de estado al iniciador, indicando que se ha

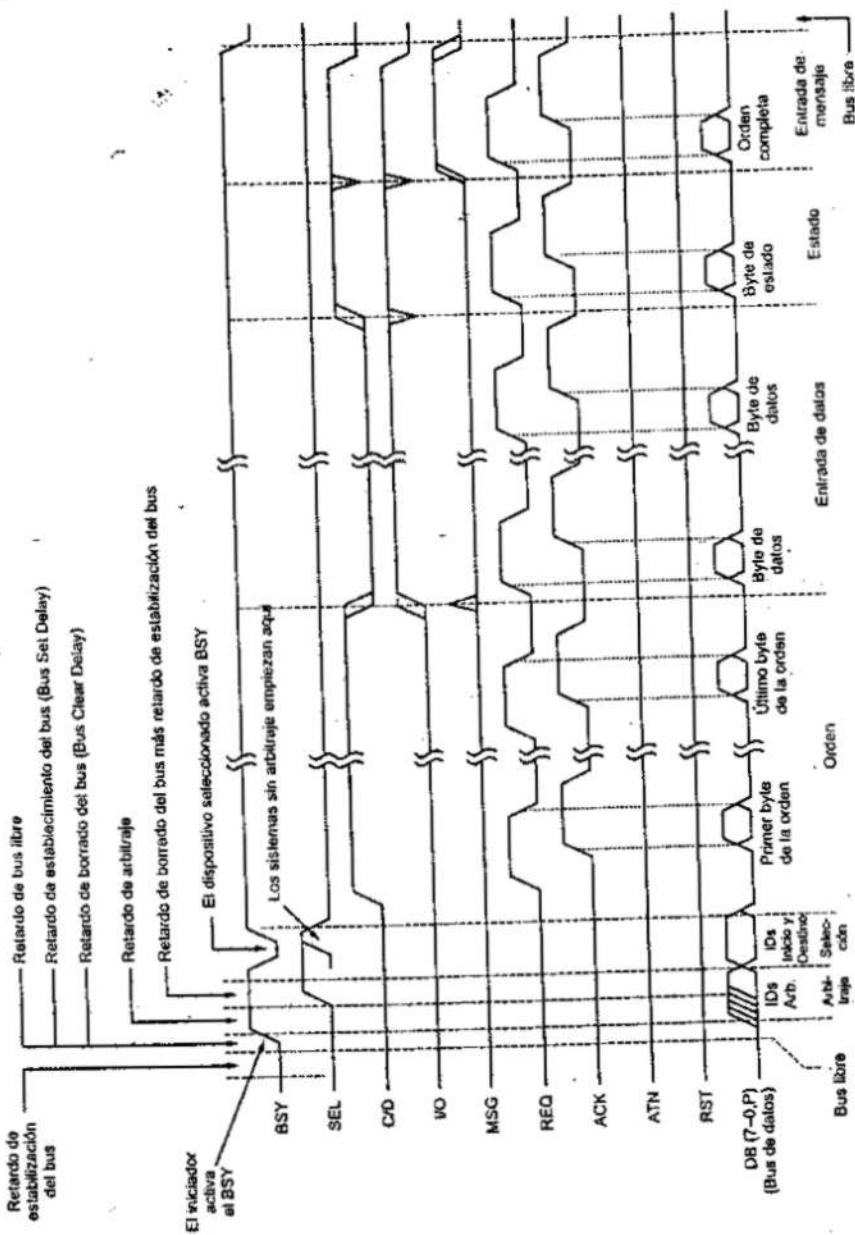


Figura 6.18. Ejemplo de diagrama de tiempos SCSI.

completado la transferencia con éxito. En este caso, la línea C/D se activa de nuevo, y la línea de E/S se mantiene activa. El iniciador y el dispositivo seleccionado intercambian las señales REQ/ACK para coordinar la transferencia del byte de estado.

Finalmente, el dispositivo hace pasar al bus a la fase de entrada de mensaje activando la línea MSG y transfiriendo un byte con el mensaje de orden completa. Una vez que el byte es recibido por el iniciador, el dispositivo seleccionado libera todas las señales del bus para hacer pasar al bus a la fase de bus libre.

La transacción que se representa en la Figura 6.18 es un ejemplo de transferencia asíncrona. Con la transferencia asíncrona, siempre se deben intercambiar las señales REQ/ACK para transferir un byte. El SCSI también permite la transferencia síncrona, que implica un retraso menor. El modo de transferencia síncrona se utiliza sólo en las fases de entrada de datos y salida de datos. Puesto que el modo de transferencia implícito es el asíncrono, el modo síncrono debe negociarse. El dispositivo seleccionado envía al iniciador un mensaje de solicitud de transferencia de datos síncrona, que contiene su periodo de transferencia mínimo y el tiempo máximo que permite entre REQ y su ACK correspondiente. El iniciador responde con el mismo mensaje, indicando su propio periodo de transferencia mínimo y el máximo tiempo entre REQ y ACK.

Una vez que se ha producido este intercambio, el modo de transferencia síncrona se establece con el mayor de los dos períodos mínimos de transferencia y con el menor de los dos tiempos mínimos entre REQ y ACK. La transferencia de datos se produce como sigue. El emisor enviará los bytes de datos separados, al menos por un tiempo igual al periodo de transferencia mínimo y señalará el envío de cada byte mediante un pulso en REQ. El receptor no necesita reconocer mediante ACK cada byte recibido, pero debe activar ACK dentro del tiempo máximo entre REQ y ACK. El emisor puede enviar una secuencia continua de bytes mientras esté recibiendo los ACK dentro del periodo de tiempo de retardo. Si un ACK se retrasa, el emisor debe esperar a que llegue el reconocimiento.

### Mensajes

Los mensajes se intercambian entre los iniciadores y los dispositivos para gestionar la interfaz SCSI. Hay tres formatos de mensaje: de un sólo byte, de dos bytes y mensajes extendidos a tres o más bytes. Algunos ejemplos de mensaje son

- **Orden completa:** Enviado por el dispositivo al iniciador para indicar que se ha terminado la ejecución de una orden y que se ha enviado el estado al iniciador.
- **Desconexión:** Enviado desde el dispositivo seleccionado para informar al iniciador de que la conexión actual se va a cortar, pero que se necesita una nueva conexión para terminar la transferencia en curso.
- **Error detectado en el iniciador:** Envulado por el iniciador para informar al dispositivo de que se ha producido un error (por ejemplo, de paridad) que no impide que el dispositivo pueda reintentar la operación.
- **Abortar:** Envulado por el iniciador al dispositivo para terminar la operación en curso.
- **Transferencia síncrona de datos:** Intercambiado entre el iniciador y el dispositivo para establecer la transferencia de datos síncrona.

### Órdenes

El núcleo del protocolo SCSI es el conjunto de órdenes. Un iniciador envía una orden para producir un efecto en el dispositivo seleccionado. La orden puede significar la entrada de

datos desde el dispositivo (lectura), enviar datos al dispositivo (escritura), o algún otro tipo de acción específica de un periférico concreto. En todos los casos, la ejecución de la orden implica alguno o todos los pasos siguientes:

- El dispositivo recibe y decodifica la información de la orden.
- Los datos se transfieren a, o desde, el dispositivo (no todas las órdenes realizan este paso).
- El dispositivo genera y devuelve información de su estado.

La orden está definida en un bloque descriptor de órdenes (CDB, Command Descriptor Block) dispuesto por el iniciador. Una vez que se establece la conexión entre un iniciador y el dispositivo seleccionado, el iniciador transfiere el CDB al dispositivo a través del bus de datos para empezar la ejecución de la orden.

La Figura 6.19 muestra el formato general del CDB. Consta de los siguientes campos:

- **Código de operación:** Este código especifica la orden concreta. El código de operación también indica el formato del resto del CDB.
- **Número de unidad lógica:** Identifica un dispositivo físico o virtual. Los números de unidad lógica pueden utilizarse para direccionar varios dispositivos periféricos que comparten un mismo controlador o varios volúmenes lógicos de un disco.
- **Dirección de bloque lógico:** Para las operaciones de lectura/escritura, el CDB incluye una dirección lógica de comienzo para la transferencia.



Figura 6.19. Formato del bloques descriptor de órdenes (Command Descriptor Block) SCSI.

- **Longitud de la transferencia:** Para las operaciones de lectura/escritura, este campo especifica el número de bloques lógicos contiguos de datos a transferir.
- **Longitud de la lista de parámetros:** Especifica el número de bytes enviados durante la fase de salida de datos. Este campo se usa normalmente para los parámetros que se envían al dispositivo (por ejemplo: modo, diagnóstico y parámetros de identificación).
- **Longitud de la asignación:** Especifica el número máximo de bytes que el iniciador ha asignado para los datos recibidos.
- **Control:** Este campo incluye los bits de enlace (Link) e indicador (Flag), que controlan el mecanismo de órdenes enlazadas. Si el bit de enlace está activo, la orden en curso no termina con la fase de bus libre, sino que se agrega a la siguiente orden, empezando su fase de orden. Si el bit indicador está activo, el dispositivo seleccionado envía el mensaje de orden enlazada completa, si la orden termina correctamente, y sitúa en el bit indicador el mismo valor que se tiene en el CDB. Normalmente, un valor de 1 en el bit indicador es utilizado por el dispositivo para originar una interrupción en el iniciador entre órdenes enlazadas.

La especificación SCSI define una amplia gama de tipos de órdenes. La mayor parte de éstas son específicas para una clase particular de dispositivos. El estándar SCSI-2 incluye órdenes para los siguientes tipos de dispositivos:

- Dispositivos de acceso directo
- Dispositivos de acceso secuencial
- Impresoras
- Procesadores
- Dispositivos de una escritura (Write-Once)
- CD-ROM
- Escáneres
- Dispositivos de memoria óptica
- Dispositivos de cambio de medio
- Dispositivos de comunicación

Además, hay un conjunto de 17 órdenes que se aplican a todos los tipos de dispositivos. De estos, cuatro son obligatorias y deben utilizarse por todos los dispositivos:

- **Pregunta (Inquiry):** Solicita que los parámetros del módulo y de los dispositivos conectados a él se envíen al iniciador.
- **Solicitud de situación (Request Sense):** Solicita que el dispositivo seleccionado envíe ciertos datos de su estado (o situación) al iniciador. Estos datos incluyen las condiciones de error (por ejemplo, fin de papel), información de posicionamiento (por ejemplo, final del medio), e información del estado lógico (por ejemplo, la orden actual ha alcanzado una marca).
- **Enviar diagnóstico (Send Diagnostic):** Solicita que el módulo aplique los test para el diagnóstico de fallos a él mismo, a los periféricos que tiene conectados o a ambos.
- **Comprobar si unidad lista (Test Unit Ready):** Proporciona una forma de comprobar si la unidad lógica está preparada.

El repertorio de órdenes del SCSI es uno de los puntos fuertes de esta especificación. Proporciona una forma estándar de acceder a la mayoría de los dispositivos más usuales en los com-

putadores personales y en las estaciones de trabajo, simplificando la tarea de escribir software de E/S para el computador anfitrión.

### BUS SERIE FIREWIRE

Con velocidades en los procesadores próximas a los 100 MHz, y dispositivos de almacenamiento con capacidades del orden de varios gigabits, las demandas de E/S de los computadores personales, las estaciones de trabajo y los servidores son impresionantes. Las tecnologías de canales de E/S de alta velocidad que se han desarrollado para los grandes computadores centrales (mainframes) y los supercomputadores son todavía demasiado caras y voluminosas para que se utilicen en sistemas pequeños. En consecuencia, ha existido un gran interés en desarrollar alternativas a la SCSI y a otras interfaces de E/S, que permitan velocidades elevadas en sistemas pequeños. El resultado es el estándar IEEE 1394 para un bus serie de altas prestaciones, conocido como FireWire.

FireWire presenta ciertas ventajas sobre SCSI y otras interfaces de E/S. Es de muy alta velocidad, bajo costo y fácil de implementar. De hecho, FireWire no sólo se está utilizando en computadores, sino también para productos de electrónica de consumo, tales como cámaras digitales, VCR y televisiones. En estos productos, el bus FireWire se usa para transmitir imágenes de video, que provienen cada vez con más frecuencia de fuentes digitalizadas.

Una de las ventajas de la interfaz FireWire es que utiliza transmisión serie (un bit cada vez) en lugar de paralela. Las interfaces paralelas, como SCSI, necesitan más líneas, lo que significa cables más anchos y más caros, y conectores más anchos y caros con más terminales que se pueden doblar o romper. Un cable con más líneas necesita estar protegido frente a las posibles interferencias eléctricas entre las líneas. Además, con una interfaz paralela es necesaria la sincronización entre las líneas, y constituye un problema más grave a medida que aumenta la longitud del cable.

Además, los computadores son cada vez más pequeños físicamente, incluso a medida que aumentan sus prestaciones y sus necesidades de E/S. Los computadores portátiles y los de bolsillo tienen poco espacio para los conectores, pero necesitan velocidades de datos elevadas para poder manejar imágenes y video.

FireWire intenta proporcionar una única interfaz de E/S con un conector sencillo, que pueda manejar diversos dispositivos a través de un único puerto, de manera que los conectores del ratón, de la impresora, de SCSI, del controlador de disco externo, de sonido, y los de la red de área local, puedan reemplazarse por este único conector. El conector se inspira en el utilizado en la videoconsola Gameboy de Nintendo. Es tan cómodo que basta con localizarlo detrás de la máquina y conectar el dispositivo sin más.

### Configuraciones de FireWire

FireWire utiliza la configuración de conexión en cadena (Daisy-chain), con hasta 63 dispositivos conectados con un sólo puerto. Además, pueden interconectarse mediante adaptadores (bridges) hasta 1.022 buses FireWire, posibilitando que el sistema soporte tantos periféricos como se necesite.

FireWire soporta lo que se conoce como conexión rápida (*«hot plugging»*), que permite conectar y desconectar periféricos sin tener que reconfigurar el sistema. Además, FireWire permite la configuración automática; y no es necesario fijar manualmente los identificadores (ID) del dispositivo o tener en cuenta la posición relativa de los dispositivos. La Figura 6.20 compara la configuración del bus FireWire con la del SCSI. En el bus SCSI, ambos extremos

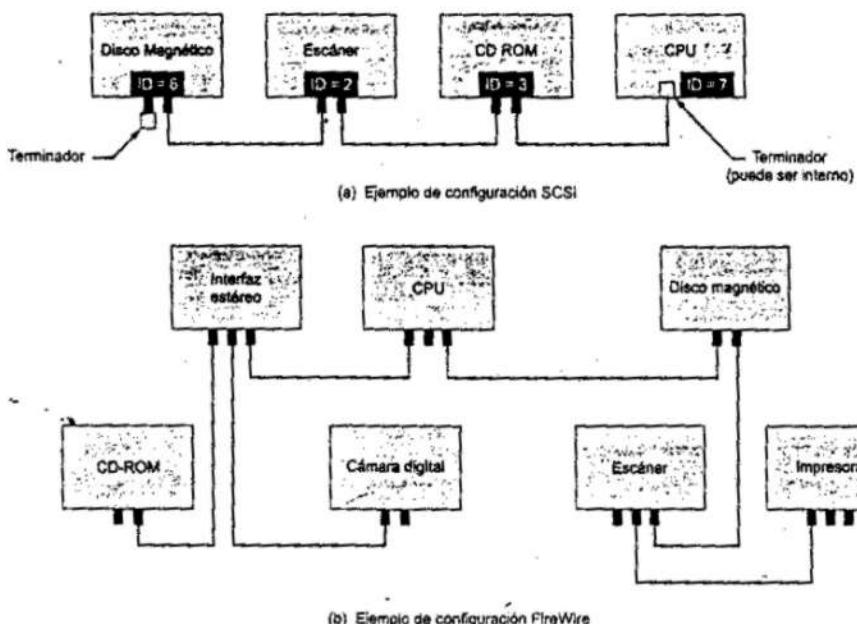


Figura 6.20. Comparación de configuraciones SCSI y FireWire.

del bus deben tener terminadores, y se debe asignar a cada dispositivo una única dirección al configurar el sistema. En el bus FireWire no hay terminadores, y el sistema realiza automáticamente la configuración para asignar direcciones. Obsérvese además, que los dispositivos en el bus FireWire no necesitan conectarse estrictamente en cadena (Daisy Chain). Se trata más bien de una configuración con estructura de árbol.

Una propiedad importante del estándar FireWire es que especifica un conjunto de protocolos de tres capas para estandarizar la forma en la que el computador anfitrión interactúa con los dispositivos periféricos a través del bus serie. La Figura 6.21 ilustra estas capas. Las tres capas son:

- **Capa física:** Define los medios de transmisión que permite el bus FireWire y las características eléctricas y de las señales de cada uno.
- **Capa de enlace:** Describe la transmisión de datos mediante paquetes.
- **Capa de transacción:** Define el protocolo de petición/respuesta que oculta a las aplicaciones los detalles de las capas inferiores del bus FireWire.

#### Capa física

La Capa física del bus FireWire especifica varios medios de transmisión alternativos y sus conectores, con diferentes propiedades físicas y de transmisión de datos. Se definen velocidades de datos desde 25 a 400 Mbps. La capa física transforma los datos binarios en las seña-

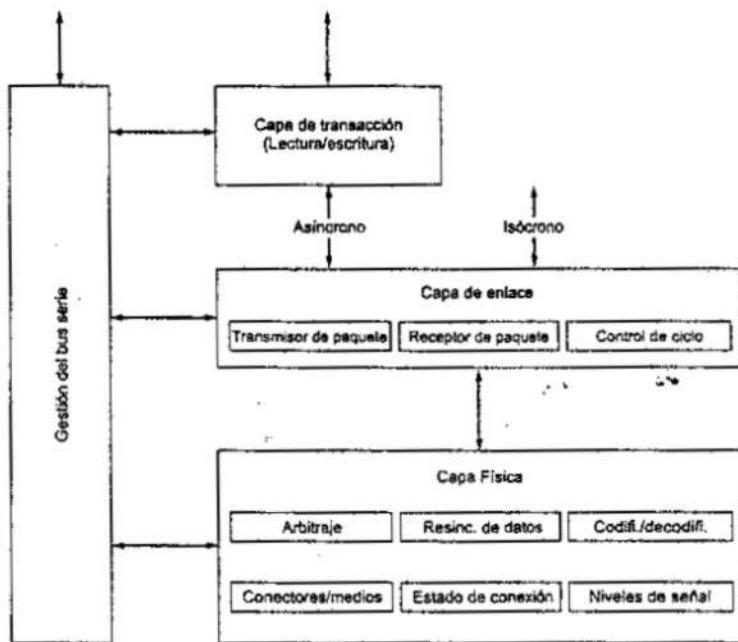


Figura 6.21. Pila del protocolo FireWire.

les eléctricas precisas para diversos medios físicos. Esta capa también proporciona el servicio de arbitraje que garantiza que sólo un dispositivo transmitirá datos en cada momento.

FireWire proporciona dos formas de arbitraje. La forma más simple se basa en la estructura de tipo árbol de los nodos del bus FireWire mencionada antes. Un caso especial de esta estructura es la conexión lineal en cadena (Daisy Chain). La capa física incluye la lógica que permite que todos los dispositivos conectados se configuren de manera que un nodo sea designado como la raíz del árbol, y los otros se organicen mediante relaciones de padre/hijo, conformando la topología de tipo árbol. Una vez que se ha establecido la configuración, el nodo raíz actúa como un árbitro central, y procesa las solicitudes de acceso al bus de manera que la primera en llegar es la primera en atenderse («first-come-first-served»). En el caso de solicitudes simultáneas, se cede el acceso al nodo con mayor prioridad natural. La prioridad natural es mayor cuanto mayor sea la cercanía al nodo raíz, y entre los nodos que están a igual distancia, es mayor para aquel que tenga el menor valor del número de identificación (ID).

El método anterior de arbitraje se complementa con dos funciones adicionales: arbitraje equitativo y arbitraje urgente. Con el arbitraje equitativo, el tiempo del bus se organiza en *intervalos de equidad*. Al comienzo de un intervalo, cada nodo activa un bit de autorización de arbitraje. Durante el intervalo, cada uno de los nodos puede competir por el acceso al bus. Cuando un nodo gana el acceso al bus, desactiva su bit de autorización de arbitraje y no puede competir de nuevo por el acceso al bus durante el intervalo actual. Este esquema hace que el arbitraje sea más equitativo, puesto que evita que uno o más dispositivos de prioridad alta puedan monopolizar el uso del bus.

Junto con este esquema de igualdad, algunos dispositivos se pueden configurar para que tengan una prioridad *urgente*. Estos nodos pueden ganar el acceso al bus varias veces durante un intervalo de equidad. En esencia, se trata de utilizar un contador en cada nodo de prioridad elevada que autoriza a estos nodos a controlar el 75 % del tiempo del bus. Por cada paquete que se transmite como no urgente, se pueden transmitir tres paquetes urgentes.

### Capa de enlace

La capa de enlace define la transmisión de los datos en forma de paquetes. Se permiten dos tipos de transmisión:

- **Asíncrona:** Se transmite un paquete, constituido por una cantidad variable de datos y varios bytes, de la capa de transacción a una dirección explícita, y se devuelve información de reconocimiento.
- **Isócrona:** Se transmite una cantidad variable de datos, mediante una secuencia de paquetes de tamaño fijo, a intervalos regulares. Esta forma de transmisión usa un direccionamiento simplificado y no utiliza reconocimiento.

La transmisión asíncrona se utiliza para datos que no necesitan tener una velocidad de transferencia fija. Tanto el esquema de arbitraje equitativo como el urgente pueden utilizarse para la transmisión asíncrona. El método implícito es el arbitraje equitativo. Los dispositivos que necesitan una fracción sustancial de la capacidad del bus, o presentan requisitos de retardo exigentes, utilizan el método de arbitraje urgente. Por ejemplo, un nodo que debe absorber datos en tiempo real a elevadas velocidades, puede utilizar arbitraje urgente cuando los buffers de datos están a la mitad de su capacidad máxima.

La Figura 6.22a describe una transacción asíncrona típica. El proceso de enviar un paquete se llama subacción (subaction). La subacción consta de cinco períodos de tiempo:

- **Secuencia de arbitraje:** Es el intercambio de señales que se precisa para ceder el control del bus a un dispositivo.
- **Transmisión de paquete:** Todo paquete incluye una cabecera que contiene los identificadores (ID) de la fuente y el destino. La cabecera también contiene información acerca del tipo de paquete, un código de redundancia cíclica (CRC) e información de los parámetros del tipo específico de paquete. Un paquete puede incluir también un bloque de datos formado por datos de usuario y otro código CRC.
- **Intervalo de reconocimiento:** Este es el retardo necesario para que en el destino se reciba y decodifique un paquete y se genere el reconocimiento.
- **Reconocimiento:** El receptor del paquete devuelve un paquete de reconocimiento con un código que indica la acción realizada por el receptor.
- **Intervalo de subacción:** Es un periodo forzoso de inactividad para asegurar que ningún nodo del bus empieza el arbitraje, antes de que el paquete de reconocimiento haya terminado de transmitirse.

En el momento en que se envía el reconocimiento, el nodo que lo está enviando tiene el control del bus. Puesto que el intercambio es una interacción de petición/respuesta entre dos nodos, el nodo que responde puede transmitir inmediatamente el paquete de respuesta sin tener que realizar una secuencia de arbitraje (Figura 6.22b).

Existe la posibilidad de acceso isócrono para los dispositivos que generan o consumen datos de manera regular, tales como los de sonido o vídeo digital. Este método asegura que

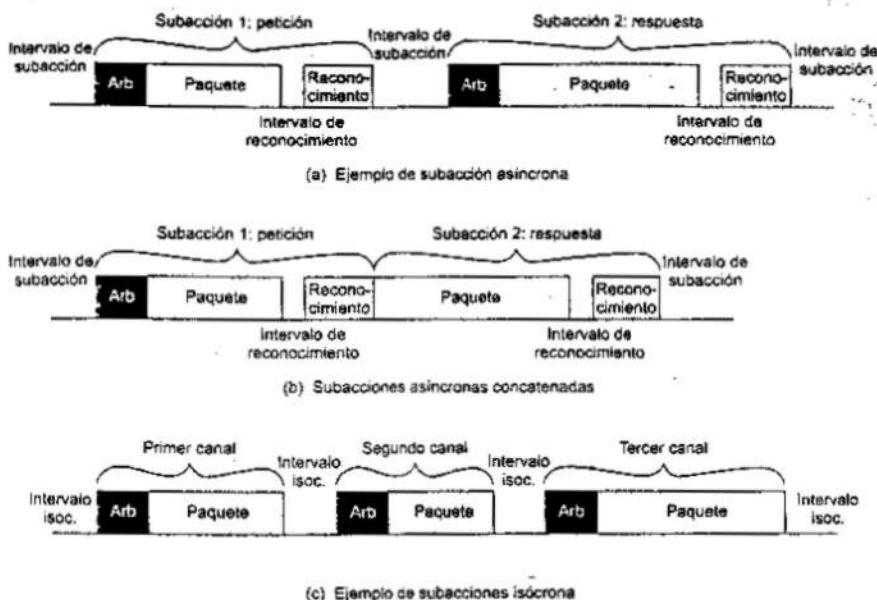


Figura 6.22. Subacciones FireWire.

los datos pueden generarse dentro de unos intervalos especificados para garantizar cierta velocidad de datos.

Para permitir una carga mixta de tráfico de datos isócronos y asíncronos, uno de los nodos se designa como *maestro de ciclo*. Periódicamente, el maestro de ciclo genera un paquete de comienzo de ciclo. Este indica a los otros nodos que se ha iniciado un ciclo isócrono. Durante este ciclo, sólo se pueden enviar paquetes isócronos (Figura 6.22c). Cada fuente de datos isócronos interviene en el arbitraje para acceder al bus. El nodo ganador transmite un paquete inmediatamente. No existe reconocimiento para este paquete y, de esta forma, otras fuentes de datos isócronos arbitran el acceso al bus inmediatamente después de que se haya transferido el paquete previo. Como resultado, existe un pequeño intervalo, determinado por los retardos del bus, entre la transmisión de un paquete y el período de arbitraje para el siguiente paquete. Este retardo, denominado «intervalo isócrono», es menor que el intervalo de subacción.

Después de que todas las fuentes isócronas hayan transmitido, el bus permanecerá inactivo el tiempo suficiente para que se produzca un intervalo de subacción. Esta es la señal para que las fuentes asíncronas puedan competir por acceder al bus. Las fuentes asíncronas pueden utilizar el bus entonces hasta el comienzo del siguiente ciclo isócrono.

Los paquetes isócronos se etiquetan con números de canal de 8 bits, que se asignan previamente mediante un diálogo entre los dos nodos que intercambian datos isócronos. La cabecera, que es más corta que la de los paquetes asíncronos, también incluye un campo de longitud de datos y un CRC para la cabecera.

**6.8. LECTURAS Y SITIOS WEB RECOMENDADOS.**

Una buena descripción de los módulos de E/S y la arquitectura de Intel, incluyendo el 82C59A y el 82C55A, puede encontrarse en [BREY97].

Dos libros que proporcionan buenas introducciones a SCSI son [SCHM97] y [NCR90]. FireWire se trata con gran detalle en [ANDE98].

**ANDE98** Anderson, D. *FireWire System Architecture*. Reading, MA: Addison-Wesley, 1998.

**BREY97** Brey, B. *The Intel Microprocessors: 8086/8066, 80186/80188, 80286, 80386, 80486, Pentium, and Pentium Processor*. Upper Saddle River, NJ: Prentice Hall, 1997.

**NCR90** NCR Corp. *SCSI: Understanding the Small Computer System Interface*. Englewood Cliffs, NJ: Prentice Hall, 1990.

**SCHM97** Schmidt, F. *The SCSI Bus and IDE Interface*. Reading, MA: Addison-Wesley, 1997.

**SITIOS WEB RECOMENDADOS:**

- **T10 Home Page:** T10 es un comité técnico del Comité Nacional de Estándares en Tecnología de la Información (National Committee on Information Technology Standards) responsable de las interfaces de bajo nivel. Su principal trabajo es la interfaz SCSI.
- **SCSI Trade Association:** Incluye información técnica y punteros a proveedores.
- **1394 Trade Association:** Incluye información técnica y punteros a proveedores de Fire-Wire.

**9. PROBLEMAS**

- 6.1. En la Sección 6.3 se enumeraron una ventaja y un inconveniente de la E/S asignada en memoria comparada con la E/S aislada. Enumere dos ventajas y dos inconvenientes más.
- 6.2. En casi todos los sistemas que tienen módulos de DMA, el acceso del módulo de DMA a memoria principal tiene más prioridad que el acceso de la CPU a memoria principal. ¿Por qué?
- 6.3. Considere el sistema de disco descrito en el Problema 5.6. Una CPU lee un sector del disco utilizando E/S con interrupciones, con una interrupción por byte. Si se necesitan 2,5 microsegundos para procesar cada interrupción, ¿qué porcentaje de tiempo pasará la CPU gestionando la E/S (desprecie el tiempo de búsqueda)?
- 6.4. Repita el Problema 6.3 utilizando DMA y asumiendo una interrupción por sector.
- 6.5. Un módulo de DMA está transfiriendo caracteres a memoria mediante robo de ciclo desde un dispositivo que transmite a 9.600 bps. La CPU capta instrucciones a una velocidad de 1 millón de instrucciones por segundo (1 MIPS). ¿En qué medida se reduce la velocidad del procesador debido a la actividad del DMA?

- 6.6. Un computador de 32 bits tiene dos canales selectores y un canal multiplexor. Cada canal selector soporta dos discos magnéticos y dos unidades de cinta magnética. El canal multiplexor tiene conectadas dos impresoras de línea, dos lectores de tarjetas y 10 terminales VDT. Suponga las siguientes velocidades de transferencia:

|                           |              |
|---------------------------|--------------|
| Unidad de disco           | 800 KBytes/s |
| Unidad de cinta magnética | 200 KBytes/s |
| Impresora de línea        | 6.6 KBytes/s |
| Lector de tarjetas        | 1.2 KBytes/s |
| VDT                       | 1 KBytes/s   |

Estime la máxima velocidad total de transferencia de E/S en el sistema.

- 6.7. Un computador está constituido por un procesador y un dispositivo D de E/S, conectado a la memoria principal M a través de un bus compartido de una palabra. El procesador puede ejecutar un máximo de  $10^6$  instrucciones por segundo. Por término medio, las instrucciones necesitan cinco ciclos máquina, tres de los cuales utilizan el bus de memoria. Una operación de lectura o escritura en memoria utiliza un ciclo máquina. Suponga que el procesador se encuentra ejecutando continuamente programas en segundo plano («background») que requieren el 95 % de la velocidad de ejecución de sus instrucciones, pero ninguna instrucción de E/S. Asuma que un ciclo del procesador es igual a un ciclo del bus. En un momento dado, el dispositivo de E/S se utiliza para transferir bloques-muy grandes de datos entre la memoria principal (M) y D.
- Si se utiliza la E/S programada, y cada transferencia de una palabra requiere que el procesador ejecute dos instrucciones, estime la máxima velocidad (en palabras por segundo) de transferencia de datos de E/S. posible a través de D.
  - Estime la misma magnitud si se utiliza DMA.
- 6.8. Una fuente de datos produce caracteres ASCII de 7 bits, y se añade un bit de paridad a cada uno. Obtenga la expresión de la máxima velocidad efectiva de transferencia de datos (velocidad de bits de datos ASCII) en una línea de R bps para las situaciones siguientes:
- Transmisión asíncrona, con 1.5 bits de parada.
  - Transmisión síncrona de bit, con una trama formada por 48 bits de control y 128 bits de información.
  - Igual que (b), pero con un campo de información de 1.024 bits.
  - Transmisión síncrona de carácter, con 9 caracteres de control por trama y 16 caracteres de información.
  - Igual que (d), con 128 caracteres de información.
- 6.9. El siguiente problema se basa en la ilustración de un mecanismo de E/S que se sugiere en [ECKE90] (Figura 6.23):

Dos niños están jugando cada uno a un lado de una valla muy alta. Uno de los niños, llamado «servidor de manzanas», tiene en su lado de la valla un manzano cargado de manzanas, y disfruta proporcionando manzanas al otro niño cuando éste se las pide. Al otro niño, llamado «comedor de manzanas», le encanta comer manzanas, pero no tiene ninguna. De hecho, debe comer manzanas con una frecuencia fija establecida (una manzana cada día, por prescripción facultativa). Si come manzanas con

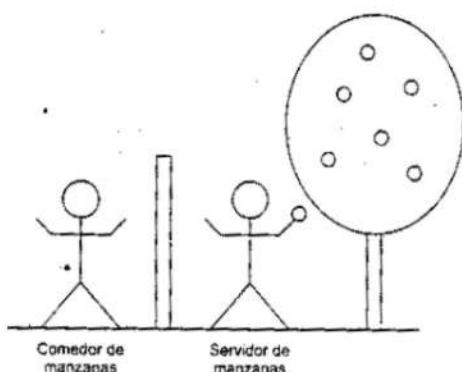


Figura 6.23. Un problema de manzanas.

una frecuencia mayor, se volvería loco. Si las come menos frecuentemente, sufriría anemia. Ninguno de los dos niños puede hablar, por lo que el problema consiste en que «servidor de manzanas» proporcione manzanas a «comedor de manzanas» con la velocidad correcta.

- Asuma que hay un reloj despertador en lo alto de la valla, y que el despertador permite seleccionar varias horas de alarma. ¿Cómo puede utilizarse el reloj para resolver el problema? Dibuje un diagrama de tiempos para ilustrar la solución.
  - Ahora asuma que no hay reloj despertador. En su lugar, «comedor de manzanas» tiene un banderín que puede agitar cada vez que necesite una manzana. Sugiera una nueva solución. ¿Sugiere de algo que «servidor de manzanas» tuviera otro banderín? Si es así, considérelo en la solución. Discuta las desventajas de esta posibilidad.
  - Ahora olvide el banderín y asuma que se dispone de una cuerda suficientemente larga. Sugiera una solución mejor a la indicada en (b), utilizando la cuerda.
- 6.10. Asuma que un procesador de 16 bits y dos de 8 bits deben conectarse a un bus del sistema. Considere los siguientes detalles:
- Todos los microprocesadores tienen el hardware necesario para cualquier tipo de transferencia: E/S programada, E/S mediante interrupciones y DMA.
  - Todos los microprocesadores tienen un bus de direcciones de 16 bits.
  - Hay dos tarjetas de memoria, de 64 Kbytes cada una, conectadas al bus. El diseñador desea que se comparta la mayor cantidad de memoria posible.
  - El bus del sistema permite un máximo de cuatro líneas de interrupción y una de DMA. Haga las suposiciones adicionales que necesite, y:
    - Establezca las especificaciones del bus en términos del número de líneas.
    - Explique cómo es la interfaz de los dispositivos indicados arriba para conectarse al bus.

Fuente: [ALEX93].

- 6.11. En un bus SCSI, cada dispositivo de E/S negocia con el computador anfitrión para determinar la velocidad de transferencia de ráfagas a utilizar (usualmente la velocidad más rápida de las mutuamente soportadas). Asuma que la velocidad máxima de transferencia a ráfagas que permite el computador anfitrión es 20 MBytes/s. Suponga que todos los dispositivos tienen los buffers suficientes para poder mantener su velocidad de transferencia sostenida, incluso cuando compiten por tiempo de bus.
- Asuma que un dispositivo de cinta con una velocidad de transferencia sostenida de 500 KB/s y una velocidad de ráfaga de 4 MB/s está conectado al SCSI. Desea conectar discos al mismo bus, cada uno con una velocidad de transferencia sostenida de 6 MB/s y una velocidad de ráfaga de 20 MB/s. ¿Cuántos dispositivos de disco puede conectar al bus si quiere que todos los dispositivos puedan funcionar simultáneamente y a toda su velocidad? ¿Cuál sería la utilización del bus en este caso? Idea: Determine el porcentaje de tiempo que necesita cada dispositivo para transferir todos sus datos. (Nota: utilice  $1K = 1.000$  en lugar de  $1.024$ ;  $1M = 1.000.000$  en lugar de  $1.048.576$ ; las aproximaciones son lo suficientemente precisas para el problema. Las velocidades de transferencia se indican normalmente utilizando números en base decimal, pero números tales como el tamaño de los buffers y los datos transferidos se expresan mediante valores en base binaria.)
  - Ahora mejore ligeramente el modelo. Asuma que el dispositivo de cinta necesita 4 ms del tiempo de bus para cada transferencia, y que el tamaño máximo de la transferencia es 64 KB. Los dispositivos de disco también necesitan 4 ms del tiempo de bus por transferencia, pero pueden transferir hasta 256 KB por petición. Vuelva a calcular la utilización del bus que hace cada dispositivo, y la utilización total. ¿Sigue siendo el bus adecuado para el número de dispositivos indicado en su respuesta a (a)?

---

## **CAPÍTULO 7**

# **El soporte del sistema operativo**

### **7.1 Conceptos básicos sobre sistemas operativos**

Objetivos y funciones del sistema operativo  
Tipos de sistemas operativos

### **7.2 Planificación**

Planificación a largo plazo  
Planificación a medio plazo  
Planificación a corto plazo

### **7.3 Gestión de la memoria**

Intercambio (Swapping)  
Definición de particiones  
Paginación  
Memoria virtual  
Buffer de traducción anticipada (Translation Lookaside Buffer)  
Segmentación

### **7.4 Gestión de memoria en el Pentium II y en el PowerPC**

Hardware de gestión de memoria en el Pentium II  
Hardware de gestión de memoria en el PowerPC

### **7.5 Lecturas y sitios Web recomendados**

### **7.6 Problemas**



- El sistema operativo (SO) es el software que controla la ejecución de los programas en el procesador y gestiona los recursos del procesador. Ciertas funciones del sistema operativo, como la planificación de procesos y la gestión de memoria, sólo pueden realizarse eficaz y rápidamente si el procesador incluye ciertos elementos hardware que den soporte al sistema operativo. Prácticamente todos los procesadores disponen de dichos elementos en mayor o menor medida, incluyendo hardware para la gestión de la memoria virtual y para la gestión de procesos. Este hardware incluye registros y buffers de propósito específico, y circuitería para realizar tareas básicas de gestión de recursos.
- Una de las funciones más importantes del sistema operativo es la planificación de procesos o tareas. El sistema operativo determina qué proceso debe ejecutarse en cada momento. Usualmente, el hardware interrumpirá un proceso en ejecución en determinados instantes para permitir que el sistema operativo tome una nueva decisión de planificación, de forma que el tiempo se reparta por igual entre los procesos.
- Otra función importante del sistema operativo es la gestión de memoria. La mayoría de los sistemas operativos actuales implementan la memoria virtual. Esta proporciona dos beneficios: (1) un proceso puede ejecutarse en memoria principal sin que todas sus instrucciones y datos se encuentren en memoria principal en un momento determinado, y (2) el espacio de memoria disponible para un programa puede exceder bastante del espacio existente en la memoria principal del sistema. Aunque el software es el encargado de la gestión de memoria, el sistema operativo aprovecha el soporte hardware que proporciona el procesador, que incluye el hardware para la paginación y la segmentación.

**A**unque este texto se centra en el hardware del computador, hay un área del software que debe considerarse: el sistema operativo del computador. El sistema operativo es un programa que administra los recursos del computador, proporciona servicios a los programadores y planifica la ejecución de otros programas. Un cierto conocimiento de los sistemas operativos es esencial para entender los mecanismos mediante los que la CPU controla el computador. En particular, los efectos de las interrupciones y de la gestión de la jerarquía de memoria se explican mejor en este contexto.

El capítulo comienza con una revisión, una breve historia de los sistemas operativos y un examen de los tipos de servicios que proporcionan al procesador. La mayor parte del capítulo considera las dos funciones del sistema operativo más relevantes para el estudio de la organización y la arquitectura del computador: la planificación y la gestión de memoria.

## 7.1 CONCEPTOS BÁSICOS SOBRE SISTEMAS OPERATIVOS

### OBJETIVOS Y FUNCIONES DEL SISTEMA OPERATIVO

Un sistema operativo es un programa que controla la ejecución de los programas de aplicación y actúa como interfaz entre el usuario y el hardware del computador. Se puede considerar que un sistema operativo tiene dos objetivos:

- **Comodidad:** Un sistema operativo hace que un computador sea más fácil y cómodo de usar.
- **Eficiencia:** Un sistema operativo permite que los recursos del computador se utilicen de forma eficiente.

Examinemos, uno por uno, estos dos aspectos del sistema operativo.

#### El sistema operativo como una interfaz usuario/computador

El hardware y el software utilizado por las aplicaciones de usuario puede verse como una jerarquía o serie de capas, tal y como se representa en la Figura 7.1. El usuario de las aplicaciones se denomina «usuario final» y, generalmente, no conoce la arquitectura del computador. Así, el usuario final tiene una visión del computador en términos de una aplicación. Esta aplicación puede utilizarse mediante un lenguaje de programación, y ha sido desarrollada por un programador de aplicaciones. Resulta evidente que si los programas de aplicación se tuvieran que desarrollar en términos del repertorio de instrucciones máquina, que son las que permiten el control directo del hardware del computador, la tarea sería de una complejidad abrumadora. Para facilitar el trabajo, existe un conjunto de programas del sistema. Algunos de estos programas se denominan «utilidades». Éstas realizan funciones utilizadas frecuentemente para ayudar en la elaboración de los programas, la gestión de los ficheros y el control de los dispositivos de E/S. Un programador hará uso de estos medios al desarrollar una aplicación, y la aplicación (mientras se está ejecutando) llamará a las utilidades para realizar ciertas funciones. El programa del sistema más importante es el sistema operativo. El sistema operativo oculta los detalles del hardware al programador y le proporciona una interfaz adecuada para utilizar el sistema. Actúa como mediador, facilitando al programador y a los programas de aplicación el acceso y el uso de los medios y servicios del sistema. -

Resumiendo, el sistema operativo usualmente proporciona servicios en las siguientes áreas:

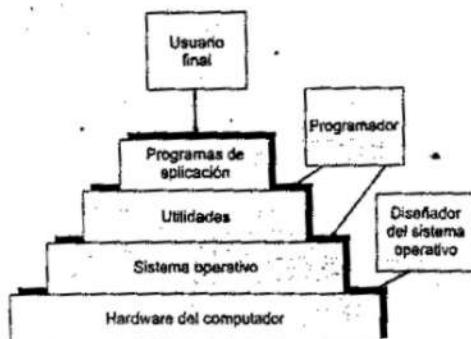


Figura 7.1. Capas y puntos de vista de un computador.

- **Creación de programas:** El sistema operativo proporciona cierta variedad de servicios y medios para ayudar al programador en la elaboración de programas. Usualmente, estos servicios son utilidades que no son propiamente parte del sistema operativo, pero se accede a ellos a través de dicho sistema operativo.
- **Ejecución de programas:** Para ejecutar un programa es preciso realizar una serie de tareas. Las instrucciones y los datos deben cargarse en memoria principal, los dispositivos de E/S y los ficheros deben iniciarse, y deben prepararse otros recursos. El sistema operativo proporciona todo eso al usuario.
- **Acceso a los dispositivos de E/S:** Cada dispositivo de E/S necesita su conjunto particular de instrucciones y señales de control para poder operar. El sistema operativo se encarga de esos detalles, para que el programador pueda pensar simplemente en términos de lecturas y escrituras.
- **Acceso controlado a los ficheros:** En el caso de ficheros, el control debe incluir el conocimiento no sólo de la naturaleza del dispositivo (disco, cinta), sino también del formato del fichero y del medio de almacenamiento. Nuevamente, el sistema operativo se ocupa de los detalles. Es más, en el caso de un sistema con múltiples usuarios simultáneos, el sistema operativo puede proporcionar mecanismos de protección para controlar el acceso a los recursos compartidos, tales como los ficheros.
- **Acceso al sistema:** En el caso de un sistema compartido o público, el sistema operativo controla el acceso al sistema como un todo y a los recursos específicos del sistema. La función de acceso debe proporcionar protección de los recursos y datos frente a los usuarios no autorizados, y debe resolver los conflictos por el acceso a los recursos compartidos.
- **Detección de errores y respuesta:** Mientras el computador está funcionando, pueden producirse una serie de errores. Entre estos están los errores hardware internos y externos, tales como los errores de memoria o los fallos o comportamiento incorrecto de dispositivos, y errores diversos del software, tales como el desbordamiento (overflow) aritmético, el intento de acceder a una posición de memoria no permitida o la incapacidad del sistema operativo para responder una petición generada por una aplicación. En cada caso, el sistema operativo debe responder de forma que se supere la condición de error con el menor impacto para las aplicaciones que se están ejecutando. La respuesta

del sistema operativo puede implicar abortar el programa que causó el error, reintentar la operación o, simplemente, notificar el error a la aplicación.

- **Contabilidad:** Un buen sistema operativo debe almacenar la estadística de uso de los distintos recursos y supervisar los parámetros de prestaciones, tales como el tiempo de respuesta. En cualquier sistema, esta información es útil para anticipar la necesidad de futuras ampliaciones y ajustes que mejoren las prestaciones del sistema. En un sistema multiusuario, esta información puede utilizarse para determinar las cantidades que deben aportar los usuarios.

### **El sistema operativo como administrador de recursos**

Un computador es un conjunto de recursos para transferir, almacenar y procesar datos y para controlar esas funciones. El sistema operativo es responsable de la administración de esos recursos.

¿Es correcto decir que es el sistema operativo el que controla la transferencia, el almacenamiento y el procesamiento de los datos? Desde un punto de vista, la respuesta es sí: al administrar los recursos del computador, el sistema operativo controla las funciones básicas del computador. Pero este control se ejerce de una forma curiosa. Normalmente, se piensa en un mecanismo de control como algo externo a aquello que se controla o, al menos, como algo que es una parte distinta y separada de lo que se controla (por ejemplo: un sistema de calefacción se controla mediante un termostato, que es algo completamente distinto al sistema de generación y distribución de calor). Este no es el caso del sistema operativo, que es un mecanismo de control inusual por dos razones:

- El sistema operativo funciona de la misma forma que el software ordinario del computador; esto es, se trata de un programa ejecutado por el procesador.
- El sistema operativo frecuentemente cede el control y depende del procesador para recuperar el control.

El sistema operativo, de hecho, no es nada más que un programa de computador. Como otros programas, proporciona instrucciones al procesador. La única diferencia se encuentra en el objetivo del programa. El sistema operativo dirige al procesador en el uso de otros recursos del sistema y en la temporización de la ejecución de otros programas. Pero para que el procesador pueda realizar esas cosas, debe dejar de ejecutar el sistema operativo y ejecutar otros programas. Así, el sistema operativo cede el control para que el procesador pueda realizar el trabajo «útil», y recupera el control posteriormente para preparar al procesador para el siguiente trozo de trabajo a realizar. Los mecanismos implicados en este proceso seclararán a lo largo del capítulo.

La Figura 7.2 indica los principales recursos que administra el sistema operativo. Una parte del sistema operativo está en la memoria principal. Ésta incluye el núcleo (kernel), que realiza las funciones más frecuentemente utilizadas por el sistema operativo, y, en un momento dado, las otras partes del sistema operativo que están actualmente en uso. El resto de la memoria principal contiene otros programas y datos. Como se verá, la asignación de este recurso (memoria principal) está controlada conjuntamente por el sistema operativo y el hardware de gestión de memoria del procesador. El sistema operativo decide cuándo un programa en ejecución puede usar un dispositivo de E/S y controla el acceso y el uso de los ficheros. El procesador es en sí un recurso, y el sistema operativo debe determinar el tiempo que el procesador dedica a la ejecución de cada programa. En el caso de un sistema multiprocesador, esta decisión debe incluir a todos los procesadores.

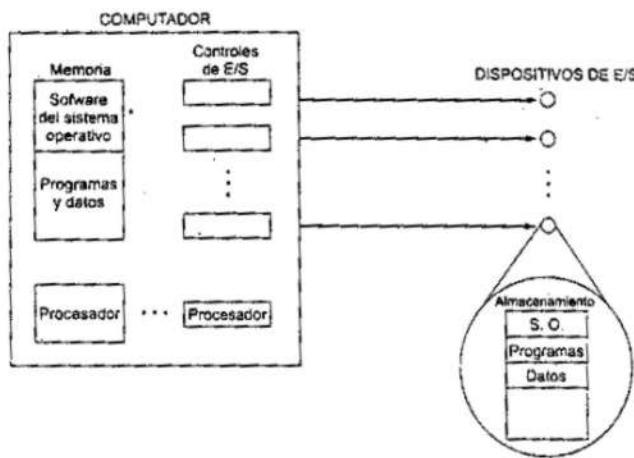


Figura 7.2. El sistema operativo como gestor de recursos.

## TIPOS DE SISTEMAS OPERATIVOS

Para distinguir entre los distintos tipos de sistemas operativos, existen ciertas características clave. Las características se agrupan en dos dimensiones distintas. La primera dimensión especifica si se trata de un sistema de colas (batch) o interactivo. En un sistema *interactivo*, el usuario/programador interactúa directamente con el computador, usualmente a través de un terminal de pantalla y teclado, para solicitar la ejecución de un trabajo o realizar una transacción. Además, el usuario puede, según la naturaleza de la aplicación, comunicarse con el computador durante la ejecución del trabajo. Un sistema *de colas* es lo opuesto a uno interactivo. El programa de usuario se introduce en una cola junto con programas de otros usuarios. Después de que el programa ha terminado, los resultados se proporcionan al usuario. Actualmente es raro encontrar sistemas de colas puros. Sin embargo, resulta útil para la descripción de los sistemas operativos contemporáneos examinar brevemente los sistemas de colas.

Otra dimensión independiente especifica si el sistema utiliza *multiprogramación* o no. Con la multiprogramación se intenta mantener el procesador ocupado tanto como sea posible, haciéndolo trabajar en más de un programa al mismo tiempo. Varios programas se cargan en la memoria, y el procesador conmuta rápidamente entre ellos. La alternativa es un sistema de *monoprogramación*, que trabaja sólo en un programa en cada momento.

### Los primeros sistemas

En los primeros computadores (desde el final de la década de los 40 a la mitad de la de los 50), el programador interactuaba directamente con el hardware del computador. Estas máquinas se accionaban desde una consola, constituida por luces indicadoras, interruptores, algún dispositivo de entrada y una impresora. Los programas en código máquina se cargaban mediante el dispositivo de entrada (por ejemplo, un lector de tarjetas). Si un error hacía dete-

nerse al programa, las luces indicaban la condición de error. El programador debía proceder a comprobar los registros y la memoria principal para determinar la causa del error. Si el programa terminaba, la salida aparecía en la impresora.

Estos primeros sistemas presentaban dos problemas fundamentales:

- **Planificación:** La mayoría de las instalaciones utilizaban una lista para reservar tiempo en la máquina. Un usuario podía reservarse normalmente espacios de tiempo múltiples de media hora. Sin embargo, podía haber reservado una hora y, en cambio, terminar en 45 minutos, lo que ocasionaba un tiempo desperdiciado en el que el computador estaba parado. Por otra parte, el usuario podía tener problemas al ejecutar el programa, no terminar en el tiempo asignado, y verse forzado a parar sin resolver el problema.
- **Tiempo de preparación:** Un único programa, llamado trabajo (job), se encargaba de cargar en memoria el compilador y el programa en lenguaje de alto nivel (programa fuente), guardar el programa compilado (programa objeto) y, después, cargar y enlazar juntos el programa objeto y las funciones comunes. Cada uno de estos pasos podía implicar montar y desmontar cintas o activar terminales de tarjetas. Por tanto, se consumía una considerable cantidad de tiempo sólo en preparar el programa para que se pudiera ejecutar.

Este modo de funcionamiento podría llamarse «procesamiento en serie», reflejando el hecho de que los usuarios acceden en serie al computador. Con el tiempo, se fueron desarrollando diversas herramientas integrantes del software del sistema que proporcionaban un procesamiento en serie más eficiente. Entre éstas están las bibliotecas de funciones usuales, enlazadores, cargadores, depuradores y rutinas de control de E/S, de las que todos los usuarios pueden disponer.

### Sistemas de colas simples

Las primeras máquinas eran muy caras y, por ello, era muy importante maximizar la utilización de la máquina. El tiempo perdido debido a la planificación y a la preparación era inaceptable.

Para mejorar la utilización, se desarrollaron los sistemas de colas sencillos. Con un sistema de este tipo, llamado *monitor*, el usuario ya no tiene acceso directo a la máquina. En cambio, el usuario envía el trabajo, en tarjetas o en cinta, a un operador del computador, que pone los trabajos en cola y sitúa toda la cola en un dispositivo de entrada al que accede el monitor.

Para comprender cómo trabaja el esquema, considerémoslo desde dos puntos de vista: el del monitor y el del procesador. Desde el punto de vista del monitor, es él el que controla la secuencia de eventos. Para que esto sea así, el monitor está siempre en la memoria principal y dispuesto para ejecutarse (Figura 7.3). Esta parte se denomina *monitor residente*. El resto del monitor consiste en utilidades y funciones comunes, que son cargadas como subrutinas del programa de usuario al iniciarse cualquier trabajo que las necesite. El monitor introduce uno a uno los trabajos desde el dispositivo de entrada (usualmente un lector de tarjetas o de cintas magnéticas). A medida que es leído, el trabajo en cuestión se sitúa en el área de programas de usuario, y se cede el control a dicho trabajo. Cuando el trabajo termina, se devuelve el control al monitor, que inmediatamente lee el siguiente trabajo. Los resultados de cada trabajo se imprimen para que el usuario pueda disponer de ellos.

Ahora consideremos esta secuencia desde el punto de vista del procesador. En cierto momento, el procesador está ejecutando instrucciones captadas de la porción de memoria que contiene al monitor. Estas instrucciones hacen que se lea el siguiente trabajo y se pase a otra

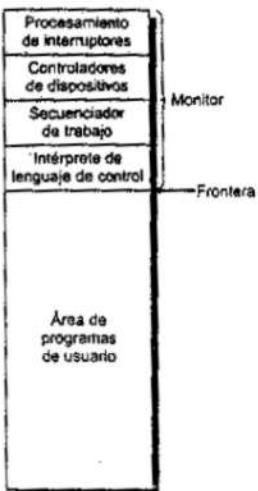


Figura 7.3. Distribución de la memoria para un monitor residente.

zona de memoria principal. Una vez que se ha introducido el trabajo, el procesador ejecutará una instrucción de salto del monitor, que hace que el procesador prosiga la ejecución en otra posición de memoria (el comienzo del programa de usuario). El procesador ejecutará entonces las instrucciones del programa de usuario hasta que encuentre una condición de final o de error. En cualquiera de los casos, el procesador capta la siguiente instrucción de ejecutar del programa monitor. Así, la frase: «el control pasa a un trabajo», simplemente significa que el procesador está captando y ejecutando instrucciones de un programa de usuario, y «el control se devuelve al monitor» significa que el procesador está captando y ejecutando instrucciones del programa monitor.

El monitor resuelve el problema de la planificación. Al existir una serie de trabajos en cola, se pueden ejecutar tan rápido como sea posible, sin que haya tiempos muertos.

¿Qué pasa con el problema de la preparación? El monitor lo resuelve también. Dentro de cada trabajo se incluyen instrucciones en un *lenguaje de control de trabajos* (JCL, Job Control Language). Se trata de un tipo especial de lenguaje de programación, utilizado para dar las instrucciones al monitor. Un ejemplo sencillo es aquél en el que el usuario está enviando un programa escrito en FORTRAN, más algunos datos que se utilizan en el programa. Cada instrucción FORTRAN y cada dato se encuentran en una tarjeta perforada distinta, o en un registro diferente de una cinta magnética. Además de las líneas FORTRAN y de datos, el trabajo incluye las instrucciones de control de trabajo, que se distinguen porque empiezan con «\$». El formato del trabajo podría ser:

```

$JOB
SFTN
• } Instrucciones FORTRAN
•
SLOAD

```

```

$RUN
• }
 • } Datos
$SEND

```

Para ejecutar este trabajo, el monitor lee la línea \$FTN y carga el compilador apropiado desde el dispositivo de almacenamiento masivo (usualmente una cinta). El compilador traduce el programa de usuario a código objeto, que se almacena en memoria o en el dispositivo de almacenamiento masivo. Si se almacena en memoria, la operación se denomina «compilar, cargar y ejecutar». Si se almacena en cinta magnética, entonces es necesario utilizar la instrucción \$LOAD. Esta instrucción es leída por el monitor, que vuelve a tomar el control tras la compilación. El monitor llama al cargador, que sitúa el programa en memoria en lugar del compilador y le transfiere el control. De esta forma, una gran parte de la memoria principal puede compartirse por subsistemas diferentes, aunque sólo uno de ellos puede estar residente y ejecutándose en cada instante.

El monitor, o sistema operativo de colas, es simplemente un programa de computador. Se basa en la posibilidad que tiene el procesador de captar instrucciones de diferentes zonas de la memoria principal para tomar y ceder el control. Además se necesita que el hardware proporcione ciertas funciones:

- **Protección de memoria:** Mientras el programa de usuario se está ejecutando, no debe alterarse el área de memoria que contiene al monitor. Si se intenta, el hardware del procesador detecta un error y transfiere el control al monitor. El monitor aborta el trabajo, imprime un mensaje de error y carga el siguiente trabajo.
- **Temporización:** Se debe utilizar un temporizador para evitar que un único trabajo monopolice el uso del sistema. El temporizador se actualiza al comienzo de cada trabajo. Si el tiempo termina, se produce una interrupción y el control vuelve al monitor.
- **Instrucciones privilegiadas:** Ciertas instrucciones, que se denominan «privilegiadas» sólo pueden ser ejecutadas por el monitor. Entre éstas están las instrucciones de E/S, para que el monitor tenga el control de todos los dispositivos de E/S. Esto impide, por ejemplo, que un programa de usuario lea accidentalmente las instrucciones del siguiente trabajo. Si un programa de usuario desea realizar una E/S, debe solicitar al monitor que realice la operación por él. Si el procesador encuentra una instrucción privilegiada mientras ejecuta un programa de usuario, el hardware del procesador lo considera un error y transfiere el control al monitor.
- **Interrupciones:** Los primeros modelos de computadores no disponían de esta capacidad. Esta característica proporciona al procesador más flexibilidad para ceder y recuperar el control de los programas de usuario.

El tiempo del procesador se alterna entre la ejecución de los programas de usuario y la ejecución del monitor. Se han sacrificado dos cosas: parte de la memoria principal está ocupada por el monitor, y parte del tiempo de la máquina es consumido por el monitor. Ambas cosas constituyen una cierta penalización («overhead»). Incluso con esta penalización, los sistemas de colas sencillos mejoran la utilización del computador.

### Sistemas de colas multiprogramados

Incluso con la sucesión automática de trabajos que proporcionan los sistemas de colas sencillos, el procesador está parado a menudo. El problema surge porque los dispositivos de E/S son lentos en comparación con el procesador. La Figura 7.4 describe una situación típica.

|                               |                                         |
|-------------------------------|-----------------------------------------|
| Ler un registro               | 0,0015 segundos                         |
| Ejecutar 100 instrucciones    | 0,0001 segundos                         |
| Escribir un registro          | 0,0015 segundos                         |
| <b>TOTAL</b>                  | <b>0,0031 segundos</b>                  |
| Porcentaje de uso de la CPU = | $\frac{0,0001}{0,0031} = 0,032 = 3,2\%$ |

Figura 7.4. Ejemplo de utilización del sistema.

El cálculo se refiere a un programa que procesa un fichero de registros y ejecuta, por término medio, 100 instrucciones máquina por registro. En este ejemplo el computador pasa alrededor del 96 % de su tiempo esperando que los dispositivos de E/S terminen de transferir datos. La Figura 7.5a ilustra esta situación. El procesador consume cierto tiempo ejecutando instrucciones hasta que llega a una instrucción de E/S. Entonces debe esperar hasta que esa instrucción de E/S concluya para continuar.

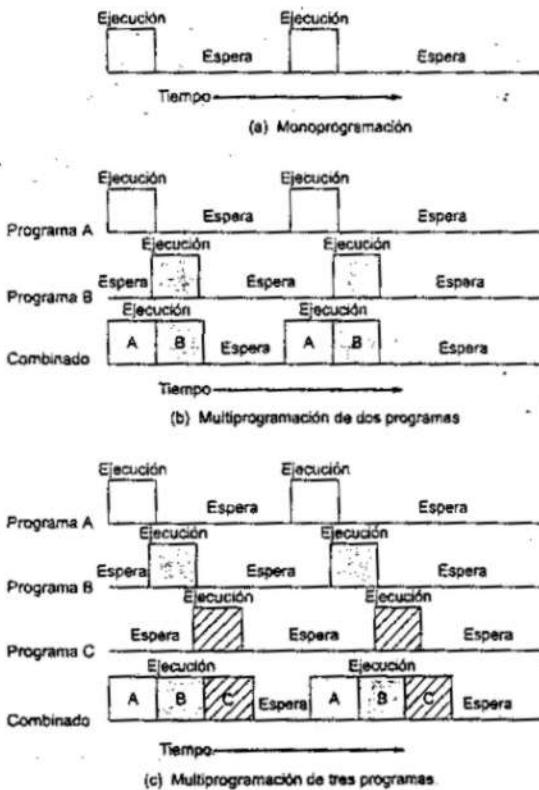


Figura 7.5. Ejemplo de multiprogramación.

Tabla 7.1. Atributos de ejecución de un programa ejemplo

|                      | Trabajo 1       | Trabajo 2  | Trabajo 3  |
|----------------------|-----------------|------------|------------|
| Tipo de trabajo:     | Muchos cálculos | Muchas E/S | Muchas E/S |
| Duración:            | 5 min.          | 15 min.    | 10 min.    |
| Memoria necesaria:   | 50K             | 100K       | 80K        |
| ¿Necesita disco?     | No              | No         | Sí         |
| ¿Necesita terminal?  | No              | Sí         | No         |
| ¿Necesita impresora? | No              | No         | Sí         |

Esta ineficiencia se puede evitar. Se ha indicado que debe haber memoria suficiente para dar cabida al sistema operativo (monitor residente) y a un programa de usuario. Supóngase que hay sitio para el sistema operativo y dos programas de usuario. En ese caso, cuando un trabajo necesita esperar debido a una E/S, el procesador puede conmutar al otro trabajo, que posiblemente no estará esperando una E/S (Figura 7.5b). Es más, se podría expandir la memoria para disponer de tres, cuatro o más programas entre los que conmutar (Figura 7.5c). Este proceso se conoce como multiprogramación o multitarea. Es el tema central de los sistemas operativos modernos.

Para ilustrar el beneficio de la multiprogramación, utilizaremos un ejemplo. Considérese un computador con una memoria disponible (no utilizada por el sistema operativo) de 256K palabras, un disco, un terminal y una impresora. Se envían al mismo tiempo tres programas, Trabajo1, Trabajo2 y Trabajo3, para su ejecución. Sus atributos se enumeran en la Tabla 7.1. Asumimos requisitos mínimos de procesador para Trabajo2 y Trabajo3, y un uso continuo del disco y la impresora por parte de Trabajo3. En un entorno de colas simple, estos trabajos se ejecutarían sucesivamente uno tras otro. Así, Trabajo1 termina en 5 minutos. Trabajo2 debe esperar a que los 5 minutos hayan pasado, y termina 15 minutos después. Trabajo3 empieza después de 20 minutos y termina 30 minutos después de que se envíara. La utilización media de los recursos, el rendimiento y los tiempos de respuesta se muestran en la columna de monoprogramación de la Tabla 7.2. La utilización dispositivo por dispositivo se ilustra en la Figura 7.6. Es evidente que hay una importante infroutilización de todos los recursos cuando se promedia su uso en el periodo de tiempo de 30 minutos.

Ahora suponga que los trabajos se ejecutan concurrentemente bajo un sistema operativo con multiprogramación. Puesto que hay poca competencia entre los trabajos por los recursos, los tres pueden ejecutarse en un tiempo casi mínimo, al coexistir en el computador con el resto (asumiendo que se asigna a Trabajo2 y Trabajo3 tiempo de procesador suficiente para mantener activas sus operaciones de entrada y salida). El trabajo Trabajo1 todavía necesitará 5 minutos para terminar, pero al final de ese tiempo, Trabajo2 se habrá completado en un tercio y Trabajo3 en la mitad. Los tres trabajos habrán terminado en un tiempo de 15 minu-

Tabla 7.2. Efectos de la multiprogramación sobre la utilización de recursos

|                              | Monoprogramación | Multiprogramación |
|------------------------------|------------------|-------------------|
| Utilización del procesador:  | 17 %             | 33 %              |
| Utilización de la memoria:   | 30 %             | 57 %              |
| Utilización del disco:       | 33 %             | 67 %              |
| Utilización de la impresora: | 33 %             | 67 %              |
| Tiempo transcurrido:         | 30 min.          | 15 min.           |
| Rendimiento:                 | 6 trabajos/hora  | 12 trabajos/hora  |
| Tiempo de respuesta medio:   | 18 min.          | 10 min.           |

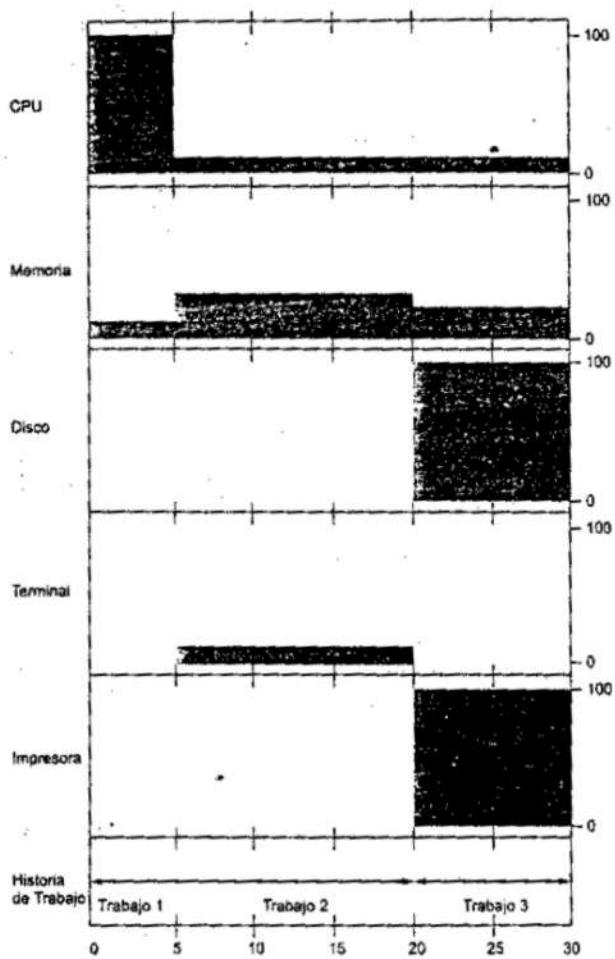


Figura 7.6. Histograma de utilización en el caso de monoprogramación.

tos. La mejora es evidente si se examina la columna de multiprogramación de la Tabla 7.2, obtenida a partir del histograma de la Figura 7.7.

Igual que en un sistema de colas sencillo, un sistema de colas multiprogramado es un programa que se apoya en ciertas características del hardware del computador. La característica más notable de utilidad para la multiprogramación es el soporte hardware para las interrupciones y el DMA. Con las E/S mediante interrupciones o mediante DMA, la CPU puede lanzar una orden de E/S para un trabajo y continuar ejecutando otro trabajo mientras el controlador de dispositivo se encarga de realizar la E/S. Cuando se completa la operación de

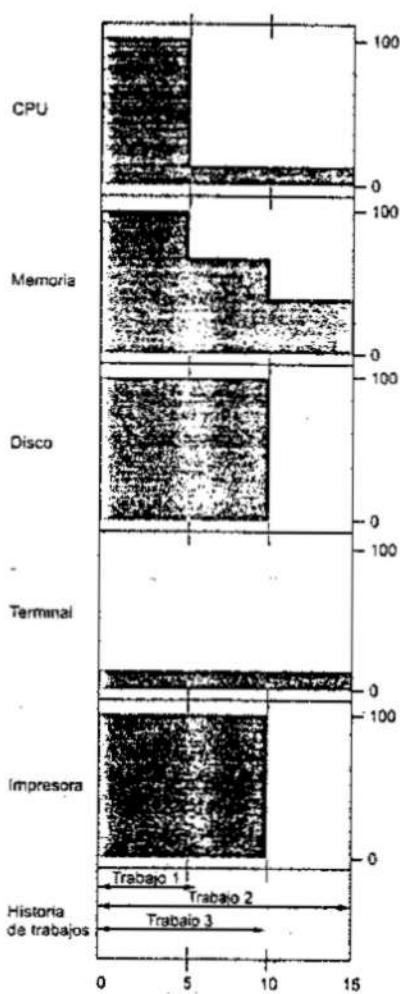


Figura 7.7. Histograma de utilización en el caso de mutiprogramación.

E/S, la CPU es interrumpida, y el control pasa a un programa de gestión de interrupciones del sistema operativo. Entonces, el sistema operativo pasa el control a otro trabajo.

Los sistemas operativos multiprogramados son bastante sofisticados en comparación con los sistemas de un sólo programa, o monoprogramados. Para tener varios trabajos listos para ejecutarse, deben mantenerse en memoria, precisándose una cierta gestión de la memoria. Además, si varios trabajos están listos para ejecutarse, el procesador debe decidir cuál de ellos se ejecuta, lo que implica utilizar algún algoritmo de planificación. Estos conceptos se discuten más adelante en este capítulo.

Tabla 7.3. Multiprogramación con colas frente a tiempo compartido

|                                                   | Multiprogramación con colas                                                           | Tiempo compartido                          |
|---------------------------------------------------|---------------------------------------------------------------------------------------|--------------------------------------------|
| Objetivo principal                                | Maximizar la utilización del procesador                                               | Minimizar el tiempo de respuesta           |
| Fuente de instrucciones para el sistema operativo | Instrucciones de un lenguaje de control de trabajos que proporciona el propio trabajo | Órdenes introducidas a través del terminal |

### Sistemas de tiempo compartido

Con el uso de la multiprogramación, el procesamiento en colas puede ser bastante eficiente. Sin embargo, para muchos trabajos es deseable disponer de un modo en el cual el usuario interactúe directamente con el computador. De hecho, para algunos trabajos, tales como el procesamiento de transacciones, es esencial el modo interactivo.

Hoy en día, los requisitos para el procesamiento interactivo pueden ser, y a menudo son, satisfechos por un microcomputador. Esta opción no era posible en los sesenta, cuando la mayoría de los computadores eran grandes y costosos. En su lugar, se desarrolló el tiempo compartido.

Igual que la multiprogramación permite que el procesador ejecute varios trabajos de la cola en un intervalo de tiempo, también se puede hacer que ejecute varios trabajos interactivos. En este caso, la técnica se llama **tiempo compartido**, puesto que el tiempo del procesador se comparte entre varios usuarios. En un sistema de tiempo compartido, varios usuarios acceden simultáneamente al sistema a través de terminales mientras el sistema operativo alterna la ejecución de fragmentos o ráfagas de cómputo correspondientes a cada usuario. Así, si hay  $n$  usuarios que solicitan servicio al mismo tiempo, cada usuario sólo aprovechará, por término medio, una fracción igual a  $1/n$  de la velocidad efectiva del procesador, y eso sin contar el tiempo dedicado al sistema operativo. No obstante, dado el tiempo relativamente elevado de reacción humana, el tiempo de respuesta de un sistema diseñado correctamente debería ser comparable al que proporciona un computador dedicado.

Tanto las colas multiprogramadas como el tiempo compartido, usan multiprogramación. Las diferencias esenciales se enumeran en la Tabla 7.3.

## PLANIFICACIÓN

La clave de la multiprogramación es la planificación. De hecho, usualmente implica tres tipos de planificación (Tabla 7.4). Las describiremos aquí, pero primero introduciremos el concepto de proceso. Este término fue utilizado por primera vez por los diseñadores de Multics en los años sesenta. En cierta forma, se trata de un término más general que «trabajo». Se han dado muchas definiciones del término «proceso», entre ellas:

- Un programa en ejecución
- El «espíritu animado» de un programa
- Aquella entidad a la que se asigna un procesador

El concepto se aclarará a medida que avancemos.

**Tabla 7.4.** Tipos de planificación

|                             |                                                                                                                   |
|-----------------------------|-------------------------------------------------------------------------------------------------------------------|
| Planificación a largo plazo | Decide si se añade al conjunto de programas a ejecutar.                                                           |
| Planificación a medio plazo | Decide si se añade al número de procesos que están total o parcialmente en memoria principal.                     |
| Planificación a corto plazo | Decide cuál de los procesos disponibles ejecutará el procesador.                                                  |
| Planificación de E/S        | Decide el procesador cuya solicitud de E/S pendiente se va a atender por parte del dispositivo de E/S disponible. |

## PLANIFICACIÓN A LARGO PLAZO

El planificador a largo plazo determina qué programas se admiten para ser procesados en el sistema. De esta manera, este planificador controla el grado de multiprogramación (número de procesos en memoria). Una vez admitido, un trabajo o programa de usuario pasa a ser un proceso, y se añade a una cola asociada al planificador a corto plazo. En algunos sistemas, un proceso nuevo comienza a partir de una sustitución en el intercambio (swapping), en cuyo caso se añade a la cola del planificador a medio plazo.

En un sistema de colas, o en la parte de colas de un sistema operativo de uso general, los trabajos nuevos que se envían pasan al disco y se mantienen en una cola. El planificador a largo plazo selecciona trabajos de esta cola cuando puede. Esto implica tomar dos decisiones. En primer lugar, el planificador debe decidir si el sistema operativo puede aceptar uno o más procesos adicionales. En segundo lugar, el planificador debe decidir qué trabajo o trabajos acepta y transforma en procesos. Los criterios que se utilizan deben incluir la prioridad, el tiempo de ejecución esperado y las E/S que se requieren.

Para los programas interactivos en un sistema de tiempo compartido, se genera una solicitud de proceso cuando un usuario intenta conectarse al sistema. Los usuarios en tiempo compartido no se introducen en una cola para mantenerse esperando a que el sistema los acepte. Por el contrario, el sistema operativo aceptará a todos los usuarios autorizados hasta que el sistema se sature. En ese momento, si se produce una solicitud de conexión, se responde con un mensaje que indica que el sistema está completo, y el usuario debe intentar la conexión de nuevo, pasado un cierto tiempo.

## PLANIFICACIÓN A MEDIO PLAZO

La planificación a medio plazo es parte de la función de intercambio, descrita en la Sección 7.3. Usualmente, la decisión de intercambiar un proceso se toma en función del grado de multiprogramación que se desea mantener. En un sistema que no utilice memoria virtual, la gestión de la memoria también debe considerarse por el planificador a medio plazo, y en las decisiones tomadas en el intercambio, deben tenerse en cuenta las necesidades de memoria de los procesos intercambiados.

## PLANIFICACIÓN A CORTO PLAZO

El planificador a largo plazo se ejecuta de manera relativamente poco frecuente, y toma las decisiones más genéricas sobre si aceptar un nuevo proceso o no, y qué proceso aceptar. El planificador a corto plazo, conocido también como distribuidor (dispatcher), se ejecuta frecuentemente, y toma la decisión más específica sobre qué trabajo se ejecuta a continuación.

### Estados de los procesos

Para comprender el funcionamiento del planificador a corto plazo, necesitamos considerar el concepto de estado de un proceso. Durante el tiempo de vida de un proceso, la situación en que se encuentra cambiará un cierto número de veces. Su situación en cada instante de tiempo se denomina **estado**. El término «estado» se utiliza porque tiene la connotación de que existe cierta información que define la situación en que se encuentra el proceso en ese momento. Usualmente, se definen cinco estados para un proceso (Figura 7.8):

- **Nuevo (New):** El planificador de alto nivel admite un programa, pero todavía no está preparado para ejecutarse. El sistema operativo iniciará el proceso, pasándolo al estado preparado.
- **Preparado (Ready):** El proceso está preparado para ejecutarse y se encuentra esperando acceso al procesador.
- **En ejecución (Running):** El proceso está siendo ejecutado por el procesador.
- **En espera (Waiting):** El proceso ha suspendido su ejecución al estar esperando algún recurso del sistema, tal como una E/S.
- **Parado (Halted):** El proceso ha terminado, y será eliminado por el sistema operativo.

Para cada proceso del sistema, el sistema operativo debe mantener información de su estado, indicando la situación en que se encuentra el proceso y cualquier información adicional necesaria para la ejecución del mismo. Para eso, cada proceso se representa en el sistema operativo mediante un *bloque de control de proceso* (Figura 7.9), que usualmente está constituido por:

- **Identificador:** Cada proceso en curso tiene un identificador único.
- **Estado:** El estado actual del proceso (nuevo, preparado, etc.).
- **Prioridad:** El nivel de prioridad relativo.
- **Contador de programa:** La dirección de la siguiente instrucción del programa a ejecutar.
- **Punteros a memoria:** Las posiciones de memoria de inicio y final del proceso.
- **Datos de contexto:** Son los datos de los registros del procesador cuando el proceso se está ejecutando, y se discutirán en la Parte Tercera. Por ahora, es suficiente decir que



Figura 7.8. Modelo de proceso de cinco estados.

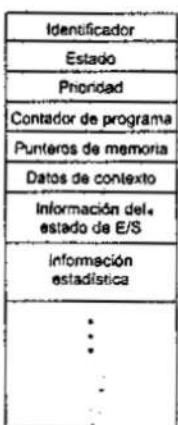


Figura 7.9. Bloque de control de procesos.

estos datos representan el «contexto» del proceso. El contexto, junto con el contador de programa, se guarda cuando el procesador abandona el estado «en ejecución». El procesador los recupera cuando reanuda la ejecución del proceso.

- **Información del estado de las E/S:** Incluye las solicitudes de E/S pendientes, los dispositivos de E/S (por ejemplo, cintas) asignados al proceso, la lista de ficheros asignados al proceso, etc.
- **Información estadística:** Puede incluir el tiempo total y el tiempo de procesador utilizados, los límites de tiempo, los datos de las cuentas, etc.

Cuando el planificador acepta un nuevo trabajo o solicitud de ejecución de un usuario, crea un bloque de control de procesos en blanco, y sitúa en él al proceso asociado en el estado nuevo. Una vez que el sistema haya completado correctamente el bloque de control de proceso, el proceso se transfiere al estado «preparado».

### Técnicas de planificación

Para entender cómo el sistema operativo realiza la planificación de los trabajos en memoria, empezaremos considerando el ejemplo de la Figura 7.10. La figura muestra cómo se divide la memoria principal en un instante de tiempo dado. El núcleo del sistema operativo, por supuesto, siempre está residente. Además, hay un cierto número de procesos activos, por ejemplo A y B, a cada uno de los cuales se les asigna una porción de memoria.

Empezamos en un instante de tiempo dado, cuando el proceso A está ejecutándose. El procesador toma las instrucciones del programa contenido en la partición de memoria de A. En un instante posterior, el procesador deja de ejecutar instrucciones de A y empieza a ejecutar instrucciones del área del sistema operativo. Esto puede suceder debido a una de estas tres razones:

1. El proceso A genera una llamada a un servicio (por ejemplo, una solicitud de E/S) del sistema operativo. La ejecución de A se suspende hasta que el sistema operativo ha completado el servicio solicitado.

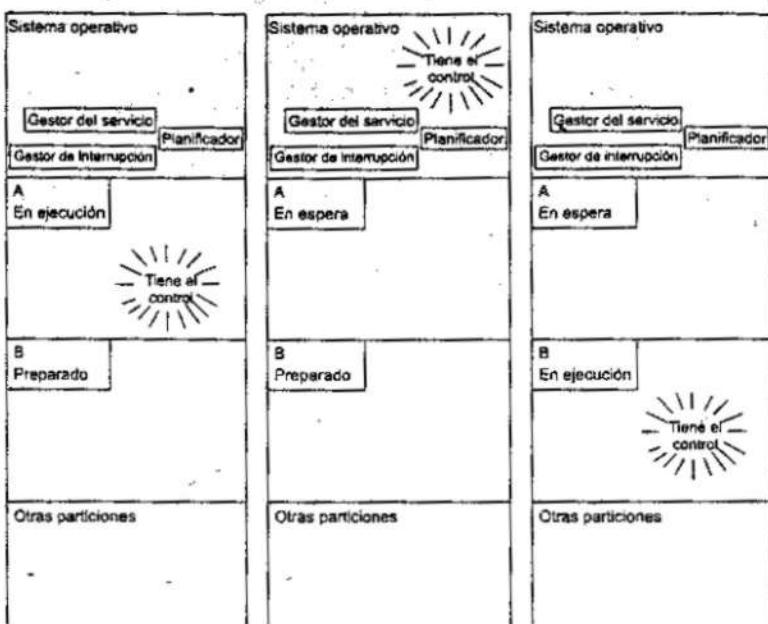


Figura 7.10. Ejemplo de planificación.

2. El proceso A origina una *interrupción*. Una interrupción es una señal generada por el hardware que se envía al procesador. Cuando se detecta la señal, el procesador deja de ejecutar A y pasa al gestor de interrupciones incluido en el sistema operativo. Hay una cierta variedad de eventos de A que pueden ocasionar la interrupción. Por ejemplo, un error tal como el intento de ejecutar una instrucción privilegiada. También se genera una interrupción cuando se agota el tiempo asignado al proceso; para evitar que un proceso monopolice al procesador, cada proceso dispone del procesador sólo durante un corto periodo de tiempo.
3. Algun hecho no relacionado con el proceso A, que requiere atención, origina una interrupción. Por ejemplo cuando se completa una operación de E/S.

En cualquier caso, el resultado es el siguiente: El procesador guarda los datos del contexto actual y el contador de programa de A en el bloque de control del proceso A, y empieza a ejecutar el sistema operativo. El sistema operativo puede realizar alguna actividad, como iniciar una operación de E/S. Entonces, la porción del sistema operativo correspondiente al planificador a corto plazo decide el proceso que se ejecuta a continuación. En este ejemplo, se elige B. El sistema operativo hace que se restauren en el procesador los datos del contexto de B, y se prosigue con la ejecución de B donde se dejó.

Este sencillo ejemplo aclara el funcionamiento básico del planificador a corto plazo. La Figura 7.11 muestra los elementos del sistema operativo que intervienen de manera más importante en la multiprogramación y en la planificación de procesos. El sistema operativo recibe el control del procesador al ejecutarse el gestor de interrupciones, si se produce una

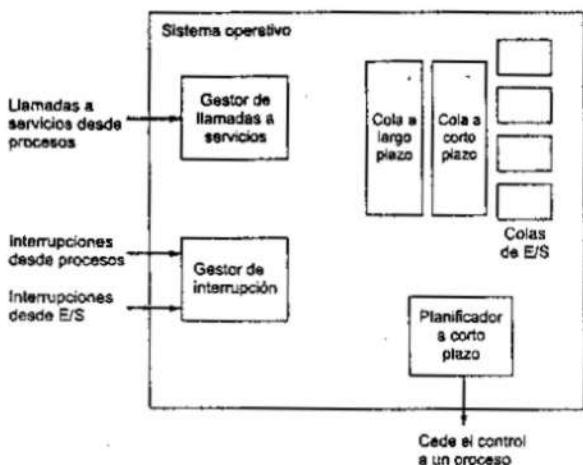


Figura 7.11. Elementos clave de un sistema operativo con multiprogramación.

interrupción, y al ejecutarse el gestor de llamadas de servicio, si se solicita un servicio. Una vez se ha servido la llamada o la interrupción, vuelve a intervenir el planificador a corto plazo, que selecciona un proceso para su ejecución.

Para realizar este trabajo, el sistema operativo utiliza un cierto número de colas. Cada cola es simplemente una lista de espera de procesos que necesitan un recurso. La *cola a largo plazo* es una lista de trabajos que esperan utilizar el sistema. Cuando las condiciones lo permitan, el planificador a largo plazo asignará memoria y creará un proceso para uno de los elementos que esperan en la cola. La *cola a corto plazo* contiene a los procesos que se encuentran en estado «preparado». Cada uno de estos procesos podría ser el siguiente en utilizar el procesador. Depende de cual sea el que elija el planificador a corto plazo. Generalmente, esto se hace mediante un algoritmo de turno rotatorio (round-robin), cediendo el tiempo a cada proceso por turnos. También se pueden usar niveles de prioridad. Finalmente, hay una *cola de E/S* para cada dispositivo de E/S. Más de un proceso puede solicitar el uso del mismo dispositivo de E/S. Todos los procesos que esperan para utilizar cada dispositivo se introducen en la cola de ese dispositivo.

La Figura 7.12 sugiere cómo los procesos avanzan en el computador bajo el control del sistema operativo. Cada solicitud de proceso (desde los trabajos en cola a los trabajos interactivos) se sitúa en la cola a largo plazo. A medida que los recursos están disponibles, una solicitud de proceso se hace proceso y pasa al estado preparado, situándose en la coila de corto plazo. Alternativamente, el procesador ejecuta instrucciones del sistema operativo y de los procesos de usuario. Mientras el sistema operativo dispone del control, decide qué proceso de la cola de corto plazo debería ejecutarse a continuación. Cuando el sistema operativo ha terminado sus tareas inmediatas, devuelve el procesador a los procesos elegidos.

Como se mencionó anteriormente, un proceso en ejecución puede suspenderse por varias razones. Si se suspende porque el proceso solicita una E/S, se sitúa en la cola de E/S apropiada. Si se suspende porque ha transcurrido el tiempo que se le asignó, o porque el sistema operativo debe atender alguna tarea urgente, se pone en estado preparado y se devuelve a la cola a corto plazo.

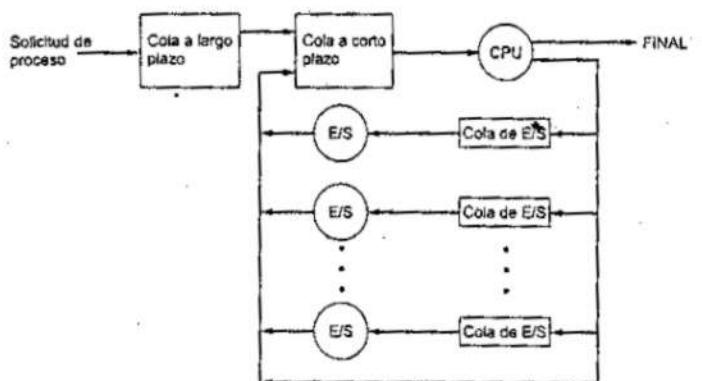


Figura 7.12. Representación de un diagrama de colas para la planificación del procesador.

Finalmente, mencionaremos que el sistema operativo también gestiona las colas de E/S. Cuando finaliza una operación de E/S, el sistema operativo suprime de la cola de E/S el proceso atendido y lo sitúa en la cola de corto plazo. Después selecciona otro proceso en estado de espera (si lo hay) y actúa sobre el dispositivo de E/S correspondiente para que satisfaga la solicitud del proceso.

### 3.2. GESTIÓN DE LA MEMORIA

En un sistema de monoprogramación, la memoria principal se divide en dos partes: una parte para el sistema operativo (el monitor residente), y otra parte para el programa que se está ejecutando. En un sistema multiprogramado, la parte de «usuario» de la memoria además debe subdividirse para dar cabida a los distintos procesos. La tarea de subdivisión la realiza dinámicamente el sistema operativo, y se conoce como *gestión de memoria*.

Una gestión eficiente de la memoria es vital en un sistema multiprogramado. Si hay pocos procesos en memoria, puede ocurrir que todos los procesos estén esperando completar una E/S, con lo que el procesador permanecerá inactivo. En consecuencia, la memoria debe asignarse eficientemente, para situar en memoria tantos procesos como sea posible.

### INTERCAMBIO (SWAPPING)

Volviendo a la Figura 7.12, se han discutido tres tipos de colas: la cola a largo plazo, para solicitar procesos nuevos; la cola a corto plazo, con los procesos preparados para utilizar el procesador; y las distintas colas de E/S de los procesos que no están preparados para usar el procesador. Recuérdese que la razón última de estos mecanismos era que las actividades de E/S son mucho más lentas que el cálculo y que, por consiguiente, el procesador en un sistema con monoprogramación está la mayor parte del tiempo parado.

Sin embargo, el esquema de la Figura 7.12 no resuelve el problema por completo. Es verdad que, en este caso, la memoria contiene múltiples procesos, y que el procesador puede conmutar a otro proceso cuando el proceso en curso tenga que esperar. Pero el procesador es

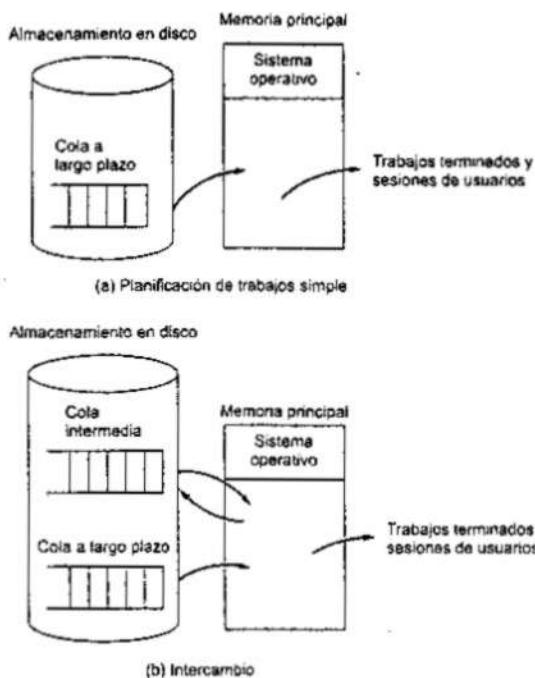


Figura 7.13. Uso del intercambio («swapping»).

tan rápido en comparación con las E/S. que puede ser frecuente que *todos* los procesos de la memoria estén esperando una E/S. Por eso, incluso con la multiprogramación, un procesador puede estar parado la mayor parte del tiempo.

¿Qué se puede hacer? La memoria principal podría ampliarse y, así, ser capaz de dar cabida a más procesos. Pero hay dos problemas en esta solución. Primero, incluso hoy día la memoria principal es cara. Segundo, la necesidad de memoria de los programas han crecido tan rápido como ha caído el costo de la memoria. Por eso, una memoria mayor origina procesos mayores, no más procesos.

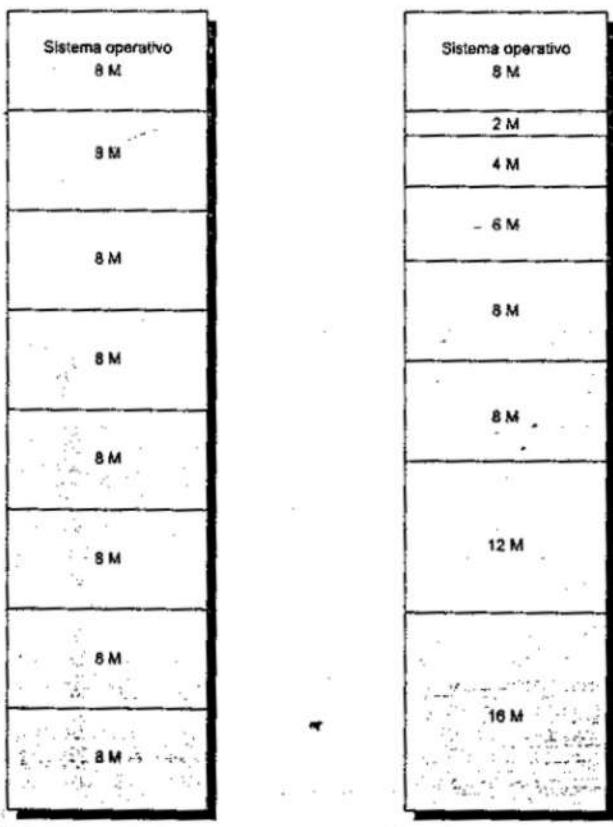
Otra solución es el *intercambio* (swapping), representado en la Figura 7.13. Tenemos una cola a largo plazo de solicitudes de proceso, usualmente almacenadas en disco. Estas solicitudes se traen a memoria, una a una, a medida que hay espacio disponible. Conforme terminan, los procesos se sacan de la memoria principal. Ahora, podría ocurrir que ninguno de los procesos en la memoria principal esté en el estado preparado (por ejemplo, todos están esperando una operación de E/S). En lugar de permanecer parado, el procesador *intercambia* uno de esos procesos, situándolo en el disco en una *cola intermedia*. Esta es una cola de procesos existentes que se han sacado temporalmente de memoria. El sistema operativo trae entonces otro proceso de la cola intermedia o acepta una nueva petición de proceso de la cola de largo plazo. La ejecución continúa con el proceso recientemente activado.

El intercambio es de hecho una operación de E/S y, por consiguiente, existe la posibilidad de empeorar el problema más que de solucionarlo. No obstante, puesto que la E/S en disco

es generalmente la operación de E/S más rápida (comparada con la E/S en cinta o mediante impresora), usualmente el intercambio mejora las prestaciones. Un esquema más sofisticado, que implica el uso de la memoria virtual, mejora las prestaciones con respecto al intercambio simple. Esto se discutirá en breve, pero primero debemos proporcionar los fundamentos explicando la definición de particiones y la paginación.

## DEFINICIÓN DE PARTICIONES

El esquema más simple para definir particiones en la memoria disponible es utilizar *particiones de tamaño fijo*, como muestra la Figura 7.14. Observe que, aunque las particiones son de tamaño fijo, no todas tienen igual tamaño. Cuando un proceso se introduce en memoria, se sitúa en la partición disponible más pequeña que puede incluirlo.



(a) Particiones de igual tamaño

(b) Particiones de distinto tamaño

Figura 7.14. Ejemplo de particiones fijas de una memoria de 64 Mbytes.

Incluso con el uso de particiones de distintos tamaños, se desperdiciará memoria. En la mayoría de los casos, un proceso no necesitará exactamente la memoria que proporciona una partición. Por ejemplo, un proceso que precise 3 Mbytes de memoria se podría situar en la partición de 4 M de la Figura 7.14b, desperdimando 1 M que podría utilizarse para otro proceso.

Una posibilidad más eficiente consiste en utilizar *particiones de tamaño variable*. Cuando un proceso se introduce en memoria, se le asigna exactamente la memoria que necesita, y no más. En la Figura 7.15 se muestra un ejemplo utilizando 1 Mbyte de memoria principal. Inicialmente, salvo por el sistema operativo, la memoria principal está vacía (a). Los primeros tres procesos se cargan, empezando por donde termina el sistema operativo y ocupando justo el espacio necesario para cada proceso (b), (c) y (d). Esto deja un «hueco» al final de la memoria que es demasiado pequeño para un cuarto proceso. En cierto instante, ninguno de los procesos de memoria está preparado. El sistema operativo saca de memoria al proceso 2 (e),

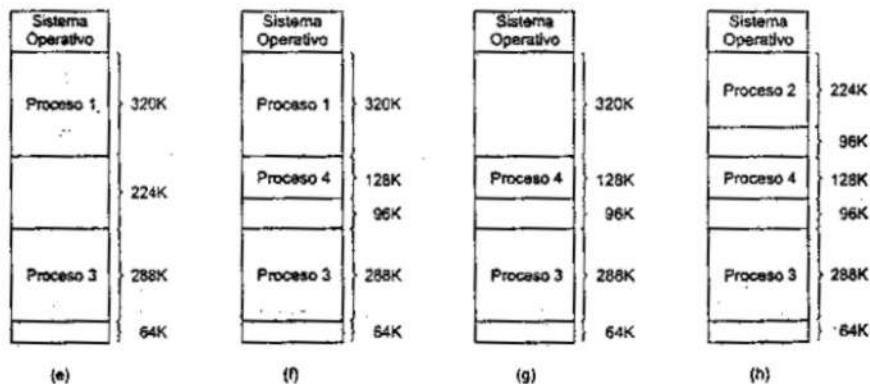
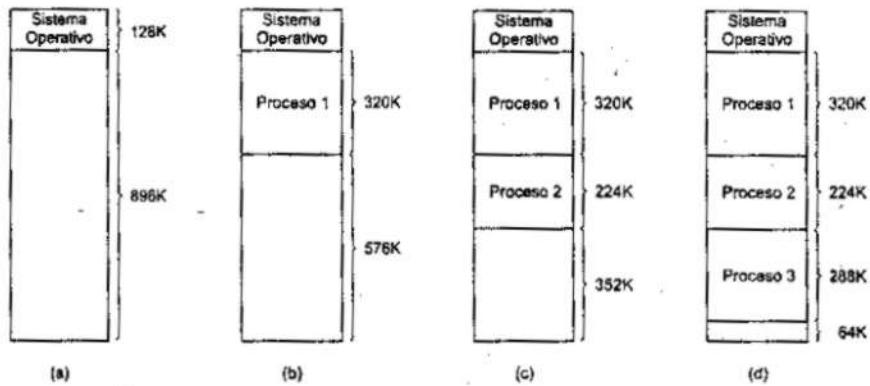


Figura 7.15. Efecto de la partición dinámica.

dejando espacio suficiente para cargar un nuevo proceso, el proceso 4 (f). Dado que el proceso 4 es más pequeño que el proceso 2, se crea otro hueco pequeño. Posteriormente, se produce la situación en la que ninguno de los procesos que están en la memoria está preparado, excepto el proceso 2, que está disponible puesto que se encuentra en el estado preparado-suspendido. Como hay un espacio de memoria insuficiente para el proceso 2, el sistema operativo retira de memoria al proceso 1 (g), y vuelve a introducir al proceso 2 (h). Como muestra este ejemplo, el método empieza bien, pero, eventualmente, puede llevar a situaciones en las que hay muchos huecos pequeños en memoria. A medida que pasa el tiempo, la memoria se fragmenta más y más, y empeora su utilización. Una técnica para solucionar este problema es la *compactación*: de vez en cuando, el sistema operativo desplaza los procesos en memoria para juntar toda la memoria libre en un bloque. Este es un procedimiento que consume parte del tiempo del procesador.

Antes de considerar formas de solucionar los problemas de la definición de particiones, debemos aclarar cierto extremo. Si el lector presta atención a la Figura 7.15 por un momento, resulta obvio que un proceso difícilmente se cargará en el mismo lugar de la memoria principal cada vez que se intercambia. Es más, si se utiliza compactación, un proceso puede desplazarse mientras se encuentra en memoria principal. La memoria del proceso está constituida por instrucciones y datos. Las instrucciones contendrán direcciones de posiciones de memoria de dos tipos:

- Direcciones de datos.
- Direcciones de instrucciones, usadas por las instrucciones de salto.

Pero estas direcciones no son fijas. Cambiarán cada vez que el proceso se intercambie. Para resolver este problema, se distingue entre direcciones lógicas y direcciones físicas. Una dirección lógica indica una posición relativa al comienzo del programa. Las instrucciones del programa contienen sólo direcciones lógicas. Una dirección física es, por supuesto, la posición actual en la memoria principal. Cuando el procesador ejecuta un proceso, automáticamente convierte las direcciones lógicas en físicas, sumando a cada dirección lógica la posición de comienzo actual del proceso, llamada dirección base. Este es otro ejemplo de un elemento hardware de la CPU diseñado para satisfacer las necesidades del sistema operativo. Las características exactas de este hardware dependen de la estrategia de gestión de memoria utilizada. Más adelante, en este mismo capítulo, veremos varios ejemplos.

## PAGINACIÓN

Tanto las particiones de tamaño fijo como las de tamaño variable son ineficaces en el aprovechamiento de la memoria. Supóngase, no obstante, que la memoria se divide en trozos iguales de tamaño fijo y relativamente pequeño, y que cada proceso también se divide en pequeños trozos de tamaño fijo. Después, los trozos de un programa, conocidos como páginas, se podrían asignar a los trozos de memoria disponibles, conocidos como marcos (frames), o marcos de página. Entonces, el espacio de memoria desperdiciado por un proceso es, como mucho, una fracción de la última página.

La Figura 7.16 muestra un ejemplo del uso de las páginas y los marcos. En un momento dado, algunos de los marcos de memoria están ocupados y otros están libres. La lista de marcos libres es gestionada por el sistema operativo. El proceso A, almacenado en disco, consta de cuatro páginas. Cuando llega el momento de cargar este proceso, el sistema operativo encuentra cuatro marcos libres, y carga las cuatro páginas del proceso A en cuatro marcos.

Supóngase ahora, como en este ejemplo, que no hay suficientes marcos contiguos sin utilizar para el proceso. ¿Hará esto que el sistema operativo no cargue A? La respuesta es no,

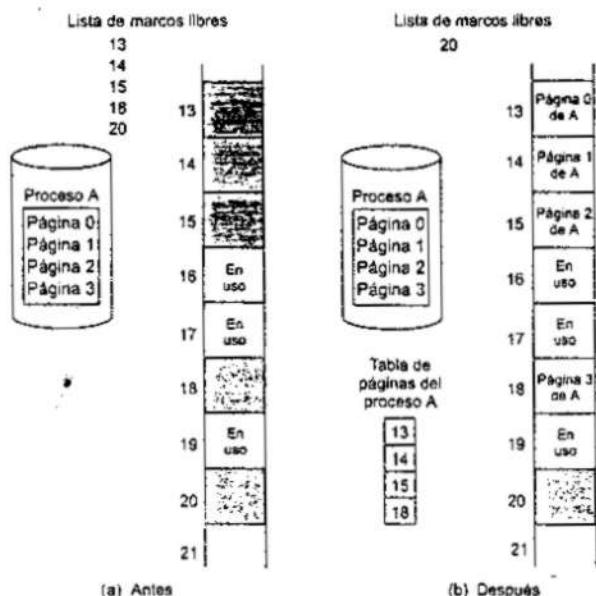


Figura 7.16. Asignación de marcos libres.

porque de nuevo se utiliza, una vez más, el concepto de dirección lógica. Ya no es suficiente una única dirección de base. En cambio, el sistema operativo mantiene una *tabla de páginas* para cada proceso. La tabla de páginas indica el marco que aloja a cada página del proceso. Dentro del programa, cada dirección lógica está constituida por un número de página y una dirección relativa dentro de la página. Recuérdese que, en el caso de particiones simples, una dirección lógica era la posición de una palabra en relación con el comienzo del programa; el procesador la traduce a una dirección física. Con la paginación, la traducción de dirección lógica a dirección física también la realiza el hardware del procesador. El procesador debe saber cómo acceder a la tabla de páginas del proceso en curso. A partir de la dirección lógica (número de página, dirección relativa), el procesador utiliza la tabla de páginas para generar la dirección física (número de marco, dirección relativa). Un ejemplo se muestra en la Figura 7.17.

Esta aproximación resuelve el problema anteriormente indicado. La memoria principal se divide en muchos marcos pequeños de igual tamaño. Cada proceso se divide en páginas del tamaño de los marcos; los procesos más pequeños necesitan menos páginas y los procesos mayores necesitan más. Cuando un proceso se lleva a memoria, sus páginas se cargan en los marcos disponibles, y la tabla de páginas se actualiza.

## MEMORIA VIRTUAL

### Paginación por demanda

Con el uso de la paginación, se dispone de sistemas con multiprogramación verdaderamente efectivos. Es más, la sencilla táctica de dividir el proceso en páginas llevó al desarrollo de otro concepto decisivo: la *memoria virtual*.

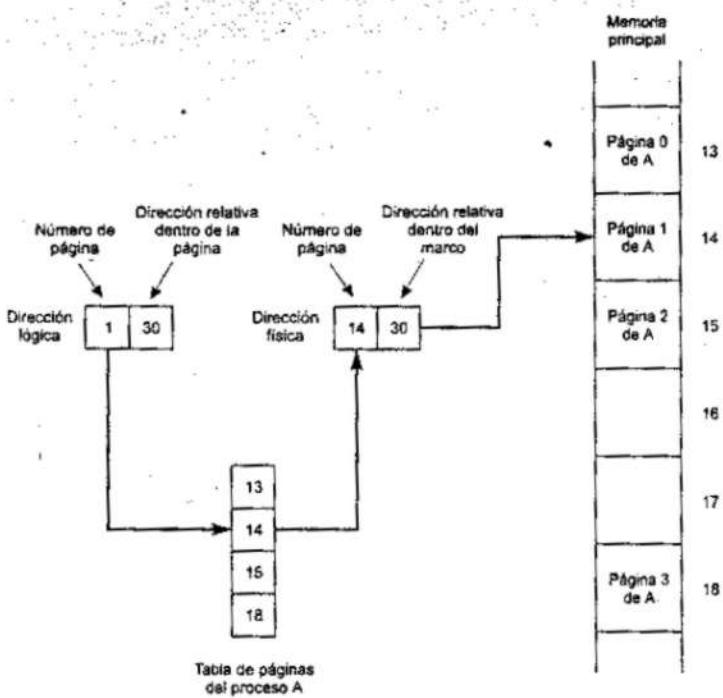


Figura 7.17. Direcciones lógicas y físicas.

Para entender la memoria virtual, debemos añadir una mejora al esquema de paginación discutido. Esta mejora es la *paginación por demanda*, que simplemente significa que cada página de un proceso se introduce en memoria sólo cuando se necesita (es decir, cuando se solicita o demanda).

Considérese un proceso de tamaño elevado, consistente en un programa largo más un cierto número de matrices de datos. En un intervalo de tiempo corto, la ejecución puede confinarse a una pequeña sección del programa (por ejemplo una subrutina), y quizás sólo se estén usando una o dos matrices de datos. Este es el principio de localidad, que se introdujo en el Apéndice 4A. Sería claramente un derroche cargar todas las páginas del proceso cuando sólo se utilizarán unas pocas antes de que el proceso se suspenda. Podemos hacer un mejor uso de la memoria cargando sólo unas pocas páginas. Después, si el programa salta a una instrucción de una página que no está en memoria, o si el programa hace referencia a un dato de una página que no está en memoria, se produce una *fallo de página*. Esto indica al sistema operativo que debe cargar la página deseada.

Así, en un momento dado, sólo unas pocas páginas de un proceso están en memoria y, en consecuencia, se pueden mantener en memoria más procesos. Además se ahorra tiempo, puesto que las páginas que no se utilizan no tienen que sufrir intercambios de almacenamiento. No obstante, el sistema operativo debe ser lo suficientemente ingenioso para manejar este esquema. Cuando se introduce una página en memoria, debe sacar otra fuera. Si saca una

página justo en el momento en que va a empezar a utilizarse, tendrá que volver a introducirla en memoria casi inmediatamente. Si esto ocurre frecuentemente se produce una situación conocida como *hiperpaginación* (*thrashing*): el procesador pasa la mayor parte de su tiempo intercambiando páginas, en lugar de ejecutar instrucciones. Las formas de evitar la hiperpaginación constituyeron una importante área de investigación en los años setenta, que dio lugar a una variedad de algoritmos complejos pero efectivos. En esencia, el sistema operativo intenta predecir, basándose en su historia reciente, qué páginas se utilizarán con menos probabilidad en el futuro próximo.

Con la paginación por demanda, no es necesario cargar el proceso entero en la memoria principal. Este hecho tiene una consecuencia importante: *es posible que un proceso sea mayor que toda la memoria principal*. Una de las restricciones más importantes de la programación ha sido vencida. Sin paginación por demanda, un programador debe tener en cuenta la memoria disponible. Si el programa que se está escribiendo es demasiado largo, el programador debe buscar formas de estructurar el programa en trozos que puedan cargarse uno a uno. Con demanda de página, ese trabajo se deja al sistema operativo y al hardware. En lo que al programador concierne, él, o ella, dispone de una cantidad de memoria enorme, el tamaño asociado al espacio en disco.

Puesto que un proceso se ejecuta sólo si está en memoria principal, ésta recibe el nombre de *memoria real*. Pero el programador o usuario percibe una memoria mucho mayor (la que hay disponible en disco). En consecuencia, ésta última se denomina *memoria virtual*. La memoria virtual posibilita una multiprogramación muy efectiva, y libera al usuario de las innecesarias y exigentes restricciones de memoria principal.

### Estructura de la tabla de páginas

El mecanismo básico para leer una palabra de memoria implica la traducción, mediante una tabla de páginas, de una dirección virtual o lógica (consistente en un número de página y un desplazamiento) a una dirección física (constituida por un número de marco y un desplazamiento). Puesto que la tabla de páginas tiene una longitud variable dependiendo del tamaño del proceso, no es posible almacenarla en los registros. En su lugar, debe accederse a ella en memoria principal. La Figura 7.17 sugiere una implementación hardware de este esquema. Cuando un proceso determinado está ejecutándose, un registro contiene la dirección de inicio de la tabla de páginas de ese proceso. El número de página de una dirección virtual se utiliza como índice en la tabla para buscar el correspondiente número de marco. Éste se combina con la parte de desplazamiento de la dirección virtual para construir la dirección real deseada.

En la mayoría de los sistemas, hay una tabla de páginas por proceso. Pero cada proceso puede ocupar una gran cantidad de memoria virtual. Por ejemplo, en la arquitectura VAX, cada proceso puede tener hasta  $2^{31} = 2$  GBytes de memoria virtual. Utilizando páginas de  $2^9 = 512$  bytes, eso significa que se necesitan tablas de páginas de  $2^{22}$  elementos *por proceso*. Claramente, la cantidad de memoria dedicada sólo a tablas de páginas podría ser inaceptablemente alta. Para solucionar este problema, la mayoría de los esquemas de memoria virtual almacenan las tablas de páginas en la memoria virtual, en lugar de en la memoria real. Esto significa que la tabla de páginas también está sujeta a paginación, igual que el resto de páginas. Cuando un proceso se está ejecutando, al menos una parte de su tabla de páginas, incluyendo el elemento correspondiente a la página actualmente en ejecución, debe estar en la memoria principal. Algunos procesadores hacen uso de un esquema de dos niveles para organizar las tablas de páginas grandes. En este esquema, hay una página de directorio en la que cada elemento apunta a una tabla de páginas. Así, si la longitud de la página de directorio es

$X$ , y si la longitud máxima de una tabla de páginas es  $Y$ , un proceso puede estar constituido por hasta  $X \times Y$  páginas. Típicamente, la longitud máxima de una tabla de páginas se restringe al tamaño de una página. Veremos un ejemplo de esta aproximación de dos niveles más adelante en este mismo capítulo, cuando estudiemos el Pentium II.

Una aproximación alternativa al uso de tablas de páginas de una o dos niveles es el uso de una estructura de *tabla de páginas invertida* (Fig. 7.18). Esta aproximación se utiliza en el AS/400 de IBM y en todos sus productos RISC, incluyendo el PowerPC.

En esta aproximación, la porción de la dirección virtual correspondiente al número de página se mapea en una tabla de dispersión (hash) mediante una función de dispersión sencilla<sup>1</sup>. La tabla de dispersión incluye un puntero a una tabla de páginas invertida, que contiene los elementos de la tabla de páginas. Con esta estructura, hay un elemento en la tabla de dispersión y en la tabla de páginas invertida para cada página de memoria real, en vez de para cada página de memoria virtual. Así, se necesita una porción fija de la memoria real para las tablas, independientemente del número de procesos o páginas virtuales que se admitan. Puesto que más de una dirección virtual puede apuntar al mismo elemento de la página de dispersión, se utiliza una técnica de encadenamiento para solucionar este problema. La técnica de dispersión da lugar a cadenas usualmente cortas, con uno o dos elementos.

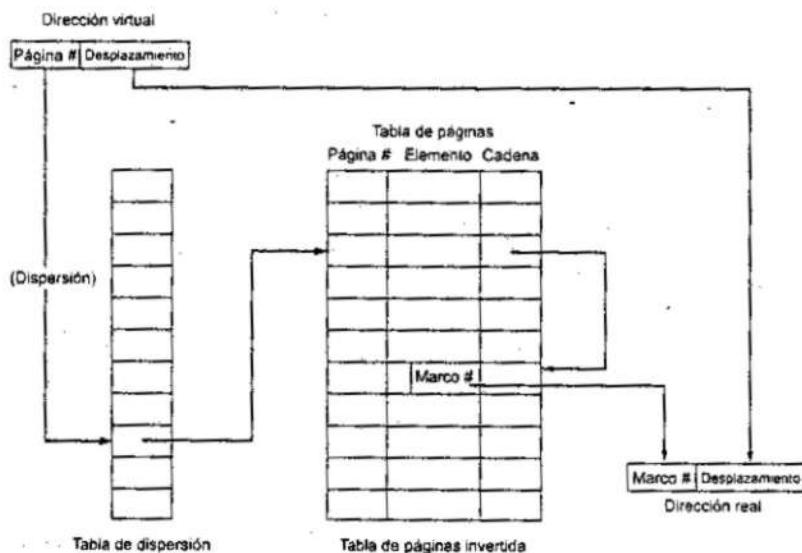


Figura 7.18. Estructura de la tabla de páginas invertida.

<sup>1</sup> Una función de dispersión asocia números comprendidos entre 0 y  $M$  con números entre 0 y  $N$ , donde  $M > N$ . La salida de la función de dispersión se utiliza como índice en la tabla de dispersión. Puesto que más de una entrada se asocia a la misma salida, es posible que un elemento de entrada apunte a una posición de la tabla de dispersión que está ocupada. En ese caso, el nuevo elemento debe pasar a otra posición de la tabla de mezcla. Usualmente, el nuevo elemento se sitúa en la primera posición vacía que se encuentra, y se establece un puntero desde la posición original para encadenar juntas las posiciones que se van ocupando. Véase [STAL98] para una discusión más detallada de las tablas de dispersión.

## BUFFER DE TRADUCCIÓN ANTICIPADA (TRANSLATION LOOKASIDE BUFFER)

En principio, toda referencia a memoria virtual puede ocasionar dos accesos a la memoria física: uno para captar el elemento de la tabla de páginas apropiada, y otro para captar el dato deseado. Como consecuencia, un esquema de memoria virtual directo tendría el efecto de duplicar el tiempo de acceso a memoria. Para resolver este problema, la mayoría de los esquemas de memoria virtual hacen uso de una cache especial para los elementos de la tabla de páginas, llamada usualmente «buffer de traducción anticipada» (TLB, Translation Lookaside Buffer). Este buffer funciona de la misma manera que una memoria cache, y contiene aquellos elementos de la tabla de páginas a los que se ha accedido más recientemente. La Figura 7.19 es un diagrama de flujo que muestra el uso del TLB. Por el principio de localidad, la mayoría de las referencias a memoria corresponderán a posiciones de las páginas recientemente usadas. Por eso, la mayoría de las referencias implican a elementos de la tabla de páginas incluidas en el TLB. Estudios del TLB de VAX muestran que este esquema puede mejorar significativamente las prestaciones [CLAR85, SATY81].

Observe que el mecanismo de memoria virtual debe interactuar con el sistema de cache (no con la cache que implementa el TLB, sino con la cache de la memoria principal). Esto se ilustra en la Figura 7.20. Una dirección virtual estará generalmente en la forma de número de página más desplazamiento. Primero, el sistema de memoria consulta el TLB para comprobar si hay coincidencia con algún elemento de la tabla de páginas incluido en él. Si es así, se genera la dirección real (física), combinando el número de marco con el desplazamiento. Si no, se accede al elemento correspondiente de la tabla de páginas. Una vez que se ha generado la dirección real, constituida por una marca y los bits restantes (véase la Figura 4.17), se consulta la cache para ver si el bloque que contiene la palabra está presente. Si es así, se envía al procesador. Si no, se busca la palabra en memoria principal.

El lector puede apreciar la complejidad del hardware del procesador implicado en una simple referencia a memoria. La dirección virtual es traducida a una dirección real. Esto implica una referencia a la tabla de páginas, que puede estar en el TLB, en memoria principal o en disco. La palabra referenciada puede estar en cache, en memoria principal, o en disco. En este último caso, la página que contiene a la palabra debe cargarse en la memoria principal, y su bloque debe pasar a la cache. Además, el elemento de la tabla de páginas correspondiente a esa página debe actualizarse.

## SEGMENTACIÓN

Hay otra forma en la que puede subdividirse la memoria direccionable, conocida como *segmentación*. Mientras que la paginación es invisible para el programador, y sirve para proporcionar al programador un espacio de direcciones mayor, la segmentación es usualmente visible para el programador, y proporciona una forma conveniente de organizar los programas y los datos, para asociar los privilegios y los atributos de protección con las instrucciones y los datos.

La segmentación permite que el programador vea la memoria constituida por múltiples espacios de direcciones o segmentos. Los segmentos tienen un tamaño variable, dinámico. Usualmente, el programador o el sistema operativo asignará programas y datos a segmentos distintos. Puede haber segmentos de programa distintos para varios tipos de programas, y también distintos segmentos de datos. Se pueden asignar a cada segmento derechos de acceso y uso. Las referencias a memoria se realizan mediante direcciones constituidas por un número de segmento y un desplazamiento.

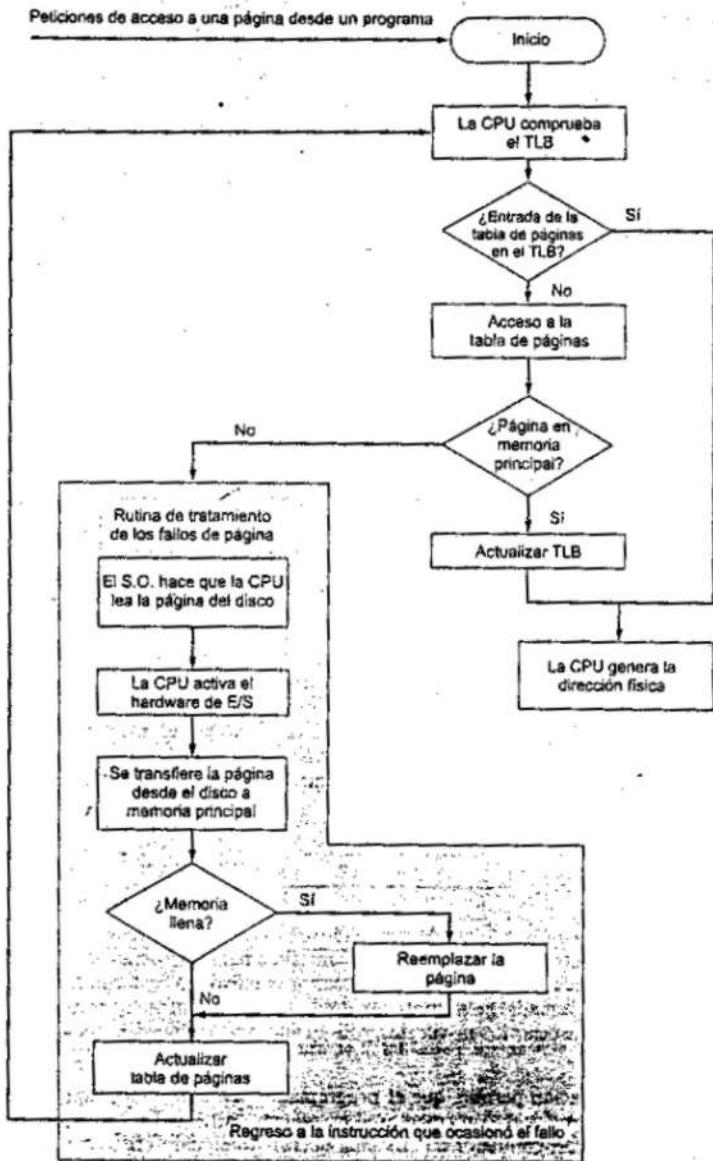


Figura 7.19. Funcionamiento de la paginación y del buffer de traducción anticipada (TLB) (FURH87).

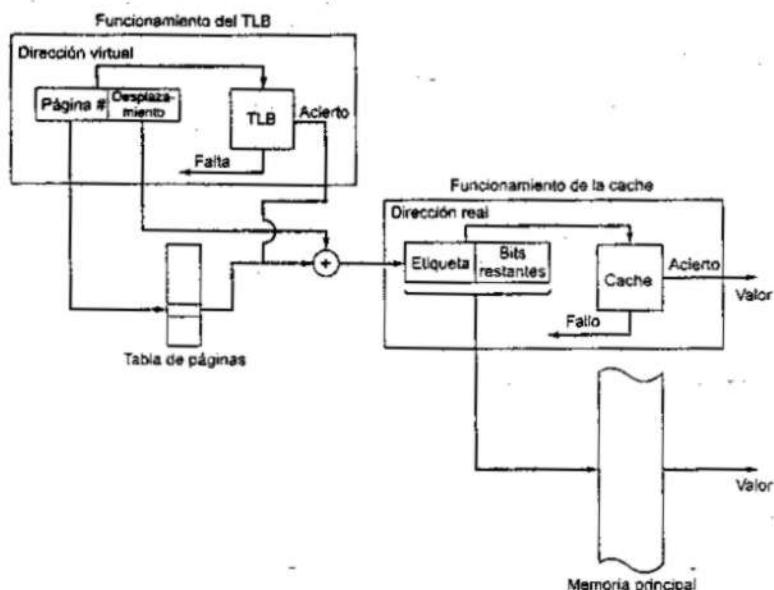


Figura 7.20. Buffer de traducción anticipada y operación de la cache.

Esta organización tiene ciertas ventajas para el programador, frente a un espacio de direcciones no segmentado:

1. Simplifica la gestión de estructuras crecientes de datos. Si el programador no conoce a priori el tamaño que puede llegar a tener una estructura de datos particular, no es necesario que lo presuponga. A la estructura de datos se le asigna su propio segmento, y el sistema operativo lo expandirá o lo reducirá según sea necesario.
2. Permite modificar los programas y recompilarlos independientemente, sin que sea necesario volver a enlazar y cargar el conjunto entero de programas. De nuevo, esto se consigue utilizando varios segmentos.
3. Permite que varios procesos compartan segmentos. Un programador puede situar un programa correspondiente a una utilidad o una tabla de datos de interés en un segmento, que puede ser direccionado por otros procesos.
4. Se facilita la protección. Puesto que un segmento se construye para contener un conjunto de programas o datos bien definido, el programador o el administrador del sistema puede asignar privilegios de acceso de forma adecuada.

Estas ventajas no se tienen con la paginación, que es invisible para el programador. Por otra parte, hemos visto que la paginación proporciona una forma eficiente de gestionar la memoria. Para combinar las ventajas de ambas, algunos sistemas están equipados con el hardware y el software del sistema operativo que permite las dos.

**GESTIÓN DE MEMORIA EN EL PENTIUM II****HARDWARE DE GESTIÓN DE MEMORIA EN EL PENTIUM II**

Desde la introducción de las arquitecturas de 32 bits, los microprocesadores han desarrollado esquemas de gestión de memoria sofisticados, aprovechando la experiencia obtenida con los sistemas de medio y gran tamaño. En muchos casos, las versiones de los microprocesadores son superiores a sistemas de media y gran escala que les antecedieron. Puesto que los sistemas fueron desarrollados por los fabricantes del hardware de microprocesadores, y debieran poder utilizarse con una cierta variedad de sistemas operativos, tienden bastante a ser de uso general. Un ejemplo representativo es el esquema utilizado por el Pentium II. El hardware de gestión de memoria del Pentium II es esencialmente el mismo que se usa en los procesadores 80386 y 80486 de Intel, con ciertas mejoras.

**Espacios de direcciones**

El Pentium II incluye hardware, tanto para segmentación como para paginación. Ambos mecanismos se pueden desactivar, permitiendo elegir entre cuatro formas de ver la memoria:

- **Memoria no segmentada y no paginada:** En este caso, la dirección virtual es la misma que la dirección física. Esto es útil, por ejemplo, cuando se utiliza como controlador de baja complejidad y elevadas prestaciones.
- **Memoria paginada no segmentada:** La memoria se ve como un espacio lineal de direcciones paginado. La protección y la gestión de memoria se realiza a través de la paginación. Esta es la forma preferida por ciertos sistemas operativos (por ejemplo, el UNIX de Berkeley).
- **Memoria segmentada no paginada:** Se ve la memoria como un conjunto de espacios de direcciones lógicas. La ventaja de esta imagen sobre el enfoque de la paginación estriba en que proporciona protección por debajo del nivel de byte, si es necesario. Es más, a diferencia de la paginación, garantiza que la tabla de traducción necesaria (la tabla de segmentos) se encuentra almacenada en el chip cuando el segmento está en memoria. De esta forma, la segmentación sin páginas da lugar a tiempos de acceso predecibles.
- **Memoria segmentada paginada:** Se utiliza la segmentación para definir particiones lógicas de memoria en el control de acceso, mientras que la paginación se usa para gestionar la asignación de memoria dentro de las particiones. Ciertos sistemas operativos, tales como el UNIX System V, prefieren esta visión de la memoria.

**Segmentación**

Cuando se utiliza segmentación, cada dirección virtual (llamada «dirección lógica» en la documentación del Pentium) consta de una referencia al segmento de 16 bits y un desplazamiento de 32 bits. Dos bits de la referencia al segmento se utilizan para el mecanismo de protección, y los 14 bits restantes para especificar al segmento en cuestión. Así, con una memoria no segmentada, la memoria virtual de usuario es  $2^{32} = 4$  GBytes. Con una memoria segmentada, el espacio de memoria virtual total visto por el usuario es  $2^{46} = 64$  terabytes (TBytes). El espacio de direcciones físicas emplea direcciones de 32 bits, con un máximo de 4 GBytes.

El volumen total de memoria virtual puede ser mayor de 64 TBytes. Esto se debe a que la forma de interpretar una dirección virtual por parte del procesador depende de la forma en

que esté activo en un momento dado. Una mitad del espacio de direcciones virtuales (8K segmentos × 4 GBytes) es global, compartida por todos los procesos; el resto de la memoria es local, y distinta para cada proceso.

Hay dos formas de protección asociadas a cada segmento: nivel de privilegio y atributo de acceso. Hay cuatro niveles de privilegio, desde el más protegido (nivel 0), al menos protegido (nivel 3). El nivel de privilegio asociado a un segmento de datos es su «clasificación»; el nivel de privilegio asociado con un segmento de programa es su «acreditación» («clearance»). Un programa en ejecución puede acceder a un segmento de datos sólo si su nivel de acreditación es menor (mayor privilegio) o igual (igual privilegio) que el nivel de privilegio del segmento de datos.

El hardware no indica cómo deben utilizarse estos niveles de privilegio; esto depende del diseño y de la implementación del sistema operativo. El nivel de privilegio 1 sería utilizado por la mayor parte del sistema operativo, y el nivel 0 por una pequeña parte del mismo, dedicada a la gestión de memoria, la protección y el control del acceso. Esto deja dos niveles para las aplicaciones. En muchos sistemas, las aplicaciones se encuentran en el nivel 3, dejándose sin utilizar el nivel 2. Los subsistemas de aplicación específica que deben protegerse debido a que implementan sus propios mecanismos de seguridad son buenos candidatos para situarse en el nivel 2. Algunos ejemplos son los sistemas de gestión de bases de datos, sistemas de automatización de oficinas y entornos de ingeniería del software.

Además de regular el acceso a los segmentos de datos, el mecanismo de privilegio limita el uso de ciertas instrucciones. Algunas instrucciones, tales como las que utilizan los registros de gestión de memoria, sólo pueden ejecutarse en el nivel 0. Las instrucciones de E/S sólo pueden ejecutarse en cierto nivel, determinado por el sistema operativo; éste suele ser el nivel 1.

El atributo de acceso al segmento de datos especifica si se permiten accesos de lectura-escritura o sólo de lectura. Para los segmentos de programa, el atributo de acceso especifica si se trata de acceso de lectura/ejecución o de sólo lectura.

El mecanismo de traducción de dirección para la segmentación implica hacer corresponder una dirección virtual con lo que se denomina una dirección lineal (Figura 7.21b). Una dirección virtual consiste en un desplazamiento de 32 bits y un selector de segmento de 16 bits (Figura 7.21a). El selector de segmentos consta de los siguientes campos:

- **Indicador de tabla (TI, Table Indicator):** Indica si para la traducción se va a utilizar la tabla de segmento global o la tabla de segmento local.
- **Número de segmento:** El número del segmento. Sirve como un índice en la tabla de segmentos.
- **Nivel de privilegio solicitado (RPL, Requested Privilege Level):** El nivel de privilegio para el acceso en cuestión.

Cada elemento en la tabla de segmentos consta de 64 bits, como muestra la Figura 7.21c. Los campos se definen en la Tabla 7.5.

## Paginación

La segmentación es una propiedad opcional, y puede desactivarse. Cuando se utiliza la segmentación, las direcciones utilizadas en los programas son direcciones virtuales, y se convierten en direcciones lineales, como se ha descrito. Cuando no se utiliza segmentación, los programas utilizan direcciones lineales. En cualquiera de los casos, el siguiente paso es la traducción de una dirección lineal a una dirección real de 32 bits.

|        |   |   |     |   |
|--------|---|---|-----|---|
| 15     | 3 | 2 | 1   | 0 |
| Índice | T | I | RPL |   |

TI = Indicador de tabla

RPL = Nivel de privilegio solicitado

(a) Selector de segmento

|            |       |    |                |    |   |
|------------|-------|----|----------------|----|---|
| 31         | 22    | 21 | 12             | 11 | 0 |
| Directorio | Tabla |    | Desplazamiento |    |   |

(b) Dirección lineal

|              |        |       |                     |    |       |                      |      |             |    |     |              |
|--------------|--------|-------|---------------------|----|-------|----------------------|------|-------------|----|-----|--------------|
| 31           | 24     | 23/22 | 20                  | 19 | 18/15 | 14                   | 13   | 12          | 11 | 8/7 | 0            |
| Base 31...24 | D      | A     | Límite del segmento | P  | DPL   | S                    | Tipo |             |    |     | Base 23...16 |
| G<br>I<br>B  | V<br>L |       | 19...15             |    |       |                      |      | Base 15...0 |    |     |              |
|              |        |       |                     |    |       | Segment Limit 15...0 |      |             |    |     |              |

AVL = Disponible para su uso por el programador del sistema

G = Granularidad  = Reservado

Base = Dirección base del segmento

Límite = Límite del segmento

D/B = Tamaño de operación por defecto

P = Presencia de segmento

DPL = Privilegio del descriptor

Tipo = Tipo de segmento

S = Tipo de descriptor

(c) Descriptor de segmento (entrada de la tabla de segmentos)

|                                      |     |    |    |   |   |     |        |   |             |                  |                       |
|--------------------------------------|-----|----|----|---|---|-----|--------|---|-------------|------------------|-----------------------|
| 31                                   | 12  | 11 | 9  | 7 | 6 | 5   | 4      | 3 | 2           | 1                | 0                     |
| Dirección de marco de página 31...12 |     |    |    |   |   | AVL | P<br>S | 0 | A<br>C<br>D | P<br>C<br>W<br>T | U<br>S<br>R<br>W<br>P |
| AVL                                  | PWT | US | RW | P |   |     |        |   |             |                  |                       |

AVL = Disponible para su uso por el programador del sistema

PWT = Escritura inmediata

PS = Tamaño de página

US = Usuario (supervisor)

A = Bit de acceso

RW = Lectura/escritura

PCD = Inhabilitación de cache

P = Presencia

(d) Elemento del directorio de páginas

|                                        |     |    |    |   |   |     |        |   |                  |                  |                       |
|----------------------------------------|-----|----|----|---|---|-----|--------|---|------------------|------------------|-----------------------|
| 31                                     | 12  | 11 | 9  | 7 | 6 | 5   | 4      | 3 | 2                | 1                | 0                     |
| Directorio de marco de páginas 31...12 |     |    |    |   |   | AVL | D<br>A | 0 | P<br>C<br>W<br>T | P<br>C<br>W<br>T | U<br>S<br>R<br>W<br>P |
| D                                      | PWT | US | RW | P |   |     |        |   |                  |                  |                       |

D = Bit de modificación

(e) Elemento de la tabla de páginas

Figura 7.21. Formatos para la gestión de memoria en el Pentium II.

Tabla 7.5. Parámetros para la gestión de memoria en el Pentium II

**Descriptor de segmentos (elemento en la tabla de segmentos)****Base**

Define la dirección de comienzo del segmento dentro del espacio lineal de direcciones de 4 GBytes.

**Bit D/B**

En un segmento de código, éste es el bit D, e indica si los operandos y modos de direccionamiento son de 16 o 32 bits.

**Nivel de privilegio del descriptor (DPL, Descriptor privilege level)**

Especifica el nivel de privilegio del segmento al que se refiere el descriptor de segmento en cuestión.

**Bit de granularidad (G)**

Indica si el campo límite debe ser interpretado en unidades de un byte o de 4 Kbytes.

**Límite**

Define el tamaño del segmento. El procesador interpreta el campo límite de dos formas posibles, según el bit de granularidad: en unidades de un byte, hasta un límite de 1 MByte en el tamaño del segmento, o en unidades de 4 KBytes, hasta un límite de 4 GBytes en el tamaño del segmento.

**Bit S**

Determina si un segmento dado es un segmento del sistema, o un segmento de código o de datos.

**Bit de segmento presente (P)**

Lo usan los sistemas no paginados. Indica si el segmento está disponible en memoria principal. En los sistemas paginados este bit está siempre a 1.

**Tipo**

Distingue entre varios tipos de segmentos e indica los atributos de acceso.

**Elementos del directorio de páginas y de la tabla de páginas****Bit de acceso (A, Accessed bit)**

El procesador pone este bit a 1 en ambos niveles de las tablas de página cuando se produce una operación de lectura o escritura en la página correspondiente.

**Bit de modificación (D, Dirty bit)**

El procesador pone a 1 este bit cuando se produce una operación de escritura en la página correspondiente.

**Dirección del marco de página**

Proporciona la dirección física de la página en la memoria si el bit de presencia está activo. Puesto que los marcos de página se alinean con los bloques de 4K, los 12 bits inferiores son 0, y sólo los 20 bits superiores se incluyen en el elemento. En el directorio de páginas, la dirección es la de la tabla de páginas.

**Bit de inhabilitación de cache para la página (PCD, Page Cache Disable bit)**

Indica si los datos de la página se pueden introducir en cache.

**Bit de tamaño de página (PS)**

Indica si la página es de 4 KBytes o de 4 MBytes.

**Bit de escritura inmediata de página (PWT, Page Write Through bit)**

Indica si se utiliza la política de escritura inmediata (Write Through) o la de Post-escritura (Write Back) para actualizar los datos en la página correspondiente.

**Bit de presencia (P, Present bit)**

Indica si la tabla de páginas o la página está en memoria principal.

**Bit de lectura/escritura (RW, Read/Write bit)**

En las páginas del nivel de usuario, indica si los programas de usuario pueden acceder a la página sólo para lectura, o para lectura y escritura.

**Bit de usuario/supervisor (US)**

Indica si la página es accesible sólo para el sistema operativo (nivel supervisor) o está disponible tanto para el sistema operativo como para las aplicaciones (nivel de usuario).

Para comprender la estructura de la dirección lineal, es preciso tener en cuenta que el mecanismo de paginación del Pentium II es de hecho una operación de búsqueda en una tabla de dos niveles. El primer nivel es un directorio de páginas, que contiene hasta 1.024 elementos. Esto divide los 4 GBytes del espacio lineal de memoria en 1.024 grupos de páginas, cada uno con su propia tabla de páginas, y cada uno de 4 MBytes de longitud. La ges-

tión de memoria permite la opción de utilizar un directorio de páginas para todos los procesos, un directorio de páginas para cada proceso o una combinación de los dos. El directorio de páginas para la tarea en curso está siempre en memoria principal. Las tablas de páginas pueden estar en memoria virtual.

La Figura 7.21 muestra los formatos de los elementos de los directorios de páginas y las tablas de páginas. Los campos se definen en la Tabla 7.5. Obsérvese que los mecanismos de control de acceso pueden proporcionarse en base a una página o a un grupo de páginas.

Además, el Pentium II hace uso del buffer de traducción rápida (TLB). El buffer puede contener 32 elementos de la tabla de páginas. Cada vez que cambia el directorio de páginas, el buffer se borra.

La Figura 7.22 ilustra la combinación de los mecanismos de segmentación y paginación. Por claridad, los mecanismos del TLB y de la memoria cache no se muestran.

Finalmente, el Pentium II incluye una nueva ampliación que no existe ni en el 80386 ni en el 80486: se permiten dos tamaños de páginas. Si el bit PSE (extensión de tamaño de página) del registro de control 4 está a 1, la unidad de paginación permite que el programador del sistema operativo defina la página con un tamaño de 4 KBytes o de 4 MBytes.

Cuando se utilizan páginas de 4 MBytes, hay un sólo nivel en la tabla de búsqueda de páginas. Cuando el hardware accede al directorio de páginas, el elemento del directorio de páginas (Figura 7.21d) tiene el bit PS a 1. En este caso, se ignoran desde el bit 9 al bit 21, y los bits desde el 22 al 31 definen la dirección base de una página de memoria de 4 MBytes. Así, hay una sola tabla de páginas.

El uso de páginas de 4 MBytes reduce las necesidades de almacenamiento para la gestión de memoria en memorias principales grandes. Con páginas de 4 KBytes, una memoria principal de 4 GBytes necesita del orden de 4 MBytes de memoria sólo para la tabla de páginas. Con páginas de 4 MBytes, una única tabla, de 4 KBytes de longitud, es suficiente para la gestión de las páginas.

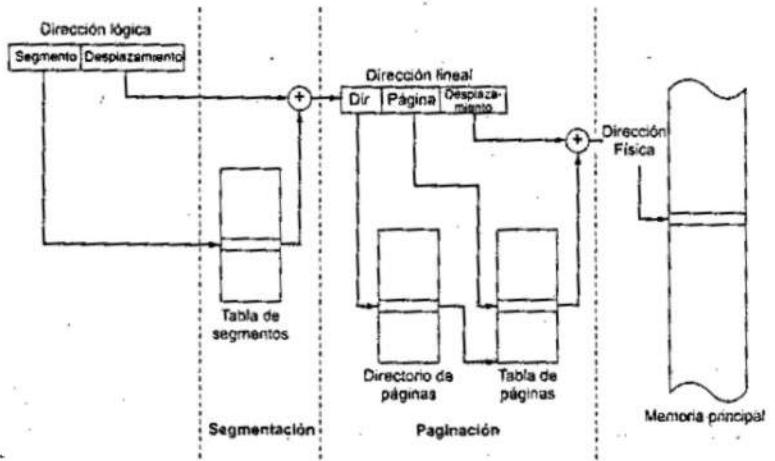


Figura 7.22. Mecanismos de traducción de una dirección de memoria en el Pentium II.

## HARDWARE DE GESTIÓN DE MEMORIA EN EL PowerPC

El PowerPC proporciona un amplio conjunto de mecanismos de direccionamiento. Para las implementaciones de la arquitectura de 32 bits, existe un esquema de paginación con un mecanismo sencillo de segmentación. Para las implementaciones de 64 bits, son posibles la paginación y un mecanismo más potente de segmentación. Además, tanto para las máquinas de 32 como de 64 bits hay un mecanismo hardware alternativo, conocido como «traducción de dirección de bloque». Brevemente, el esquema de direccionamiento de bloque está diseñado para resolver un problema de los mecanismos de paginación. Con la paginación, un programa puede hacer referencias frecuentes a un número elevado de páginas. Por ejemplo, los programas que utilizan tablas del sistema operativo, o buffers para tramas gráficas, pueden tener este comportamiento. Como resultado, las páginas frecuentemente usadas pueden estar constantemente introduciéndose y sacándose de memoria. El direccionamiento de bloque permite al procesador definir cuatro bloques grandes de la memoria de instrucciones y cuatro bloques grandes de la memoria de datos sobre los que no se aplica el mecanismo de paginación.

Una discusión sobre el direccionamiento de bloque está fuera del alcance de este capítulo. En esta subsección, nos concentraremos en los mecanismos de paginación y segmentación del PowerPC de 32 bits. El esquema para el de 64 bits es similar.

El PowerPC de 32 bits utiliza direcciones efectivas de 32 bits (Figura 7.23a). La dirección incluye un identificador de página de 16 bits y un selector de byte de 12 bits. Así pues, se utilizan páginas de  $2^{12} = 4$  KBytes. Son posibles hasta  $2^{16} = 64$ K páginas por segmento. Cuatro bits de la dirección se utilizan para designar uno de los 16 registros de segmento. Los contenidos de estos registros están controlados por el sistema operativo. Cada registro de

|          |        |         |    |
|----------|--------|---------|----|
| 0        | 3 / 4  | 19 / 20 | 31 |
| Segmento | Página | Byte    |    |

(a) Dirección efectiva

|                       |                               |              |     |
|-----------------------|-------------------------------|--------------|-----|
| 0 / 1                 |                               | 24 / 25 / 26 | 31  |
| V                     | ID de segmento virtual (VSID) | H            | API |
| Número de página real | R C WIMG PP                   |              |     |
| 0                     | 19 23 24 25 28 30 31          |              |     |

V = Bit de elemento válido  
 H = Identificador de función  
     de dispersión  
 API = Índice de página abreviada

R = Bit de referencia  
 C = Bit de cambio  
 WIMG = Bits de control de acceso a cache y memoria  
 PP = Bits de protección de página

(b) Elemento de tabla de páginas

|                       |                        |    |
|-----------------------|------------------------|----|
| 0                     | 19 / 20                | 31 |
| Número de página real | Desplazamiento de byte |    |

(c) Dirección real

Figura 7.23. Formatos para la gestión de memoria en el PowerPC de 32 bits.

segmento incluye bits de control de acceso y un identificador de 24 bits, de manera que una dirección efectiva de 32 bits se hace corresponder a una dirección virtual de 52 bits (Figura 7.24).

El PowerPC utiliza una sola tabla de páginas invertida. La dirección virtual se utiliza como un índice en la tabla de páginas, que actúa de la siguiente manera: En primer lugar, se calcula un código de mezcla como sigue:

$$H(0 \dots 18) = SID(5 \dots 23) \oplus VPN(0 \dots 18)$$

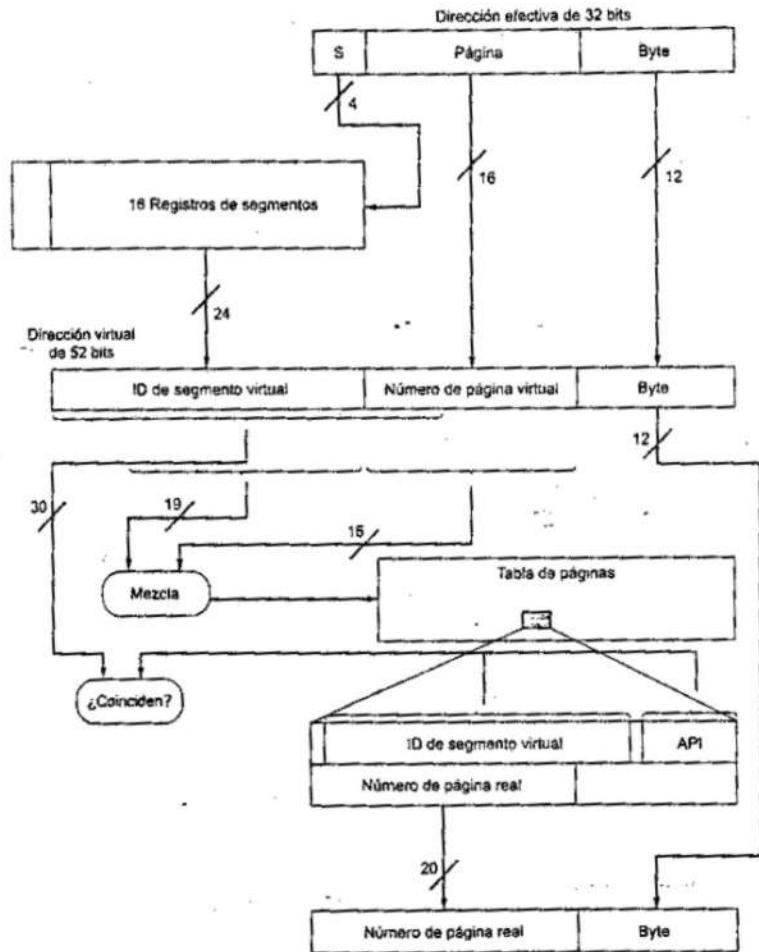


Figura 7.24. Traducción de direcciones en el PowerPC de 32 bits.

El número de página virtual (VPN) en la dirección de página virtual se completa añadiendo a su izquierda (extremo más significativo) tres ceros, para formar un número de 19 bits. Entonces se realiza la operación exclusive-or bit a bit de este número con los 19 bits más a la derecha del identificador de segmento virtual (SID) para formar un código de mezcla (H) de 19 bits. La tabla está organizada en n grupos de 8 elementos. De 10 a 19 bits del código de mezcla (según el tamaño de la tabla de páginas) se utilizan para seleccionar uno de los grupos de la tabla. El hardware de gestión de memoria comprueba luego los ocho elementos del grupo para determinar si hay alguna coincidencia con la dirección virtual.

Para determinar si hay coincidencia, cada elemento de la tabla de páginas incluye un identificador (ID) de segmento virtual y los 6 bits más a la izquierda del número de página virtual, llamado «índice de página abreviado». Puesto que al menos 10 bits del número de página virtual siempre participan en la función de mezcla para seleccionar un grupo de elementos de la tabla de páginas, sólo se necesita almacenar en el elemento de la tabla de páginas una forma abreviada del número de página virtual para comprobar la dirección virtual. Si hay coincidencia, entonces se obtiene de la dirección un número de 20 bits, correspondiente a la página real, que se concatena con los 12 bits menos significativos de la dirección efectiva para formar la dirección física de 32 bits a la que se accede.

Tabla 7.6. Parámetros para la gestión de memoria en el PowerPC

| Elementos de la tabla de segmentos                                   |                                                                                                                                                                                                                                                                                                                                                               |
|----------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Identificador (ID) de segmento efectivo</b>                       | Indica uno de los 64G segmentos efectivos; usado para determinar el elemento en la tabla de segmentos.                                                                                                                                                                                                                                                        |
| <b>Bit de elemento válido (V)</b>                                    | Indica si hay un dato válido en este elemento.                                                                                                                                                                                                                                                                                                                |
| <b>Bit de tipo de segmento (T)</b>                                   | Indica si es un segmento de memoria o de E/S.                                                                                                                                                                                                                                                                                                                 |
| <b>Clave de supervisor (Ks)</b>                                      | Usado con el número de página virtual para determinar el elemento de la tabla de páginas.                                                                                                                                                                                                                                                                     |
| Elementos de la tabla de páginas                                     |                                                                                                                                                                                                                                                                                                                                                               |
| <b>Bit de elemento válido (V)</b>                                    | Indica si hay un dato válido en este elemento.                                                                                                                                                                                                                                                                                                                |
| <b>Identificador de función de dispersión (H, inicial de «hash»)</b> | Indica si es un elemento de dispersión primario o secundario.                                                                                                                                                                                                                                                                                                 |
| <b>Índice de página abreviada (API, Abbreviated Page Index)</b>      | Utilizada para establecer la coincidencia con una dirección virtual única.                                                                                                                                                                                                                                                                                    |
| <b>Bit de referenciado (R)</b>                                       | El procesador pone a 1 este bit cuando se produce una operación de lectura o escritura a la página correspondiente.                                                                                                                                                                                                                                           |
| <b>Bit de cambiado (C)</b>                                           | El procesador pone a 1 este bit cuando se produce una operación de escritura a la página correspondiente.                                                                                                                                                                                                                                                     |
| <b>Bits WIMG</b>                                                     | <ul style="list-style-type: none"> <li>W = 0: usado con la estrategia de post-escritura.</li> <li>W = 1: usado con la estrategia de escritura directa.</li> <li>I = 0: cache no inhibida; I = 1: cache inhibida.</li> <li>M = 0: memoria no compartida; M = 1: memoria compartida.</li> <li>G = 0: memoria no protegida; G = 1: memoria protegida.</li> </ul> |
| <b>Bits de protección de página (PP)</b>                             | Bits de control de acceso utilizados con los bits K del registro de segmento o del elemento de la tabla de segmentos para definir los derechos de acceso.                                                                                                                                                                                                     |

Si no hay coincidencia, el código de mezcla se complementa para producir un nuevo índice de la tabla de páginas que está en la misma posición con respecto al extremo opuesto de la tabla. Este grupo se comprueba para determinar si hay coincidencia. Si no la hay, se produce una interrupción de falta de página.

La Figura 7.23 muestra la lógica del mecanismo de traducción de direcciones, y la Figura 7.24 muestra los formatos de las direcciones efectivas, de un elemento de la tabla de páginas y de las direcciones reales. Finalmente, la Tabla 7.6 define los parámetros de un elemento de la tabla de páginas.

El esquema de gestión de memoria de 64 bits está diseñado para ser compatible ascendente con la implementación de 32 bits. En esencia, todas las direcciones efectivas, los registros generales y los registros de direcciones de salto se amplían a la izquierda de los 32 bits para constituir los 64 bits.

## SITIOS WEB RECOMENDADOS

En [STAL98] se cubren con detalle los temas de este capítulo.

[STAL98] Stallings, W. *Operating Systems, Internals and Design Principles*. 3rd Edition. Upper Saddle River, NJ: Prentice Hall, 1998.



## SITIOS WEB RECOMENDADOS:

- Open Group Research Institute - Operating System Program: Este grupo (una fusión de la Open Software Foundation y de la compañía X/Open) realiza I + D en una amplia gama de áreas del ámbito de los sistemas operativos.
- ACM Special Interest Group on Operating Systems (SIGOPS): Información acerca de las publicaciones y conferencias de este grupo de interés en sistemas operativos de la ACM.

## PROBLEMAS

- 7.1. Suponga que tenemos un computador multiprogramado en el que cada trabajo tiene características idénticas. En un período de computación,  $T$ , para un trabajo, la mitad del tiempo corresponde a E/S y la otra mitad a actividad de la CPU. Cada trabajo se ejecuta durante un total de  $N$  períodos. Defina las siguientes cantidades:

- Tiempo de respuesta («Turnaround») = Tiempo para completar un trabajo.
- Rendimiento («Throughput») = Número medio de trabajos terminados por período de tiempo,  $T$ .
- Utilización de CPU = Porcentaje de tiempo que está activa la CPU (sin estar esperando).

Calcule estas cantidades para uno, dos y cuatro trabajos simultáneos, asumiendo que el período  $T$  se distribuye de cada una de las siguientes formas:

- a) E/S la primera mitad, CPU la segunda mitad.
- b) E/S el primer y el último cuarto de tiempo, CPU el segundo y el tercer cuarto.

- 7.2. Un programa «limitado por E/S» (I/O bound) es uno que, si se ejecuta en solitario, pasaría más tiempo esperando que se completen las E/S que utilizando el procesador. Un programa «limitado por el procesador» (processor-bound) es lo contrario.

Suponga un algoritmo de planificación a corto plazo que favorece a aquellos programas que han utilizado poco el procesador en el pasado reciente. Explique por qué este algoritmo favorece a los programas limitados por E/S y, sin embargo, no deja sin atender a los programas limitados por el procesador.

- 7.3. Un programa calcula las sumas de filas

$$C_i = \sum_{j=1}^n a_{ij}$$

de una matriz  $A$  de  $100 \times 100$ . Asuma que el computador utiliza paginación por demanda, con un tamaño de páginas de 1.000 palabras, y que la cantidad de memoria principal reservada para datos es de cinco marcos de página. ¿Habrá alguna diferencia en la fracción de faltas de página si  $A$  estuviera almacenada en memoria virtual por filas o por columnas? Explíquelo.

- 7.4. Suponga que la tabla de páginas para el proceso que se está ejecutando en un procesador es la que se muestra a continuación. Todos los números son decimales, todos se numeran desde cero, y todas las direcciones de memoria son direcciones de bytes. El tamaño de la página es de 1.024 bytes.

| Número de página virtual | Bit de validez | Bit de referencia | Bit de modificación | Número de marco de página |
|--------------------------|----------------|-------------------|---------------------|---------------------------|
| 0                        | 1              | 1                 | 0                   | 4                         |
| 1                        | 1              | 1                 | 1                   | 7                         |
| 2                        | 0              | 0                 | 0                   | —                         |
| 3                        | 1              | 0                 | 0                   | 2                         |
| 4                        | 0              | 0                 | 0                   | —                         |
| 5                        | 1              | 0                 | 1                   | 0                         |

- a) Describa exactamente cómo, en general, una dirección virtual generada por la CPU se traduce a una dirección física.
- b) ¿Qué dirección física, si existe, correspondería a cada una de las siguientes direcciones virtuales? (No gestione ningún fallo de página, si se produce.)
- (i) 1.052
  - (ii) 2.221
  - (iii) 5.499

- 7.5. Indique las razones por las que el tamaño de página en un sistema de memoria virtual no debe ser ni muy grande ni muy pequeño.
- 7.6. La siguiente secuencia de números de páginas virtuales se produce en el curso de la ejecución de un programa en un computador con memoria virtual:

3 4 2 6 4 7 1 3 2 6 3 5 1 2 3

Asuma que se ha adoptado una estrategia de reemplazo de la página menos recientemente usada. Dibuje una gráfica de la tasa de aciertos de página (fracción de referencias que encuentran la página en la memoria principal) en función de la capacidad de páginas de la memoria principal,  $n$ , para  $1 \leq n \leq 8$ . Considere que la memoria principal está inicialmente vacía.

- 7.7. En el computador VAX, las tablas de páginas del usuario se sitúan en direcciones virtuales del espacio de sistema. ¿Cuál es la ventaja de tener las tablas de páginas del usuario en la memoria virtual en lugar de en la memoria principal? ¿Cuál es la desventaja?
- 7.8. Considere un computador con segmentación y paginación. Cuando un segmento está en memoria, se desperdician algunas palabras de la última página. Además, para un segmento de tamaño  $s$  y un tamaño de página  $p$ , hay  $s/p$  elementos en la tabla de páginas. Cuanto más pequeño sea el tamaño de la página, menor será lo que se desperdicia en la última página del segmento, pero mayor será la tabla de páginas. ¿Qué tamaño de página minimiza la memoria suplementaria («overhead») total?
- 7.9. Un computador tiene una cache, memoria principal y un disco utilizado para la memoria virtual. Si una palabra está en la cache, se necesitan 20 ns para acceder a ella. Si está en memoria principal y no en la cache, se necesitan 60 ns para cargarla primero en la cache, y después empieza de nuevo la referencia a la palabra. Si la palabra no está en la memoria principal, se necesitan 12 ns para traerla del disco, más 60 ns para pasársela a la cache. La tasa de aciertos de cache es 0,9 y la tasa de aciertos de memoria principal es 0,6. ¿Cuál es el tiempo medio, en nanosegundos, que se necesita en este sistema para acceder a una palabra?
- 7.10. Suponga que una tarea está dividida en cuatro segmentos de igual tamaño, y que el sistema construye para cada segmento una tabla de descriptores de página con 8 elementos. Así pues, el sistema utiliza una combinación de segmentación y paginación. Asuma también que el tamaño de la página es de 2 KBytes.
  - a) ¿Cuál es el tamaño máximo de cada segmento?
  - b) ¿Cuál es el espacio de direcciones lógicas máximo para una tarea?
  - c) Si una tarea accede a un elemento en la posición física 00021'ABC. ¿Cuál es el formato de la dirección lógica que la tarea genera para él? ¿Cuál es el espacio máximo de direcciones físicas del sistema?

Fuente: [ALEX93].

- 7.11. Asuma que cierto microprocesador es capaz de acceder a  $2^{32}$  bytes de memoria física principal. El microprocesador implementa un espacio de direcciones lógicas segmentado de un tamaño máximo de  $2^{31}$  bytes. Cada instrucción contiene las dos partes de la dirección completa. Se utilizan unidades de gestión de memoria (MMU, Memory Management Units) externas, cuyo esquema de gestión de memoria asigna a los segmentos bloques contiguos de memoria física de un tamaño fijo de  $2^{22}$  bytes. La dirección física de comienzo de un segmento siempre es divisible por 1.024. Muestre la interconexión detallada del mecanismo de correspondencia externo que convierte las direcciones lógicas en direcciones físicas utilizando el número apropiado de MMU, y muestre la estructura interna detallada de una MMU (asuma que cada MMU contiene una cache de correspondencia directa de 128 elementos para los descriptores de segmento) y la forma de seleccionar cada MMU.

Fuente: [ALEX93].

- 7.12 Considere un espacio de direcciones lógicas paginado (compuesto por 32 páginas de 2 KBytes cada una) asignado a un espacio de memoria física de 1 MByte.
- ¿Cuál es el formato de las direcciones lógicas del procesador?
  - ¿Cuál es la longitud y la anchura de la tabla de páginas (sin considerar los bits correspondientes a los «derechos de acceso»)?
  - ¿Qué efecto se produce en la tabla de páginas si el espacio de memoria física se reduce a la mitad?

Fuente: [ALEX93].

11  
H  
W

## **P A R T E   I I I**

# **LA UNIDAD CENTRAL DE PROCESAMIENTO**

**H**emos visto hasta ahora la CPU esencialmente como una «caja negra», y hemos considerado su interacción con las E/S y la memoria. La Parte III se dedica a la estructura y funcionamiento de la CPU. La CPU consta de la unidad de control, registros, la unidad aritmético-lógica, la unidad de ejecución de instrucciones y las interconexiones entre estos componentes. Se cubren los temas de arquitectura, tales como el diseño del repertorio de instrucciones y los tipos de datos. En esta parte se tratan también aspectos relativos a organización, tales como la segmentación («pipelining»).

### **ESQUEMA DE LA TERCERA PARTE**

#### **CAPÍTULO 8. ARITMÉTICA DEL COMPUTADOR**

El Capítulo 8 examina el funcionamiento de la ALU, y se centra en la representación de los números y las técnicas para realizar operaciones aritméticas. Los procesadores normalmente admiten dos tipos de aritmética: de coma fija o de enteros, y de coma flotante. Para ambos casos, el capítulo primero analiza la representación de los números y, posteriormente, trata las operaciones aritméticas. Se estudia con detalle el importante estándar de coma flotante IEEE 754.

#### **CAPÍTULO 9. REPERTORIOS DE INSTRUCCIONES: CARACTERÍSTICAS Y FUNCIONES**

El complejo tema del diseño de repertorios de instrucciones ocupa los Capítulos 9 y 10. El Capítulo 9 se centra en los aspectos funcionales del diseño de un repertorio de instrucciones. Este capítulo examina los tipos de funciones que se especifican mediante instrucciones del computador, concentrándose entonces en los tipos de operandos (que especifican los datos con los que se opera) y en los tipos de operadores (que especifican las operaciones a realizar) que normalmente encontramos en los repertorios de instrucciones.

## CAPÍTULO 10. REPERTORIOS DE INSTRUCCIONES: MODOS DE DIRECCIONAMIENTO Y FORMATOS

Mientras el Capítulo 9 puede decirse que trata la semántica de los repertorios de instrucciones, el Capítulo 10 está más relacionado con la sintaxis de dichos repertorios. Concretamente, en este capítulo se ve la forma de especificar las direcciones de memoria y el formato general de las instrucciones del computador.

## CAPÍTULO 11. ESTRUCTURA Y FUNCIÓN DE LA CPU

El Capítulo 11 describe el uso de registros, como la memoria interna de la CPU, empleando todo el material visto hasta ese momento, para proporcionar una revisión de la estructura y funcionamiento de la CPU. Se revisa su organización general (ALU, unidad de control y banco de registros), y la organización del banco de registros. El resto del capítulo describe el funcionamiento del procesador cuando ejecuta instrucciones máquina. Se analiza el ciclo de instrucción para mostrar el funcionamiento y la interrelación de los ciclos de captación, indirecto, de ejecución y de interrupción. Finalmente, se explora con detalle la mejora de prestaciones que produce la segmentación.

## CAPÍTULO 12. COMPUTADORES DE REPERTORIO REDUCIDO DE INSTRUCCIONES

En el resto de la Parte III se ve con más detalle las tendencias clave en el diseño de la CPU. El Capítulo 12 describe la aproximación asociada con el concepto de computador de conjunto reducido de instrucciones (RISC). Este capítulo examina las motivaciones para el uso del diseño RISC, tratando a continuación los detalles de diseño del repertorio de instrucciones de un RISC y la arquitectura de su CPU.

## CAPÍTULO 13. PARALELISMO A NIVEL DE INSTRUCCIONES Y PROCESADORES SUPERESCALARES

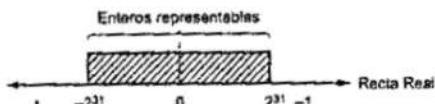
En el Capítulo 13 se ve el uso de técnicas superescalares, una aproximación utilizada en muchos de los diseños de procesadores más modernos.

## CAPÍTULO 8

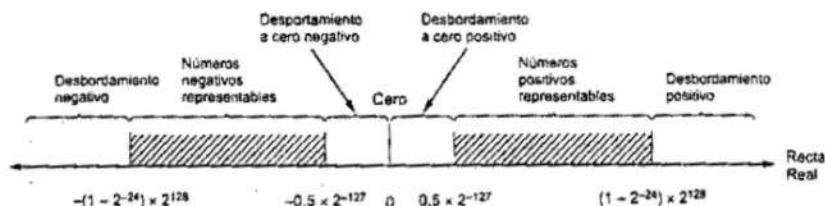
---

# Aritmética del computador

- 8.1. La unidad aritmético-lógica (ALU)
  - 8.2. Representación de enteros
    - Representación en signo y magnitud
    - Representación en complemento a dos
    - Conversión entre longitudes de bits diferentes
    - Representación en coma fija
  - 8.3. Aritmética con enteros
    - Negación
    - Suma y resta
    - Multiplicación
    - División
  - 8.4. Representación en coma flotante
    - Fundamentos
    - Estándar del IEEE para la representación binaria en coma flotante
  - 8.5. Aritmética en coma flotante
    - Suma y resta
    - Multiplicación y división
    - Consideraciones sobre precisión
    - Estándar del IEEE para la aritmética binaria en coma flotante
  - 8.6. Lecturas y sitios Web recomendados
  - 8.7. Problemas
- Apéndice 8A. Sistemas de numeración**
- Sistema decimal
  - Sistema binario
  - Conversión entre binario y decimal
  - Notación hexadecimal



(a) Enteros en complemento a dos



(b) Números en coma flotante

- Los dos aspectos fundamentales de la aritmética del computador son la forma de representar los números (el formato binario) y los algoritmos utilizados para realizar las operaciones aritméticas básicas (suma, resta, multiplicación y división). Estas dos consideraciones se aplican, tanto a la aritmética de enteros como a la de coma flotante.
- Las cantidades en coma flotante se expresan como un número (mantisa) multiplicado por una constante (base) elevada a una potencia entera (exponente). Los números en coma flotante pueden utilizarse para representar cantidades muy grandes y muy pequeñas.
- La mayoría de los procesadores implementan la normalización o estándar IEEE 754 para la representación de números y aritmética en coma flotante. Esta norma define el formato de 32 bits, así como el de 64 bits.

**C**omenzamos nuestro estudio de la CPU con la unidad aritmético-lógica (ALU). Tras una breve introducción a la ALU, el capítulo se centra en el aspecto más complejo de la misma: la aritmética del computador. Las funciones lógicas que forman parte de la ALU se describen en el Capítulo 9, y la implementación de funciones lógicas y aritméticas sencillas mediante lógica digital se describen en el Apéndice A de este libro.

La aritmética de un computador es realizada normalmente con dos tipos de números muy diferentes: enteros y en coma flotante. En ambos casos, la representación elegida es un aspecto de diseño crucial, que trataremos en primer lugar, seguido de una discusión sobre las operaciones aritméticas.

En un apéndice de este capítulo se proporciona un resumen sobre sistemas de numeración.

### 8.1 UNIDAD ARITMÉTICO-LÓGICA (ALU)

La ALU es la parte del computador que realiza realmente las operaciones aritméticas y lógicas con los datos. El resto de los elementos del computador (unidad de control, registros, memoria y E/S) están principalmente para suministrar datos a la ALU, a fin de que ésta los procese, y para recuperar los resultados. Con la ALU llegamos al estudio de lo que puede considerarse el núcleo o esencia del computador.

Una unidad aritmético-lógica y, en realidad, todos los componentes electrónicos del computador, se basan en el uso de dispositivos lógicos digitales sencillos que pueden almacenar dígitos binarios y realizar operaciones lógicas booleanas elementales. El apéndice de este texto explora, para el lector interesado, la implementación de lógica digital.

La Figura 8.1 indica, en términos muy generales, cómo se interconecta la ALU con el resto del procesador. Los datos se presentan a la ALU en registros, y en registros se almacenan los resultados de las operaciones producidos por la ALU. Estos registros son posiciones de memoria temporal internas al procesador que están conectados a la ALU (véase, por ejemplo, la Figura 2.3). La ALU puede también activar indicadores (flags) como resultado de una operación. Por ejemplo, un indicador de desbordamiento se pondrá a 1 si el resultado de una operación excede la longitud del registro en donde éste debe almacenarse. Los valores de los indicadores se almacenan también en otro registro dentro del procesador. La unidad de control proporciona las señales que gobiernan el funcionamiento de la ALU y la transferencia de datos dentro y fuera de la ALU.

### 8.2 REPRESENTACIÓN DE ENTEROS

En el sistema de numeración binaria (véase Apéndice 8A), cualquier número puede representarse tan solo con los dígitos 1 y 0, el signo menos y la coma (el punto en los países anglosajones). Por ejemplo:

$$-1.101.0101_2 = -13.3125_{10}$$

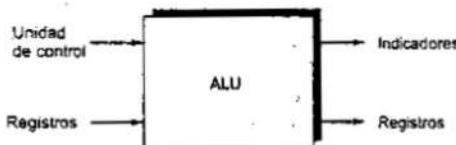


Figura 8.1. Entradas y salidas de la ALU.

Sin embargo, para ser almacenados y procesados por un computador, no se tiene la posibilidad de disponer del signo y de la coma. Para representar los números, sólo pueden utilizarse los dígitos 0 y 1. Si utilizáramos sólo enteros no negativos, su representación sería inmediata. Una palabra de 8 bits podría utilizarse para representar números desde 0 hasta 255. Por ejemplo:

$$00000000 = 0$$

$$00000001 = 1$$

$$00101001 = 41$$

$$10000000 = 128$$

$$11111111 = 255$$

En general, si una secuencia de  $n$  dígitos binarios  $a_{n-1}a_{n-2}\dots a_1a_0$  es interpretada como un entero sin signo  $A$ , su valor es:

$$A = \sum_{i=0}^{n-1} 2^i a_i$$

### REPRESENTACIÓN EN SIGNO-MAGNITUD

Existen varias convenciones alternativas para representar números enteros, tanto positivos como negativos. Todas ellas implican tratar el bit más significativo (el más a la izquierda) de la palabra como un bit de signo. Si dicho bit es 0 el número es positivo, y si es 1, el número es negativo.

La forma más sencilla de representación que emplea un bit de signo es la denominada representación signo-magnitud. En una palabra de  $n$  bits, los  $n - 1$  bits de la derecha representan la magnitud del entero. Por ejemplo:

$$+18 = 00010010$$

$$-18 = 10010010 \quad (\text{signo-magnitud})$$

El caso general puede expresarse como sigue:

$$\text{Signo-magnitud: } A = \begin{cases} \sum_{i=0}^{n-2} 2^i a_i & \text{si } a_{n-1} = 0 \\ -\sum_{i=0}^{n-2} 2^i a_i & \text{si } a_{n-1} = 1 \end{cases} \quad (8.1)$$

La representación signo-magnitud posee varias limitaciones. Una de ellas es que la suma y la resta requieren tener en cuenta tanto los signos como sus magnitudes relativas para llevar a cabo la operación en cuestión. Esto debiera quedar claro con la discusión de la Sección 8.3. Otra limitación es que hay dos representaciones del número 0:

$$+0_{10} = 00000000$$

$$-0_{10} = 10000000 \quad (\text{signo-magnitud})$$

Debido a estas limitaciones, raramente se usa la representación en signo-magnitud para implementar en la ALU las operaciones con enteros. En su lugar, el esquema más común es la representación en complemento a dos.

## REPRESENTACIÓN EN COMPLEMENTO A DOS

Al igual que la representación en signo-magnitud, la representación en complemento a dos usa el bit más significativo como bit de signo, facilitando la comprobación de si el entero es positivo o negativo. Difiere de la representación signo-magnitud en la forma de interpretar los bits restantes. La Tabla 8.1 destaca las características clave de la representación y la aritmética en complemento a dos, que serán elaboradas en esta sección y la siguiente.

La mayoría de los tratados sobre representación en complemento a dos se centran en las reglas para la obtención de los números negativos, sin pruebas formales de que el esquema utilizado funcione. En su lugar, la presentación que hacemos de los números enteros en complemento a dos, en ésta sección y en la Sección 8.3, está basada en [DATT93], donde se sugiere que la representación en complemento a dos se entiende mejor definiéndola en términos de una suma ponderada de bits, como hicimos antes para las representaciones sin signo y en signo-magnitud. La ventaja de este tratamiento del tema está en que no queda la duda permanente de que las reglas para las operaciones aritméticas con la notación en complemento a dos no funcionen en algunos casos concretos.

Consideremos un entero de  $n$  bits,  $A$ , representado en complemento a dos. Si  $A$  es positivo, el bit de signo,  $a_{n-1}$ , es cero. Los restantes bits representan la magnitud del número de la misma forma que en la representación signo-magnitud; es decir:

$$A = \sum_{i=0}^{n-2} 2^i a_i \quad \text{para } A \geq 0$$

El número cero se identifica como positivo y tiene, por tanto, un bit de signo 0 y una magnitud de todo ceros. Podemos ver que el rango de los enteros positivos que pueden representarse es desde 0 (todos los bits de magnitud son 0) hasta  $2^{n-1} - 1$  (todos los bits de magnitud son 1). Cualquier número mayor requeriría más bits.

Ahora, para un número negativo  $A$ , el bit de signo,  $a_{n-1}$ , es 1. Los  $n - 1$  bits restantes pueden tomar cualquiera de las  $2^{n-1}$  combinaciones. Por lo tanto, el rango de los enteros negativos que pueden representarse es desde -1 hasta  $-2^{n-1}$ . Sería deseable asignar los bits de los enteros negativos de tal manera que su manipulación aritmética pudiera efectuarse de

Tabla 8.1. Características de la representación numérica y la aritmética en complemento a dos

|                                   |                                                                                                                                                 |
|-----------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| Rango de valores                  | $-2^{n-1}$ a $2^{n-1} - 1$                                                                                                                      |
| N.º de representaciones del cero  | Una                                                                                                                                             |
| Negación                          | Ejecutar el complemento booleano de cada bit y sumar 1 al patrón de bits resultante.                                                            |
| Ampliación de la longitud en bits | Las posiciones de bit extra se añaden a la izquierda, rellenándolas con el valor del bit de signo original.                                     |
| Regla para el desbordamiento      | Si se suman dos números de igual signo (ambos positivos o ambos negativos), hay desbordamiento si, y sólo si, el resultado tiene signo opuesto. |
| Regla para la resta               | Para restar B de A, se toma el complemento a dos de B y se suma a A.                                                                            |

una forma directa, similar a la de los enteros sin signo. En la representación sin signo, para calcular el valor de un entero a partir de su expresión en bits, el peso del bit más significativo es  $+2^{n-1}$ . Como veremos en la Sección 8.3, para una representación con bit de signo resulta que las propiedades aritméticas deseadas se consiguen si el peso del bit más significativo es  $(-2^{n-1})$ . Este es el convenio utilizado para la representación en complemento a dos, obteniéndose la siguiente expresión para los números negativos:

$$\text{Complemento a dos: } A = -2^{n-1}a_{n-1} + \sum_{i=0}^{n-2} 2^i a_i \quad (8.2)$$

Para los enteros positivos  $a_{n-1} = 0$ , de forma que el término  $-2^{n-1}a_{n-1} = 0$ . Así pues, la ecuación (8.2) define la representación en complemento a dos tanto para los números positivos como para los negativos.

Una ilustración gráfica como la mostrada en la Figura 8.2 [BENH92] proporciona una visión más palpable de la representación en complemento a dos. Los círculos se obtienen a partir de los correspondientes segmentos lineales de números, juntando los extremos. Comenzando en cualquier número del círculo, al sumarle un positivo  $k$  (o restarle un negativo  $k$ ) nos desplazamos  $k$  posiciones en el sentido de las agujas del reloj. Restarle un positivo  $k$  (o sumarle un negativo  $k$ ) equivale a desplazarse  $k$  posiciones en sentido contrario a las agujas del reloj. Si la operación realizada hace que se sobrepase el punto en que se juntaron los extremos del segmento, el resultado es incorrecto.

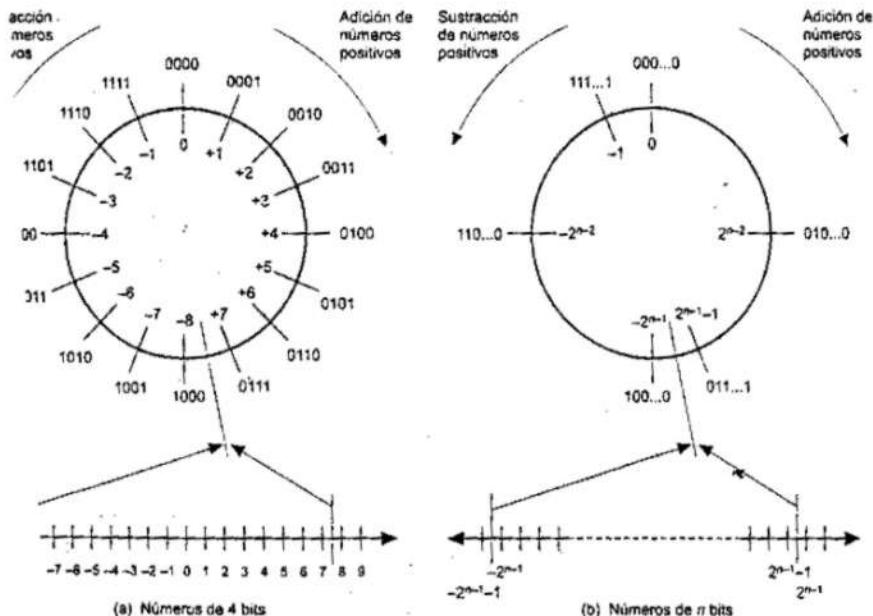


Figura 8.2. Ilustración geométrica de los números enteros en complemento a dos.

Tabla 8.2. Representaciones alternativas de los enteros de 4 bits

| Representación decimal | Representación signo-magnitud | Representación complemento a dos | Representación sesgada |
|------------------------|-------------------------------|----------------------------------|------------------------|
| +7                     | 0111                          | 0111                             | 1111                   |
| +6                     | 0110                          | 0110                             | 1110                   |
| +5                     | 0101                          | 0101                             | 1101                   |
| +4                     | 0100                          | 0100                             | 1100                   |
| +3                     | 0011                          | 0011                             | 1011                   |
| +2                     | 0010                          | 0010                             | 1010                   |
| +1                     | 0001                          | 0001                             | 1001                   |
| +0                     | 0000                          | 0000                             | 1000                   |
| -0                     | 1000                          | —                                | 0111                   |
| -1                     | 1001                          | 1111                             | 0110                   |
| -2                     | 1010                          | 1110                             | 0101                   |
| -3                     | 1011                          | 1101                             | 0100                   |
| -4                     | 1100                          | 1100                             | 0011                   |
| -5                     | 1101                          | 1011                             | 0010                   |
| -6                     | 1110                          | 1010                             | 0001                   |
| -7                     | 1111                          | 1001                             | 0000                   |
| -8                     | —                             | 1000                             | —                      |

La Tabla 8.2 compara, para enteros de 4 bits, las representaciones en signo-magnitud y en complemento a dos. Veremos que la representación en complemento a dos, aunque nos pueda resultar engorrosa, facilita las operaciones aritméticas más importantes, la suma y la resta. Por esta razón, es utilizada casi universalmente como representación de los enteros en los procesadores.

Una ilustración útil de la naturaleza de la representación en complemento a dos es una «caja» de valores, en la que el valor más a la derecha en la caja es 1 ( $2^0$ ), y cada posición consecutiva hacia la izquierda tiene un valor doble a sumar (si el correspondiente bit es 1), hasta la posición más a la izquierda, cuyo valor es a restar. Como se puede ver en la Figura 8.3a, el número en complemento a dos más negativo representable es  $-2^{n-1}$ ; si cualquiera de los bits distintos del de signo es 1, este añade una cantidad positiva al número. Además, está claro que un número negativo debe tener un 1 en la posición más a la izquierda, y un número positivo tendrá un cero en dicha posición. Por tanto, el número positivo mayor es un 0 seguido de todo unos, que es igual a  $2^n - 1$ .

El resto de la Figura 8.3 ilustra el uso de la caja de valores para convertir de complemento a dos a decimal, y de decimal a complemento a dos.

## CONVERSIÓN ENTRE LONGITUDES DE BITS DIFERENTES

A veces se desea tomar un entero de  $n$  bits y almacenarlo en  $m$  bits, siendo  $m > n$ . Esto se resuelve fácilmente en la notación signo-magnitud: simplemente trasladando el bit de signo hasta la nueva posición más a la izquierda y llenando con ceros. Por ejemplo:

$$\begin{aligned}
 +18 &= 00010010 \text{ (signo-magnitud, 8 bits)} \\
 +18 &= 0000000000010010 \text{ (signo-magnitud, 16 bits)} \\
 -18 &= 10010010 \text{ (signo-magnitud, 8 bits)} \\
 -18 &= 1000000000010010 \text{ (signo-magnitud, 16 bits)}
 \end{aligned}$$

|      |    |    |    |   |   |   |   |
|------|----|----|----|---|---|---|---|
| -128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|      |    |    |    |   |   |   |   |

(a) Caja de valores de complemento a dos, de ocho posiciones

|      |    |    |    |   |   |   |   |
|------|----|----|----|---|---|---|---|
| -128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| 1    | 0  | 0  | 0  | 0 | 0 | 1 | 1 |

(b) Conversión a decimal del número binario 10000011

|      |    |    |    |   |   |   |   |
|------|----|----|----|---|---|---|---|
| -128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| 1    | 0  | 0  | 0  | 1 | 0 | 0 | 0 |

(c) Conversión a binario del número decimal -120

Figura 8.3. Utilización de la caja de valores para convertir entre binario en complemento a dos y decimal.

Este procedimiento no funciona con los enteros negativos en complemento a dos. Utilizando el mismo ejemplo:

$$\begin{aligned} +18 &= 00010010 \text{ (complemento a dos, 8 bits)} \\ +18 &= 0000000000010010 \text{ (complemento a dos, 16 bits)} \\ -18 &= 11101110 \text{ (complemento a dos, 8 bits)} \\ -32,658 &= 1000000001101110 \text{ (complemento a dos, 16 bits)} \end{aligned}$$

La penúltima línea anterior puede comprobarse fácilmente mediante la caja de valores de la Figura 8.3. La última línea puede verificarse utilizando la ecuación (8.2).

En su lugar, la regla para los enteros en complemento a dos es trasladar el bit de signo a la nueva posición más a la izquierda y completar con copias del bit de signo. Para números positivos, rellenar con ceros, y para negativos con unos. Así pues, se tiene:

$$\begin{aligned} -18 &= 11101110 \text{ (complemento a dos, 8 bits)} \\ -18 &= 11111111101110 \text{ (complemento a dos, 16 bits)} \end{aligned}$$

Para ver que esta regla funciona, consideremos de nuevo una secuencia de  $n$  dígitos binarios  $a_{n-1}a_{n-2} \dots a_1a_0$  interpretada como entero en complemento a dos  $A$ , tal que su valor es:

$$A = -2^{n-1}a_{n-1} + \sum_{i=0}^{n-2} 2^i a_i$$

Si  $A$  es positivo, la regla funciona claramente. Ahora, supongamos que  $A$  es negativo y que queremos construir una representación de  $m$  bits, con  $m > n$ . Entonces:

$$A = -2^{m-1}a_{m-1} + \sum_{i=0}^{m-2} 2^i a_i$$

Los dos valores deben ser iguales:

$$-2^{m-1} + \sum_{i=0}^{m-2} 2^i a_i = -2^{n-1} + \sum_{i=0}^{n-2} 2^i a_i$$

$$-2^{m-1} + \sum_{i=n-1}^{m-2} 2^i a_i = -2^{n-1}$$

$$2^{n-1} + \sum_{i=n-1}^{m-2} 2^i a_i = 2^{m-1}$$

$$1 + \sum_{i=0}^{n-2} 2^i + \sum_{i=n-1}^{m-2} 2^i a_i = 1 + \sum_{i=0}^{m-2} 2^i$$

$$\sum_{i=n-1}^{m-2} 2^i a_i = \sum_{i=n-1}^{m-2} 2^i$$

$$\Rightarrow a_{m-1} = a_{m-2} = \dots = a_{n-2} = a_{n-1} = 1$$

Al pasar de la primera a la segunda ecuación, se requiere que los  $n - 1$  bits menos significativos no cambien entre las dos representaciones. Entonces, llegamos a la ecuación final, que sólo es cierta si todos los bits desde la posición  $n - 1$  hasta la  $m - 2$  son 1. En consecuencia la regla funciona.

## REPRESENTACIÓN EN COMA FIJA

Finalmente, mencionamos que la representación tratada en esta sección se denomina a veces «de coma fija». Esto es porque la coma de la base (coma binaria) está fija y se supone que a la derecha del bit menos significativo. El programador puede utilizar la misma representación para fracciones binarias escalando los números, de manera que la coma binaria esté implícitamente en alguna otra posición.

**8.3 ARITMÉTICA CON ENTEROS**

Esta sección examina funciones aritméticas comunes con números enteros representados en complemento a dos.

## NEGACIÓN

En la representación signo-magnitud, la regla para obtener el opuesto de un entero es sencilla: invertir el bit de signo. En la notación de complemento a dos, la negación de un entero puede realizarse siguiendo las reglas:

1. Obtener el complemento booleano de cada bit del entero (incluyendo el bit de signo). Es decir, cambiar cada 1 por 0, y cada 0 por 1.
2. Tratando el resultado como un entero binario sin signo, sumarle 1.

Este proceso en dos etapas se denomina transformación a complemento a dos, o obtención del complemento a dos de un entero. Por ejemplo:

$$\begin{array}{r}
 +18 = 00010010 \quad (\text{complemento a dos}) \\
 \text{complemento bit a bit} = 11101101 \\
 \begin{array}{r}
 + \quad 1 \\
 \hline
 11101110 = -18
 \end{array}
 \end{array}$$

Como es de esperar, el opuesto del opuesto es el propio número:

$$\begin{array}{r}
 -18 = 11101110 \quad (\text{complemento a dos}) \\
 \text{complemento bit a bit} = 00010001 \\
 + \qquad \qquad \qquad 1 \\
 \hline
 00010010 = +18
 \end{array}$$

Podemos demostrar la validez de la operación que acabamos de describir utilizando la definición de representación en complemento a dos dada en la ecuación (8.2). De nuevo interpretamos una secuencia de  $n$  dígitos binarios  $a_{n-1}a_{n-2}\dots a_1a_0$  como un entero en complemento a dos  $A$ , tal que su valor es:

$$A = -2^{n-1}a_{n-1} + \sum_{i=0}^{n-2} 2^i a_i$$

Ahora se construye el complemento bit a bit  $\overline{a_{n-1}} \overline{a_{n-2}} \cdots \overline{a_0}$ , y, tratándolo como un entero sin signo, se le suma 1. Finalmente, se interpreta la secuencia de  $n$  bits resultante como un entero en complemento a dos  $B$ , tal que su valor es:

$$B = -2^{n-1} \overline{a_{n-1}} + 1 + \sum_{i=0}^{n-2} 2^i \overline{a_i}$$

Ahora queremos que  $A = -B$ , lo que significa que  $A + B = 0$ . Esto se comprueba fácilmente:

$$\begin{aligned}A + B &= -(a_{n-1} + \overline{a_{n-1}})2^{n-1} + 1 + \left( \sum_{i=0}^{n-2} 2^i(a_i + \overline{a_i}) \right) \\&= -2^{n-1} + 1 + \left( \sum_{i=0}^{n-2} 2^i \right) \\&= -2^{n-1} + 1 + (2^{n-1} - 1) \\&= -2^{n-1} + 2^{n-1} = 0\end{aligned}$$

El desarrollo anterior supone que podemos, primero, tratar el complemento de  $A$  bit a bit como entero sin signo, al objeto de sumarle 1, y entonces, tratar el resultado como un entero en complemento a dos. Hay dos casos especiales a tener en cuenta. En primer lugar, consideremos que  $A = 0$ . En este caso, para una representación con 8 bits:

$$\begin{array}{r}
 0 = 00000000 \quad (\text{complemento a dos}) \\
 \text{complemento bit a bit} = 11111111 \\
 + \qquad \qquad \qquad 1 \\
 \hline
 100000000 = 0
 \end{array}$$

Hay un *acarreo* de la posición de bit más significativa, que es ignorado (el dígito que aparece sombreado). El resultado es que la negación u opuesto del 0 es 0, como debe ser.

El segundo caso especial es más problemático. Si generamos el opuesto de la combinación de bits consistente en un 1 seguido de  $n - 1$  ceros, se obtiene de nuevo el mismo número. Por ejemplo, para palabras de 8 bits:

$$\begin{array}{r} -128 = 10000000 \quad (\text{complemento a dos}) \\ \text{complemento bit a bit} = 01111111 \\ + \qquad \qquad \qquad 1 \\ \hline 10000000 = -128 \end{array}$$

Esta anomalía debe evitarse. El número de combinaciones diferentes en una palabra de 8 bits es  $2^n$ , un número par. Con ellas queremos representar enteros positivos, negativos y el 0. Cuando se representa el mismo número de enteros positivos que de negativos (en signo-magnitud) resultan dos representaciones distintas del 0. Si hay sólo una representación del 0 (en complemento a dos), entonces debe haber un número desigual de números positivos que de negativos representados. En el caso del complemento a dos, hay una representación de  $n$  bits para el  $-2^n$ , pero no para el  $2^n$ .

### SUMA Y RESTA

La suma en complemento a dos se ilustra en la Figura 8.4. Los cuatro primeros ejemplos muestran operaciones correctas. Si el resultado de la operación es positivo, se obtiene un número positivo en la notación binaria ordinaria. Si el resultado de la operación es negativo, conseguimos un número negativo en forma de complemento a dos. Obsérvese que, en algunos casos, hay un bit de acarreo más allá del final de la palabra. Este se ignora.

$$\begin{array}{r} 1001 \\ +0101 \\ \hline 1110 = -2 \end{array}$$

(a)  $(-7) + (+5)$ 

$$\begin{array}{r} 1100 \\ -0100 \\ \hline 10000 = 0 \end{array}$$

(b)  $(-4) + (+4)$ 

$$\begin{array}{r} 0011 \\ +0100 \\ \hline 0111 = 7 \end{array}$$

(c)  $(+3) + (+4)$ 

$$\begin{array}{r} 1100 \\ +1111 \\ \hline 11011 = -3 \end{array}$$

(d)  $(-4) + (-1)$ 

$$\begin{array}{r} 0101 \\ +0100 \\ \hline 1001 = \text{Desbordamiento} \end{array}$$

(e)  $(+5) + (+4)$ 

$$\begin{array}{r} 1001 \\ -1010 \\ \hline 10011 = \text{Desbordamiento} \end{array}$$

(f)  $(-7) + (-6)$ 

Figura 8.4. Suma de números representados en complemento a dos.

En cualquier suma, el resultado puede que sea mayor que el permitido por la longitud de palabra que se está utilizando. Esta condición se denomina desbordamiento (overflow). Cuando ocurre un desbordamiento, la ALU debe indicarlo para que no se intente utilizar el resultado obtenido. Para detectar el desbordamiento se debe observar la siguiente regla: al sumar dos números, cuando ambos son ó bien positivos ó bien negativos, se produce desbordamiento si, y sólo si, el resultado tiene signo opuesto. Las Figuras 8.4e y 8.4f muestran ejemplos de desbordamiento. Obsérvese que el desbordamiento puede ocurrir habiéndose producido o no acarreo.

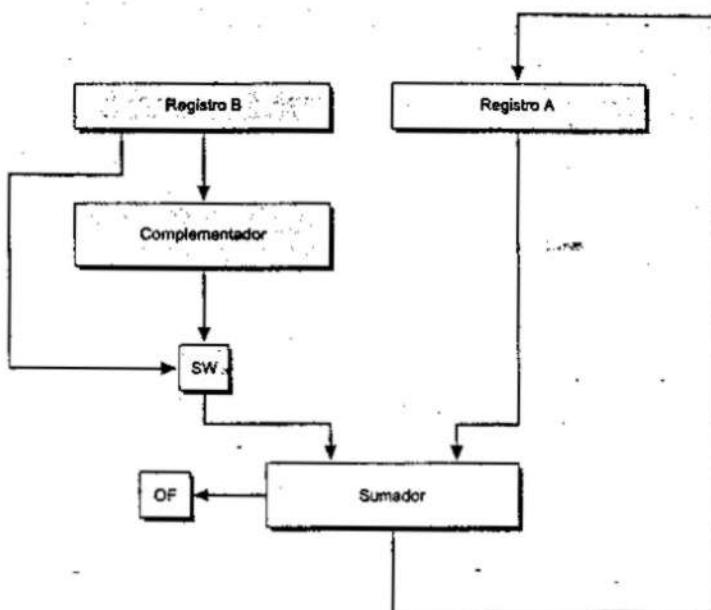
La resta se trata también fácilmente con la siguiente regla: para restar un número (el substraendo) de otro (minuendo), se obtiene el complemento a dos del substraendo y se le suma al minuendo. Así pues, la resta se consigue usando la suma, como se muestra en la Figura 8.5. Los dos últimos ejemplos demuestran que también es aplicable la regla de desbordamiento anterior.

Refiriéndonos de nuevo a la Figura 8.2, se observa también que, para números de  $n$  bits, podemos restar un positivo  $k$  (o sumar un negativo  $k$ ) con un desplazamiento de  $2^n - k$  posiciones en sentido contrario a las agujas del reloj. Pero  $2^n - k$  es lo que hemos definido como el complemento a dos de  $k$ . Esto demuestra gráficamente que la resta se consigue sumando el complemento a dos del substraendo.

La Figura 8.6 sugiere los caminos de datos y elementos hardware necesarios para realizar sumas y restas. El elemento central es un sumador binario, al que se presentan los números a sumar, y produce una suma y un indicador de desbordamiento. El sumador binario trata los dos números como binarios sin signo (una implementación lógica de un sumador se da en el Apéndice A de este libro). Para sumar, los números se presentan al sumador desde dos registros, designados en este caso registros A y B. El resultado es normalmente almacenado en uno de estos registros o en un tercero. La indicación de desbordamiento se almacena en un indicador (o biestable) de desbordamiento (OF, Overflow Flag) de 1 bit (0 = no desbordamiento; 1 = desbordamiento). Para la resta, el substraendo (registro B) se pasa a través de un complementador que presenta su salida al sumador.

|                                                                                              |                                                                                               |
|----------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------|
| $  \begin{array}{r}  0010 \\  +1001 \\  \hline  1011 = -5  \end{array}  $                    | $  \begin{array}{r}  0101 \\  +1110 \\  \hline  10011 = 3  \end{array}  $                     |
| (a) $M = 2 = 0010$                                                                           | (b) $M = 5 = 0101$                                                                            |
| $S = 7 = 0111$                                                                               | $S = 2 = 0010$                                                                                |
| $-S = 1001$                                                                                  | $-S = 1110$                                                                                   |
| <hr/>                                                                                        |                                                                                               |
| $  \begin{array}{r}  1011 \\  +1110 \\  \hline  11001 = -7  \end{array}  $                   | $  \begin{array}{r}  0101 \\  +0010 \\  \hline  0111 = 7  \end{array}  $                      |
| (c) $M = -5 = 1011$                                                                          | (d) $M = 5 = 0101$                                                                            |
| $S = 7 = 0010$                                                                               | $S = -2 = 1110$                                                                               |
| $-S = 1110$                                                                                  | $-S = 0010$                                                                                   |
| <hr/>                                                                                        |                                                                                               |
| $  \begin{array}{r}  0111 \\  +0111 \\  \hline  1110 = \text{Desbordamiento}  \end{array}  $ | $  \begin{array}{r}  1010 \\  +1100 \\  \hline  10110 = \text{Desbordamiento}  \end{array}  $ |
| (e) $M = 7 = 0111$                                                                           | (f) $M = -6 = 1010$                                                                           |
| $S = -7 = 1001$                                                                              | $S = 4 = 0100$                                                                                |
| $-S = 0111$                                                                                  | $-S = 1100$                                                                                   |

Figura 8.5. Substracción de números en la notación de complemento a dos ( $M - S$ ).



OF = Bit de desbordamiento  
 SW = Comutador (selecciona suma o resta)

Figura 8.6. Diagrama de bloques del hardware para la suma y la resta.

## MULTIPLICACIÓN

Comparada con la suma y la resta, la multiplicación es una operación compleja, ya se realice en hardware o en software. En distintos computadores se han utilizado diversos algoritmos. El propósito de esta subsección es dar al lector una idea del tipo de aproximación normalmente utilizada. Comenzaremos con el caso más sencillo de multiplicar dos enteros sin signo (no negativos), y después veremos una de las técnicas más comunes para el producto de números representados en complemento a dos.

### Enteros sin signo

- La Figura 8.7 ilustra la multiplicación de enteros binarios sin signo, que se llevaría a cabo como cuando se utiliza papel y lápiz. Se pueden hacer varias observaciones:

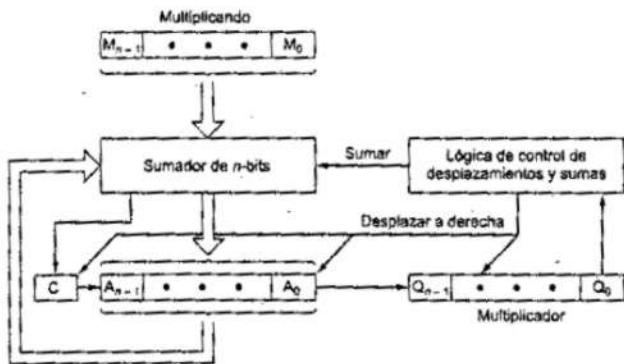
1. La multiplicación implica la generación de productos parciales, uno para cada dígito del multiplicador. Estos productos parciales se suman después para producir el producto final.
2. Los productos parciales se definen fácilmente. Cuando el bit del multiplicador es 0, el producto parcial es 0. Cuando el multiplicador es 1, el producto parcial es el multiplicando.

$$\begin{array}{r}
 1011 \\
 \times 1101 \\
 \hline
 1011 \\
 0000 \\
 1011 \\
 \hline
 1011 \\
 \hline
 10001111
 \end{array}
 \quad \begin{array}{l}
 \text{Multiplicando (11)} \\
 \text{Multiplicador (13)} \\
 \\ \text{Productos Parciales} \\
 \\ \text{Producto (143)}
 \end{array}$$

Figura 8.7. Multiplicación de enteros binarios sin signo.

- El producto total se obtiene sumando los productos parciales. Para esta operación, cada producto parcial sucesivo se desplaza una posición hacia la izquierda con respecto al producto parcial precedente.
- El producto de dos enteros binarios sin signo de  $n$  bits da como resultado un producto de hasta  $2n$  bits de longitud.

En comparación con la aproximación de «papel y lápiz», hay varias modificaciones que se pueden hacer para realizar la operación más eficientemente. En primer lugar, podemos realizar una suma progresiva de los productos parciales, en lugar de esperar hasta el final. Esto evita la necesidad de almacenar todos los productos parciales, necesitándose menos registros. En segundo lugar, podemos ahorrar algún tiempo en la generación de los produc-



(a) Diagrama de bloques

| C | A    | Q    | M    |                               |
|---|------|------|------|-------------------------------|
| 0 | 0000 | 1101 | 1011 | Valores iniciales             |
| 0 | 1011 | 1101 | 1011 | Suma                          |
| 0 | 0101 | 1110 | 1011 | Desplazamiento } Primer ciclo |
| 0 | 0010 | 1111 | 1011 | Suma } Segundo ciclo          |
| 0 | 1101 | 1111 | 1011 | Suma } Tercer ciclo           |
| 0 | 0110 | 1111 | 1011 | Desplazamiento } Tercer ciclo |
| 1 | 0001 | 1111 | 1011 | Suma } Cuarto ciclo           |
| 0 | 1000 | 1111 | 1011 | Desplazamiento } Cuarto ciclo |

(b) Ejemplo de la Figura 8.7 (producto en A, Q)

Figura 8.8. Implementación hardware de la multiplicación de binarios sin signo.

tos parciales. Para cada 1 del multiplicador se requiere un desplazamiento y una suma; pero por cada 0, sólo se necesita el desplazamiento.

La Figura 8.8a muestra una posible implementación, que hace uso de las ideas anteriores. El multiplicador y el multiplicando están ubicados en dos registros ( $Q$  y  $M$ ). Un tercer registro, el registro  $A$ , es también necesario y es inicialmente puesto a 0. Hay también un registro  $C$  de un bit, inicializado a 0, que retiene los posibles bits de acarreo resultantes de las sumas.

El multiplicador actúa de la siguiente manera. La lógica de control lee uno por uno los bits del multiplicador. Si  $Q_0$  es 1, se suma el multiplicando al registro  $A$  y el resultado es almacenado en  $A$ , utilizando el registro  $C$  para el acarreo. Entonces se desplazan todos los bits de los registros  $C$ ,  $A$  y  $Q$  una posición a la derecha, de manera que el bit de  $C$  pasa a  $A_{n-1}$ ,  $A_0$  pasa a  $Q_{n-1}$ , y  $Q_0$  se pierde. Si  $Q_0$  era 0, no se realiza la suma, sólo el desplazamiento. Este proceso se repite para cada bit del multiplicador original. El producto de  $2n$  bits resultante queda en los registros  $A$  y  $Q$ . La Figura 8.9 muestra un diagrama de flujo de la operación, y en la Figura 8.8b se da un ejemplo. Obsérvese que en el ciclo segundo, cuando el bit del multiplicador es 0, no hay operación de suma.

### Multiplicación en complemento a dos

Hemos visto que la suma y la resta pueden realizarse con números en notación de complemento a dos tratándolos como enteros sin signo. Consideremos:

$$\begin{array}{r} 1001 \\ +0011 \\ \hline 1100 \end{array}$$

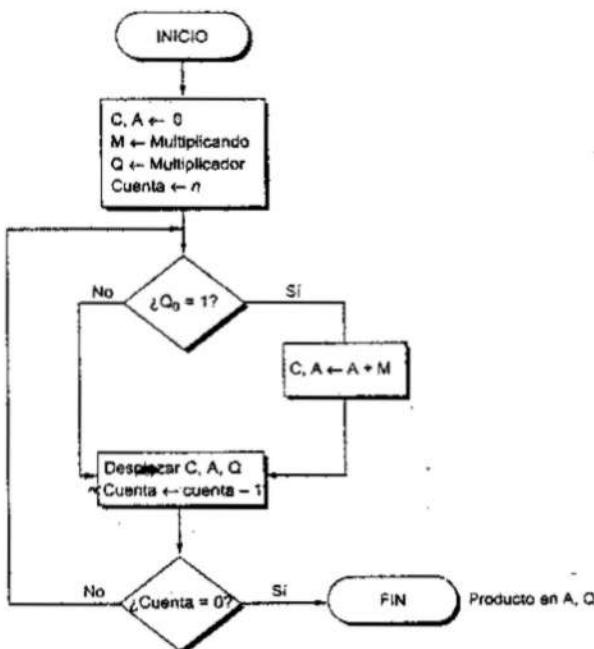


Figura 8.9. Diagrama de flujo para la multiplicación de binarios sin signo.

$$\begin{array}{r}
 1011 \\
 \times 101 \\
 \hline
 00001011 & 1011 \times 1 \times 2^0 \\
 00000000 & 1011 \times 0 \times 2^1 \\
 00101100 & 1011 \times 1 \times 2^2 \\
 \hline
 01011000 & 1011 \times 1 \times 2^3 \\
 \hline
 10001111
 \end{array}$$

Figura 8.10. Multiplicación de dos enteros sin signo de 4 bits para producir un resultado de 8 bits.

Si estos números se consideran enteros sin signo, estamos sumando 9 (1001) más 3 (0011) para obtener 12 (1100). Como enteros en complemento a dos, estamos sumando -7 (1001) a 3 (0011) para obtener -4 (1100).

Desafortunadamente, este simple esquema no es correcto para la multiplicación. Para verlo, consideremos de nuevo la Figura 8.7. Multiplicamos 11 (1011) por 13 (1101) para obtener 143 (10001111). Si interpretamos éstos como números en complemento a dos, tendríamos -5 (1011) por -3 (1101) igual a -13 (10001111). Este ejemplo demuestra que la multiplicación directa no es adecuada si tanto el multiplicando como el multiplicador, son negativos. De hecho, tampoco lo es si alguno de los dos es negativo. Para explicar este comportamiento necesitamos volver sobre la Figura 8.7 y explicar lo que se está haciendo en términos de operaciones con potencias de 2. Recuérdese que cualquier número binario sin signo puede expresarse como suma de potencias de 2. Por tanto:

$$\begin{aligned}
 1101 &= 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\
 &= 2^3 + 2^2 + 2^0
 \end{aligned}$$

Además, el producto de un número binario por  $2^n$  se obtiene desplazando dicho número  $n$  bits hacia la izquierda. Teniendo esto en mente, la Figura 8.10 reestructura la Figura 8.7 para hacer la generación de productos parciales mediante multiplicación explícita. La única diferencia en la Figura 8.10 es que reconoce que los productos parciales debieran verse como números de  $2n$  bits generados a partir del multiplicando de  $n$  bits.

Así pues, el multiplicando de 4 bits, 1011, como entero sin signo, es almacenado en una palabra de 8 bits como 00001011. Cada producto parcial (distinto del correspondiente a  $2^0$ ) consiste en dicho número desplazado a la izquierda, con las posiciones de la derecha llenadas con ceros (por ejemplo, un desplazamiento a la izquierda en dos posiciones produce 00101100).

Ahora podemos demostrar cómo la multiplicación directa no es correcta si el multiplicando es negativo. El problema es que cada contribución del multiplicando negativo como producto parcial tiene que ser un número negativo en un campo de  $2n$  bits; los bits de signo de los productos parciales deben estar alineados. Esto se demuestra en la Figura 8.11, que muestra el producto de 1001 por 0011. Si estos se tratan como enteros sin signo, se realiza el producto  $9 \times 3 = 27$ . Sin embargo, si 1001 se interpreta como -7 en complemento a dos, cada

$$\begin{array}{r}
 1001 \quad (9) \\
 \times 0011 \quad (3) \\
 \hline
 00001001 \quad (1001) \times 2^0 \\
 00010010 \quad (1001) \times 2^1 \\
 00011011 \quad (27)
 \end{array}$$

(a) Enteros sin signo

$$\begin{array}{r}
 1001 \quad (-7) \\
 \times 0011 \quad (3) \\
 \hline
 11111001 \quad (-7) \times 2^0 = (-7) \\
 11110010 \quad (-7) \times 2^1 = (-14) \\
 11101011 \quad (-21)
 \end{array}$$

(b) Enteros en complemento a dos

Figura 8.11. Comparación del producto de enteros sin signo y en complemento a dos.

el producto parcial debe ser un número negativo en complemento a dos de  $2n$  (es decir 8 bits), como muestra la Figura 8.11b. Obsérvese que eso podría hacerse rellenando la parte izquierda de cada producto parcial con unos.

Debiera quedar también claro que, si el multiplicador es negativo, la multiplicación directa tampoco es correcta. La razón es que los bits del multiplicador ya no se corresponden con los desplazamientos o productos que deben producirse. Por ejemplo, el número decimal  $-3$  se representa, con 4 bits en complemento a dos, como  $1101$ . Si simplemente tomamos los productos parciales basándonos en cada posición de bit, tendríamos la siguiente correspondencia:

$$1101 \rightarrow -(1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0) = -(2^3 + 2^2 + 2^0)$$

De hecho, lo que se quiere es  $-(2^1 + 2^0)$ . Por tanto este multiplicador no puede utilizarse directamente en la forma anteriormente descrita.

Hay varias maneras de salir de este dilema. Una sería convertir, tanto el multiplicando como el multiplicador, en números positivos, realizar el producto, y obtener después el complemento a dos del resultado si, y sólo si, el signo de los dos números iniciales difiere. Los diseñadores han preferido utilizar técnicas que no requieren esta etapa de transformación final. Una de las técnicas más comunes es el algoritmo de Booth. Este algoritmo tiene la ventaja adicional de acelerar el proceso de multiplicación respecto de una aproximación más directa.

El algoritmo de Booth se ilustra en la Figura 8.12 y puede describirse como sigue. Como

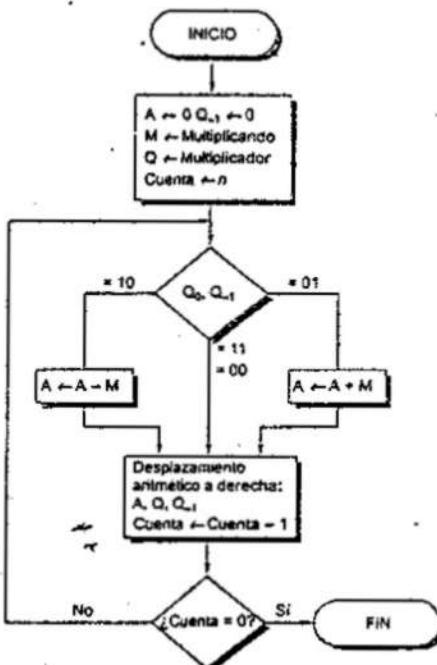


Figura 8.12. Algoritmo de Booth para la multiplicación en complemento a dos.

antes, el multiplicador y el multiplicando se ubican en los registros Q y M respectivamente. Hay también un registro de un bit con una ubicación lógica a la derecha del bit menos significativo ( $Q_0$ ) del registro Q, y que denominamos  $Q_{-1}$ ; explicaremos brevemente su uso. El producto resultante aparecerá en los registros A y Q. A y  $Q_{-1}$  se fijan inicialmente a 0. Como antes, la lógica de control recorre los bits del multiplicador uno por uno. Ahora, al examinar cada bit, también se comprueba el bit a su derecha. Si los dos son iguales (1-1 o 0-0), todos los bits de los registros A, Q, y  $Q_{-1}$  se desplazan un bit a la derecha. Si dichos bits difieren, el multiplicando se suma o se resta al registro A, según que los dos bits sean 0-1 o 1-0. A continuación de la suma o resta se realiza un desplazamiento a la derecha. En cualquier caso, el desplazamiento a la derecha es tal, que el bit más a la izquierda de A, es decir  $A_{n-1}$ , no sólo se desplaza a  $A_{n-2}$ , sino que también queda en  $A_{n-1}$ . Esto es necesario para preservar el signo del número contenido en la pareja de registros A y Q. Este desplazamiento se denomina aritmético, ya que preserva el bit de signo.

La Figura 8.13 muestra la secuencia de eventos para multiplicar 7 por 3 con el algoritmo de Booth. La misma operación se describe de manera más compacta en la Figura 8.14a. El resto de la Figura 8.14 da otros ejemplos del algoritmo. Como puede verse, actúa correctamente con cualquier combinación de números positivos y negativos. Obsérvese también la eficiencia del algoritmo. Los bloques de unos o de ceros se saltan, con un promedio de sólo una suma o resta por bloque.

¿Por qué se comporta correctamente el algoritmo de Booth? Consideré primero el caso en que el multiplicador sea positivo; en particular, un multiplicador positivo consistente en un bloque de unos con ceros a ambos lados (por ejemplo 00011110). Como sabemos, la multiplicación puede obtenerse sumando adecuadamente copias desplazadas del multiplicando:

$$\begin{aligned} M \times (00011110) &= M \times (2^4 + 2^3 + 2^2 + 2^1) \\ &= M \times (16 + 8 + 4 + 2) \\ &= M \times 30 \end{aligned}$$

El número de tales operaciones puede reducirse a dos si observamos que:

$$2^n + 2^{n-1} + \dots + 2^{n-k} = 2^{n-1} - 2^{n-k} \quad (8.3)$$

| A    | i    | $Q_{-1}$ | M    |                                |
|------|------|----------|------|--------------------------------|
| 0000 | 0011 | 0        | 0111 | Valores iniciales              |
| 1001 | 0011 | 0        | 0111 | $A \leftarrow A - M$           |
| 0100 | 1001 | 1        | 0111 | Desplazamiento } Primero ciclo |
| 1110 | 0100 | 1        | 0111 | Desplazamiento } Segundo ciclo |
| 0101 | 0100 | 1        | 0111 | $A \leftarrow A + M$           |
| 0010 | 1010 | 0        | 0111 | Desplazamiento } Tercer ciclo  |
| 0001 | 0101 | 0        | 0111 | Desplazamiento } Cuarto ciclo  |

Figura 8.13. Ejemplo de aplicación del algoritmo de Booth.

|                                                                                                                                               |                                                                                                                                               |
|-----------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| $  \begin{array}{r}  0111 \\  \times 0011 \\  \hline  11111001 \\  00000000 \\  \hline  000111 \\  00010101 \\  \hline  (21)  \end{array}  $  | $  \begin{array}{r}  0111 \\  \times 1101 \\  \hline  11111001 \\  00001111 \\  \hline  111001 \\  11101011 \\  \hline  (-21)  \end{array}  $ |
| (a) $(7) \times (3) = (21)$                                                                                                                   | (b) $(7) \times (-3) = (-21)$                                                                                                                 |
| $  \begin{array}{r}  1001 \\  \times 0011 \\  \hline  11100111 \\  00000000 \\  \hline  111001 \\  11101011 \\  \hline  (-21)  \end{array}  $ | $  \begin{array}{r}  1001 \\  \times 1101 \\  \hline  11100111 \\  111001 \\  11101011 \\  \hline  (-21)  \end{array}  $                      |
| (c) $(-7) \times (3) = (-21)$                                                                                                                 | (d) $(-7) \times (-3) = (21)$                                                                                                                 |

Figura 8.14. Ejemplos de la utilización del algoritmo de Booth.

Por tanto:

$$\begin{aligned}
 M \times (00011110) &= M \times (2^5 - 2^1) \\
 &= M \times (32 - 2) \\
 &= M \times 30
 \end{aligned}$$

Así pues, el producto puede generarse mediante una suma y una resta del multiplicando. Este esquema se extiende a cualquier número de bloques de unos del multiplicador, incluyendo el caso en que un solo 1 es tratado como bloque. Así:

$$\begin{aligned}
 M \times (01111010) &= M \times (2^6 + 2^5 + 2^4 + 2^3 + 2^1) \\
 &= M \times (2^7 - 2^3 + 2^2 - 2^1)
 \end{aligned}$$

El algoritmo de Booth obedece a este esquema, realizando una resta cuando se encuentra el primer 1 del bloque (1-0), y una suma cuando se encuentra el final (0-1) del bloque.

Para comprobar que el mismo esquema es adecuado en el caso de un multiplicador negativo, necesitamos observar lo siguiente. Sea  $X$  un número negativo en notación de complemento a dos:

$$\text{Representación de } X = (1x_{n-1}x_{n-2} \cdots x_1x_0)$$

Entonces el valor de  $X$  puede expresarse como sigue:

$$X = -2^{n-1} + (x_{n-1} \times 2^{n-2}) + (x_{n-2} \times 2^{n-3}) + \cdots + (x_1 \times 2^1) + (x_0 \times 2^0) \quad (8.4)$$

El lector puede verificarlo aplicando el algoritmo a los números de la Tabla 8.2.

Ahora sabemos que el bit más a la izquierda de  $X$  es 1, ya que  $X$  es negativo. Suponga que el bit 0 más a la izquierda está en la posición  $k$ -ésima. Entonces  $X$  será en la forma:

$$\text{Representación de } X = (111 \cdots 10x_{k-1}x_{k-2} \cdots x_1x_0) \quad (8.5)$$

Entonces el valor de  $X$  es:

$$X = -2^{n-1} + 2^{n-2} + \dots + 2^{k+1} + (x_{k-1} \times 2^{k-1}) + \dots + (x_0 \times 2^0) \quad (8.6)$$

Utilizando la ecuación (8.3) podemos decir que:

$$2^{n-2} + 2^{n-3} + \dots + 2^{k+1} = 2^{n-1} - 2^{k+1}$$

Reagrupando, se tiene:

$$-2^{n-1} + 2^{n-2} + 2^{n-3} + \dots + 2^{k+1} = -2^{k+1} \quad (8.7)$$

Sustituyendo la ecuación (8.7) en la (8.6), tenemos:

$$X = -2^{k+1} + (x_{k-1} \times 2^{k-1}) + \dots + (x_0 \times 2^0) \quad (8.8)$$

Al fin podemos volver sobre el algoritmo de Booth. Recordando la representación de  $X$  [ecuación (8.5)], está claro que todos los bits desde  $x_0$  hasta el 0 más a la izquierda son manipulados de forma apropiada, ya que producen todos los términos de la ecuación (8.8) excepto el  $(-2^{k+1})$ . Cuando el algoritmo recorre hasta el 0 más a la izquierda y encuentra el 1 siguiente ( $2^{k+1}$ ), ocurre una transición 1-0 y tiene lugar una resta  $(-2^{k+1})$ . Este es el término restante de la ecuación (8.8).

Como ejemplo, considere la multiplicación de algún multiplicando por  $(-6)$ . En la representación de complemento a dos, utilizando una palabra de 8 bits,  $(-6)$  se expresa como 11111010. Por la ecuación (8.4) sabemos que:

$$-6 = -2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^1$$

como puede verificar fácilmente el lector. Por tanto:

$$M \times (11111010) = M \times (-2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^1)$$

Usando la ecuación (8.7),

$$M \times (11111010) = M \times (-2^3 + 2^1)$$

que, como el lector puede verificar, es también  $M \times (-6)$ . Finalmente, siguiendo nuestra línea de razonamiento anterior:

$$M \times (11111010) = M \times (-2^3 + 2^2 - 2^1)$$

Podemos ver que el algoritmo de Booth se ajusta a este esquema. Realiza una resta cuando se encuentra el primer 1 (1-0), una suma cuando se encuentra (0-1), y, finalmente, resta otra vez cuando encuentra el primer 1 del siguiente bloque de unos. Por consiguiente, el algoritmo de Booth realiza menos sumas y restas que un algoritmo directo.

## DIVISIÓN

La división es algo más compleja que la multiplicación pero está basada en los mismos principios generales. Como antes, la base para el algoritmo es la aproximación de «papel y lápiz». Y la operación conlleva repetidos desplazamientos y sumas o restas.

La Figura 8.15 muestra un ejemplo de división larga de enteros binarios sin signo. Es instructivo describir en detalle el proceso. Primero se examinan los bits del dividendo de

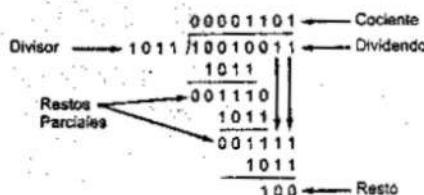


Figura 8.15. División de enteros binarios sin signo.

izquierda a derecha, hasta que el conjunto de bits examinados represente un número mayor o igual que el divisor, o, en otras palabras, hasta que el divisor sea capaz de dividir al número. Hasta que eso ocurre, se van colocando ceros en el cociente de izquierda a derecha. Cuando dicho evento ocurre, se coloca un 1 en el cociente y se resta el divisor del dividendo parcial. Al resultado se le denomina *resto parcial*. Desde este punto en adelante, la división sigue un patrón cíclico. En cada ciclo, se añaden bits adicionales del dividendo al resto parcial, hasta que el resultado sea mayor o igual que el divisor. Como antes, de este número se resta el divisor para producir un nuevo resto parcial. El proceso continúa hasta que se acaban los bits del dividendo.

La Figura 8.16 muestra un algoritmo máquina que corresponde al proceso de división larga discutido. El divisor se ubica en el registro M, y el dividendo en el registro Q. En cada paso, los registros A y Q son desplazados conjuntamente un bit a la izquierda. M es restado de A para determinar si A divide el resto parcial<sup>1</sup>. Si esto se cumple, entonces  $Q_0$  se hace 1. Si no,  $Q_0$  se hace 0, y M debe sumarse de nuevo a A para restablecer el valor anterior. La cuenta, entonces, se decremente, y el proceso continúa hasta  $n$  pasos. Al final, el cociente queda en el registro Q y el resto en el A.

Este proceso puede, con cierta dificultad, aplicarse también a números negativos. Aquí damos una posible aproximación para números en complemento a dos. En la Figura 8.17 se muestran varios ejemplos de esta aproximación. El algoritmo puede resumirse como sigue:

1. Cargar el divisor en el registro M y el dividendo en los registros A y Q. El dividendo debe estar expresado como número en complemento a dos de  $2n$  bits. Por ejemplo, el número de 4 bits 0111 pasa a ser 00000111, y el 1001 pasa a 11111001.
2. Desplazar A y Q una posición de bit a la izquierda.
3. Si M y A tienen el mismo signo, ejecutar  $A \leftarrow A - M$ ; si no,  $A \leftarrow A + M$ .
4. La operación anterior tiene éxito si el signo de A es el mismo antes y después de la operación.
  - a) Si la operación tiene éxito o ( $A = 0$  AND  $Q = 0$ ), entonces hacer  $Q_0 \leftarrow 1$ .
  - b) Si la operación no tiene éxito y ( $A \neq 0$  OR  $Q \neq 0$ ), entonces hacer  $Q_0 \leftarrow 0$ , y restablecer el valor anterior de A.
5. Repetir los pasos 2 a 4 tantas veces como número de bits tenga Q.
6. El resto está en A. Si los signos del divisor y del dividendo eran iguales, el cociente está en Q; si no, el cociente correcto es el complemento a dos de Q.

<sup>1</sup> Es una resta de enteros sin signo. Si se produce un scarreo negativo (adeudo) a partir del bit más significativo, el resultado es negativo.

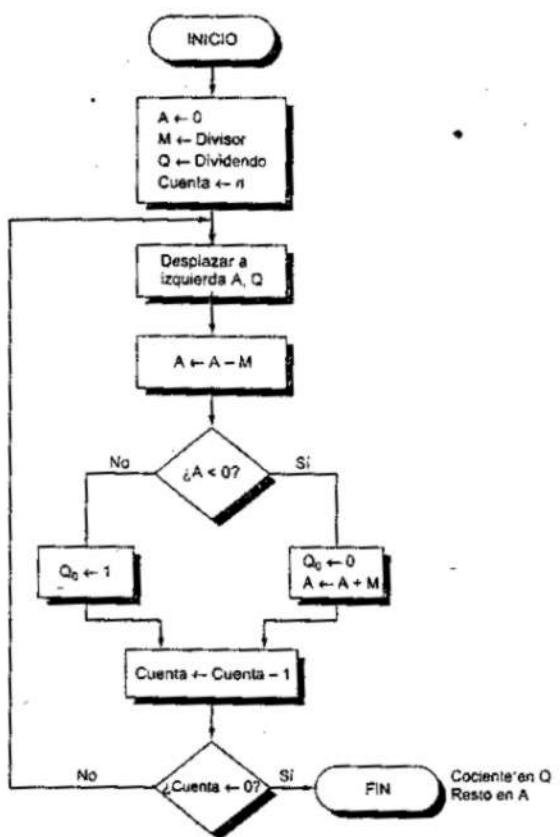


Figura 8.16. Diagrama de flujo para la división de binarios sin signo.

El lector notará en la Figura 8.17 que  $(-7) \div (3)$  y  $(7) \div (-3)$  producen restos diferentes. Esto es debido a que el resto se define como

$$D = Q \times V + R$$

donde:

$$D = \text{dividendo}$$

$$Q = \text{cociente}$$

$$V = \text{divisor}$$

$$R = \text{resto}$$

Los resultados de la Figura 8.17 son consistentes con dicha fórmula.

| A              | Q    | M = 0011                 | A              | Q    | M = 1101                 |
|----------------|------|--------------------------|----------------|------|--------------------------|
| 0000           | 0111 | Valor Inicial            | 0000           | 0111 | Valor Inicial            |
| 0000           | 1110 | Desplazamiento           | 0000           | 1110 | Desplazamiento           |
| 1101           |      | Restar                   | 1101           |      | Sumar                    |
| 0000           | 1110 | Restablecer              | 0000           | 1110 | Restablecer              |
| 0001           | 1100 | Desplazamiento           | 0001           | 1100 | Desplazamiento           |
| 1110           |      | Restar                   | 1110           |      | Sumar                    |
| 0001           | 1100 | Restablecer              | 0001           | 1100 | Restablecer              |
| 0011           | 1000 | Desplazamiento           | 0011           | 1000 | Desplazamiento           |
| 0000           |      | Restar                   | 0000           |      | Sumar                    |
| 0000           | 1001 | Poner Q <sub>0</sub> = 1 | 0000           | 1001 | Poner Q <sub>0</sub> = 1 |
| 0001           | 0010 | Desplazamiento           | 0001           | 0010 | Desplazamiento           |
| 1110           |      | Restar                   | 1110           |      | Sumar                    |
| 0001           | 0010 | Restablecer              | 0001           | 0010 | Restablecer              |
| (a) (7) + (3)  |      |                          | (b) (7) + (3)  |      |                          |
| A              | Q    | M = 0011                 | A              | Q    | M = 1101                 |
| 1111           | 1001 | Valor Inicial            | 1111           | 1001 | Valor Inicial            |
| 1111           | 0010 | Desplazamiento           | 1111           | 0010 | Desplazamiento           |
| 0010           |      | Sumar                    | 0010           |      | Restar                   |
| 1111           | 0010 | Restablecer              | 1111           | 0010 | Restablecer              |
| 1110           | 0100 | Desplazamiento           | 1110           | 0100 | Desplazamiento           |
| 0001           |      | Sumar                    | 0001           |      | Restar                   |
| 1110           | 0100 | Restablecer              | 1110           | 0100 | Restablecer              |
| 1100           | 1000 | Desplazamiento           | 1100           | 1000 | Desplazamiento           |
| 1111           |      | Sumar                    | 1111           |      | Restar                   |
| 1111           | 1001 | Poner Q <sub>0</sub> = 1 | 1111           | 1001 | Poner Q <sub>0</sub> = 1 |
| 1111           | 0010 | Desplazamiento           | 1111           | 0010 | Desplazamiento           |
| 0010           |      | Sumar                    | 0010           |      | Restar                   |
| 1111           | 0010 | Restablecer              | 1111           | 0010 | Restablecer              |
| (c) (-7) + (3) |      |                          | (d) (-7) + (3) |      |                          |

Figura 8.17. Ejemplos de división en complemento a dos.

## 8.4. REPRESENTACIÓN EN COMA FLOTANTE

### FUNDAMENTOS

Con una notación de coma fija (por ejemplo, la representación en complemento a dos) es posible representar un rango de enteros positivos y negativos centrado en el 0. Asumiendo una coma binaria fija, dicho formato permite también representar números con parte fraccionaria.

La aproximación anterior tiene limitaciones. Los números muy grandes no pueden representarse, ni tampoco las fracciones muy pequeñas. Además, en la división de dos números grandes puede perderse la parte fraccionaria del cociente.

Para números decimales, esta limitación se supera utilizando la notación científica. Así, 976.000.000.000.000 puede representarse como  $9,76 \times 10^{14}$ , y 0,0000000000000976 puede expresarse como  $9,76 \times 10^{-14}$ . Lo que se ha hecho es mover dinámicamente la coma decimal a una posición conveniente y utilizar el exponente de 10 para mantener registrada la posición de la coma. Esto permite representar un rango de números muy grandes y muy pequeños con sólo unos cuantos dígitos.

Esa misma técnica puede aplicarse a números binarios. Podemos representar un número en la forma:

$$\pm S \times B^{\pm E}$$

Este número puede almacenarse en una palabra binaria con tres campos:

- Signo: más o menos
- Parte significativa o mantisa (S, «significand»)
- Exponente (E)

La base B está implícita y no necesita memorizarse, ya que es la misma para todos los números.

Las reglas utilizadas en la representación de números binarios en coma flotante se explican mejor con un ejemplo. La Figura 8.18a muestra un formato típico de coma flotante de 32 bits. El bit más a la izquierda contiene el signo del número (0 = positivo, 1 = negativo). El valor del exponente se almacena en los bits 1 a 8. La representación utilizada se conoce con el término de «sesgo». Un valor fijo, llamado «sesgo», se resta de este campo para conseguir el valor del exponente verdadero. Normalmente, el sesgo tiene un valor ( $2^{k-1} - 1$ ), donde k es el número de bits en el exponente binario. En este caso, un campo de 8 bits comprende números entre 0 y 255. Con un sesgo de 127, los valores verdaderos de exponente varían en el rango -127 a +128. En este ejemplo, se supone la base 2.

La Tabla 8.2 mostraba la representación sesgada para enteros de 4 bits. Observe que, cuando los bits de una representación sesgada se tratan como enteros sin signo, las magnitudes relativas de los números no cambian. Por ejemplo, tanto en la representación sin signo como en la sesgada, el número más grande es el 1111 y el más pequeño el 0000. Esto no se cumple para las representaciones en signo-magnitud, en complemento a dos o en complemento a uno. Una ventaja de la representación sesgada es que los números en coma flotante no negativos pueden ser tratados de la misma forma que los enteros al efectuar comparaciones.

La parte final de la palabra (23 bits en el ejemplo de la Figura 8.18) es la parte significativa o mantisa. Ahora, cualquier número en coma flotante puede expresarse de distintas formas. Así, son equivalentes las siguientes representaciones:

$$0,110 \times 2^5$$

$$110 \times 2^2$$

$$0,0110 \times 2^6$$

Y así sucesivamente. Para simplificar los cálculos con números en coma flotante se requiere usualmente que estén normalizados. Para nuestro ejemplo, un número normalizado es aquel en la forma

$$\pm 0,1\ bbb\dots b \times 2^{\pm E}$$

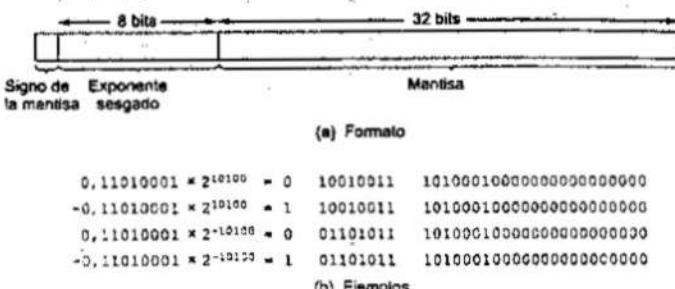


Figura 8.18. Formato típico de 32 bits en coma flotante.

donde cada b es un dígito binario (1 o 0). Esto implicaría que el bit más a la izquierda de la mantisa fuera siempre 1. Ya que es obviamente innecesario almacenar este bit, está implícito. Así, el campo de 23 bits se emplea para albergar una mantisa de 24 bits con un valor entre 0,5 y 1,0.

La Figura 8.18b da algunos ejemplos de números almacenados en este formato. Observe las siguientes características:

- El signo se almacena en el primer bit de la palabra.
- El primer bit de la mantisa original siempre es 1 y no necesita almacenarse en el campo de mantisa.
- Se suma 127 al exponente original para almacenarlo en el campo de exponente.
- La base es 2.

Con esta representación, la Figura 8.19 indica el rango de números que pueden representarse en una palabra de 32 bits. Usando la notación entera en complemento a dos, pueden representarse todos los enteros desde  $-2^{31}$  hasta  $2^{31} - 1$ , con un total de  $2^{32}$  números diferentes. Con el ejemplo de formato en coma flotante de la Figura 8.18 son posibles los siguientes rangos de números:

- Números negativos entre  $-(1 - 2^{-24}) \times 2^{128}$  y  $-0,5 \times 2^{-127}$
- Números positivos entre  $0,5 \times 2^{-127}$  y  $(1 - 2^{-24}) \times 2^{128}$

En la recta de los números reales hay cinco regiones excluidas de dichos rangos:

- Los números negativos menores que  $-(1 - 2^{-24}) \times 2^{128}$ , región denominada desbordamiento negativo.
- Los números negativos mayores que  $-0,5 \times 2^{-127}$ , denominada desbordamiento a cero negativo.
- El cero.
- Los números positivos menores que  $0,5 \times 2^{-127}$ , denominada desbordamiento a cero positivo.
- Los números positivos mayores que  $(1 - 2^{-24}) \times 2^{128}$ , denominada desbordamiento positivo.

La representación indicada no contempla un valor para el 0. Sin embargo, como veremos, en la práctica se incluye una combinación de bits especial para designar el cero. Un desbor-

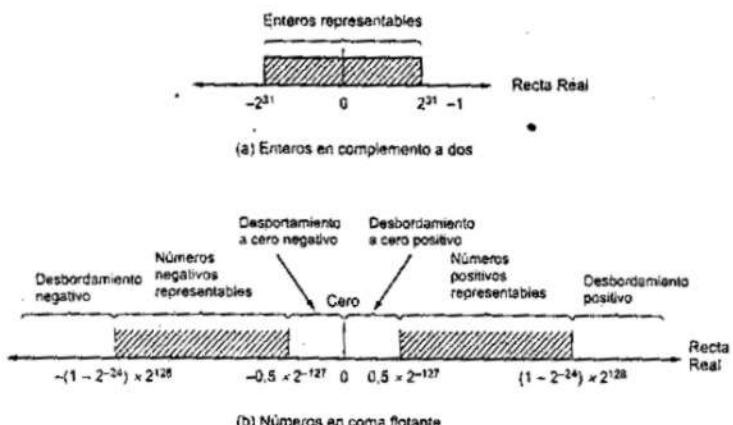


Figura 8.19. Números representables en formatos típicos de 32 bits.

damiento ocurre cuando una operación aritmética da lugar a un número cuyo exponente es mayor que 128 (por ejemplo,  $2^{120} \times 2^{100} = 2^{220}$  produciría desbordamiento). Un desbordamiento a cero ocurre cuando una magnitud fraccionaria es demasiado pequeña (por ejemplo,  $2^{-120} \times 2^{-100} = 2^{-220}$ ). Un desbordamiento a cero es un problema menos serio porque el resultado puede generalmente aproximarse satisfactoriamente por 0.

Es importante observar que, con la notación en coma flotante, no estamos representando más valores individuales. El máximo número de valores diferentes que pueden representarse con 32 bits es 2<sup>32</sup>. Lo que hemos hecho es repartir dichos números a lo largo de dos intervalos, uno positivo y otro negativo.

Obsérvese también que, al contrario de lo que ocurre con los números en coma fija, los números representados en la notación de coma flotante no están espaciados por igual a lo largo de la recta real. Los posibles valores están más próximos cerca del origen y más separados a medida que nos alejamos de él, según se ilustra en la Figura 8.20. Este es uno de los inconvenientes del cálculo en coma flotante: muchos cálculos producen resultados que no son exactos, y tienen que redondearse al valor más próximo representable con la notación.

En el formato indicado en la Figura 8.18 existe un compromiso entre rango y precisión. El ejemplo muestra 8 bits dedicados al exponente y 23 bits a la mantisa. Si incrementamos el número de bits del exponente, ampliamos el rango de números representables. Pero, ya que sólo puede expresarse un número dado de valores diferentes, habríamos reducido la densidad de dichos números y, en consecuencia, la precisión. La única forma de incrementar, tanto el



Figura 8.20. Densidad de los números en coma flotante.

rango como la precisión, es utilizar más bits. Por tanto, la mayoría de los computadores ofrecen la posibilidad de utilizar, al menos, números de precisión simple y números en doble precisión. Por ejemplo, un formato de precisión simple podría ser de 32 bits, y uno de precisión doble de 64 bits.

Existe pues, un compromiso entre el número de bits del exponente y el de la mantisa. En realidad se trata de algo más complicado, ya que la base del exponente no tiene por qué ser 2. Por ejemplo, la arquitectura IBM S/390 utiliza la base 16. El formato consiste en un exponente de 7 bits y una mantisa de 24 bits. Así, por ejemplo:

$$0.11010001 \times 2^{10100} = 0.11010001 \times 16^{101}$$

y el exponente que se almacena representa el 5 en lugar del 20.

La ventaja de utilizar una base mayor es que se puede conseguir un rango de números mayor para el mismo número de bits de exponente. Pero recuerde que con ello no incrementamos el número de valores diferentes que pueden ser representados. Así, para un formato dado, una base del exponente mayor proporciona un rango mayor a costa de menor precisión.

### ESTÁNDAR DEL IEEE PARA LA REPRESENTACIÓN BINARIA EN COMA FLOTANTE

La representación en coma flotante más importante es la definida en la norma o estándar 754 del IEEE [IEEE85]. Este estándar se desarrolló para facilitar la portabilidad de los programas de un procesador a otro y para alentar el desarrollo de programas numéricos sofisticados. Este estándar ha sido ampliamente adoptado y se utiliza prácticamente en todos los procesadores y coprocesadores aritméticos actuales.

El estándar del IEEE define, tanto el formato simple de 32 bits como el doble de 64 bits (Figura 8.21), con exponentes de 8 y 11 bits respectivamente. La base implícita es 2. Además, define dos formatos ampliados, simple y doble, cuya forma exacta depende de la implementación (puede variar de unos procesadores a otros). Estos formatos ampliados incluyen bits adicionales en el exponente (rango ampliado o extendido) y en la mantisa (precisión ampliada). Los formatos ampliados se utilizan para cálculos intermedios. Con su mayor precisión, dichos formatos reducen la posibilidad de que el resultado final se vea deteriorado por un error

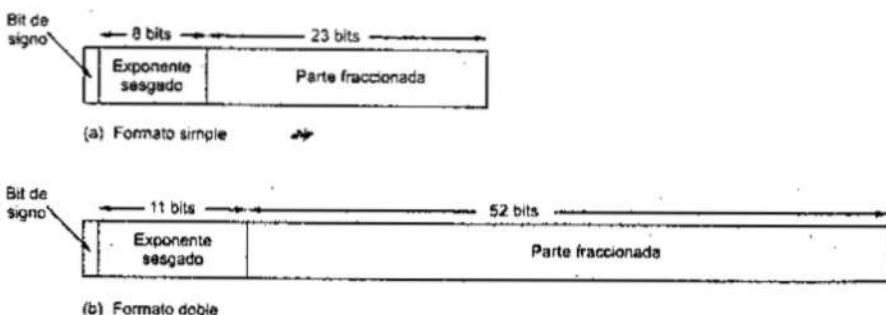


Figura 8.21. Formatos IEEE 754.

Tabla 8.3. Parámetros del formato IEEE 754

| Parámetro                       | Formato              |                 |                        |                 |
|---------------------------------|----------------------|-----------------|------------------------|-----------------|
|                                 | Simple               | Simple ampliado | Doble                  | Doble ampliado  |
| Longitud de palabra (bits)      | 32                   | $\geq 43$       | 64                     | $\geq 79$       |
| Longitud de exponente (bits)    | 8                    | $\geq 11$       | 11                     | $\geq 15$       |
| Sesgo del exponente             | 127                  | sin especificar | 1023                   | sin especificar |
| Exponente máximo                | 127                  | $\geq 1023$     | 1023                   | $\geq 16.383$   |
| Exponente mínimo                | -126                 | $\leq -1022$    | -1022                  | $\leq -16.382$  |
| Rango de números (base 10)      | $10^{-37}, 10^{+38}$ | sin especificar | $10^{-308}, 10^{+308}$ | sin especificar |
| Longitud de mantisa (bits)*     | 23                   | $\geq 31$       | 52                     | $\geq 63$       |
| Número de exponentes            | 254                  | sin especificar | 2048                   | sin especificar |
| Número de fracciones o mantisas | $2^{23}$             | sin especificar | $2^{52}$               | sin especificar |
| Número de valores               | $1.98 \times 2^{23}$ | sin especificar | $1.99 \times 2^{52}$   | sin especificar |

\* Excluyendo el bit implícito.

excesivo de redondeo; con su mayor intervalo de variación, también reducen la posibilidad de un desbordamiento intermedio que abulta un cálculo cuyo resultado habría sido representable en un formato básico. Una motivación adicional para el formato ampliado simple es que proporciona algunas de las ventajas del formato doble sin incurrir en la penalización de tiempo normalmente asociada con la elevada precisión. La Tabla 8.3 resume las características de los cuatro formatos indicados.

En los formatos del IEEE 754, los patrones de bits se interpretan de la manera habitual. Algunas combinaciones se emplean para representar valores especiales. La Tabla 8.4 indica los valores asignados a ciertos patrones de bits. Los valores extremos de exponente consistentes en todo ceros (0) y todo unos (255 en el formato simple y 2047 en el doble) definen valores especiales. Se representan las siguientes clases de números:

- Para valores de exponente desde 1 hasta 254 en el formato simple y desde 1 hasta 2046 en el formato doble, se representan números en coma flotante normalizados distintos de cero. El exponente está sesgado, siendo el rango de exponentes de (-126) a +127 en el formato simple y de (-1022) a +1023 en el doble. Un número normalizado debe tener un bit 1 a la izquierda de la coma binaria; este bit está implícito, dando una mantisa efectiva de 24 o 53 bits (denominada fracción en el estándar).
- Un exponente cero junto con una fracción cero representa el cero positivo o negativo, dependiendo del bit de signo. Como se mencionó, es útil tener una representación del valor 0 exacto.
- Un exponente todo unos junto con una parte fraccionaria cero representa, dependiendo del bit de signo, el infinito positivo o el negativo. Es también útil tener una representación de infinito. Esto deja al usuario decidir si trata el desbordamiento como error o si prosigue con él en el programa que se esté ejecutando.
- Un exponente cero junto con una parte fraccionaria distinta de cero representa un número denormalizado. En este caso, el bit a la izquierda de la coma binaria es cero y el exponente original es -126 o -1022. El número es positivo o negativo dependiendo del bit de signo.
- A un exponente de todo unos junto con una fracción distinta de cero se le da el nombre de NaN, (not a number, «no representa un número»), y se emplea para señalar varias condiciones de excepción.

El significado de los números denormalizados y de los NaN se discute en la Sección 8.5.

Tabla 8.4. Interpretación de los números en la notación de coma flotante IEEE 754

|                             | Precisión simple (32 bits) |                   |                    |                   | Doble precisión (64 bits) |                   |                    |                    |
|-----------------------------|----------------------------|-------------------|--------------------|-------------------|---------------------------|-------------------|--------------------|--------------------|
|                             | Signo                      | Exponente sesgado | Parte fraccionaria | Valor             | Signo                     | Exponente sesgado | Parte fraccionaria | Valor              |
| Cero positivo               | 0                          | 0                 | 0                  | 0                 | 0                         | 0                 | 0                  | 0                  |
| Cero negativo               | 1                          | 0                 | 0                  | -0                | 1                         | 0                 | 0                  | -0                 |
| Más infinito                | 0                          | 255 (todo unos)   | 0                  | $\infty$          | 0                         | 2047 (todo unos)  | 0                  | $\infty$           |
| Menos infinito              | 1                          | 255 (todo unos)   | 0                  | $-\infty$         | 1                         | 2047 (todo unos)  | 0                  | $-\infty$          |
| NaN silencioso              | 0 o 1                      | 255 (todo unos)   | $\neq 0$           | NaN               | 0 o 1                     | 2047 (todo unos)  | $\neq 0$           | NaN                |
| NaN indicador               | 0 o 1                      | 255 (todo unos)   | $\neq 0$           | NaN               | 0 o 1                     | 2047 (todo unos)  | $\neq 0$           | NaN                |
| Positivo normalizado ≠ cero | 0                          | $0 < e < 255$     | 1                  | $2^{e-127}(1.f)$  | 0                         | $0 < e < 2047$    | 1                  | $2^{e-1023}(1.f)$  |
| Negativo normalizado ≠ cero | 1                          | $0 < e < 255$     | 1                  | $-2^{e-127}(1.f)$ | 1                         | $0 < e < 2047$    | 1                  | $-2^{e-1023}(1.f)$ |
| Positivo denormalizado      | 0                          | 0                 | $f \neq 0$         | $2^{e-128}(0.f)$  | 0                         | 0                 | $f \neq 0$         | $2^{e-1024}(0.f)$  |
| Negativo denormalizado      | 1                          | 0                 | $f \neq 0$         | $-2^{e-128}(0.f)$ | 1                         | 0                 | $f \neq 0$         | $-2^{e-1024}(0.f)$ |

**8.5 ARITMÉTICA EN COMA FLOTANTE**

La Tabla 8.5 resume las operaciones básicas de la aritmética en coma flotante. En sumas y restas es necesario asegurar que ambos operandos tengan el mismo exponente. Esto puede requerir desplazar la coma de la base en uno de los operandos para conseguir alinearios. La multiplicación y la división son más directas.

Pueden surgir problemas como resultado de estas operaciones. Dichos problemas son:

Tabla 8.5. Números y operaciones aritméticas en coma flotante

| Números en punto flotante | Operaciones aritméticas                                                                                                     |
|---------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| $X = X_s \times B^{E_s}$  | $X + Y = (X_s \times B^{E_s} - Y_s \times B^{E_s}) \times B^{E_s}$<br>$X - Y = (X_s - B^{E_s} - Y_s) \times B^{E_s}$        |
| $Y = Y_s \times B^{E_s}$  | $X \times Y = (X_s \times Y_s) \times B^{E_s + E_s}$<br>$\frac{X}{Y} = \left( \frac{X_s}{Y_s} \right) \times B^{E_s - E_s}$ |

Ejemplos:

$$X = 0.3 \times 10^2 = 30$$

$$Y = 0.2 \times 10^2 = 200$$

$$X + Y = (0.3 \times 10^{2-2} + 0.21 \times 10^2 = 0.23 \times 10^2 = 230$$

$$X - Y = (0.3 \times 10^{2-2} - 0.21 \times 10^2 = (-0.17) \times 10^2 = -170$$

$$X \times Y = (0.3 \times 0.2) \times 10^{2-2} = 0.06 \times 10^0 = 6.000$$

$$X \div Y = (0.3 \div 0.2) \times 10^{2-2} = 1.5 \times 10^{-1} = 0.15$$

- **Desbordamiento del exponente:** un exponente positivo que excede el valor de exponente máximo posible. En algunos sistemas, este desbordamiento puede designarse como  $+\infty$  o  $-\infty$ .
- **Desbordamiento a cero del exponente:** un exponente negativo menor que el mínimo valor posible (ejemplo: -200 es menor que -127). Esto significa que el número es demasiado pequeño para ser representado, y que puede considerarse como 0.
- **Desbordamiento a cero de la mantisa:** en el proceso de alineación o ajuste de mantisas, pueden perderse dígitos por la parte derecha de la mantisa. Como comentaremos, se requiere efectuar algún tipo de redondeo.
- **Desbordamiento de la mantisa:** la suma de dos mantisas del mismo signo puede producir un acarreo procedente del bit más significativo. Esto, como se explicará, puede arreglarse con un reajuste (incremento del exponente y desplazamiento de la mantisa).

## SUMA Y RESTA

En la aritmética de coma flotante, la suma y la resta son más engorrosas que la multiplicación y la división. Esto es debido a la necesidad de ajustar las mantisas. Hay cuatro etapas básicas del algoritmo para sumar o restar:

1. Comprobar valores cero
2. Ajuste de mantisas
3. Sumar o restar las mantisas
4. Normalizar el resultado

Un diagrama de flujo típico se muestra en la Figura 8.22. En ella, una descripción paso a paso resalta las funciones principales requeridas para la suma y la resta en coma flotante. Se asume un formato similar al de la Figura 8.21. Para la operación de suma o de resta, los dos operandos deben transferirse a registros que serán utilizados por la ALU. Si el formato en coma flotante incluye un bit de mantisa implícito, dicho bit debe hacerse explícito para la operación.

Dado que la suma y la resta son idénticas, excepto por el cambio de signo, el proceso comienza cambiando el signo del substraendo cuando se trata de una resta. A continuación, si alguno de los operandos es 0, se da el otro como resultado.

La siguiente etapa manipula los números para que los dos exponentes sean iguales. Para ver la necesidad de este paso, considere la siguiente suma en decimal:

$$(123 \times 10^0) + (456 \times 10^{-2})$$

Claramente, no podemos sumar directamente sus mantisas. Los dígitos deben ponerse primero en posiciones equivalentes, es decir, el 4 del segundo número debe alinearse con el 3 del primero. Bajo estas condiciones, los exponentes serían iguales, que es la condición matemática para que dichos números se puedan sumar. Así:

$$(123 \times 10^0) + (456 \times 10^{-2}) = (123 \times 10^0) + (4.56 \times 10^0) = 127.56 \times 10^0$$

La alineación o ajuste se consigue, o bien desplazando a la derecha el número más pequeño (incrementando su exponente), o bien desplazando a la izquierda el más grande. Ya que cualquiera de dichas operaciones puede hacer que se pierdan dígitos, es el menor el que se desplaza, con lo que los dígitos que se pierden tienen una importancia relativa pequeña. El ajuste se

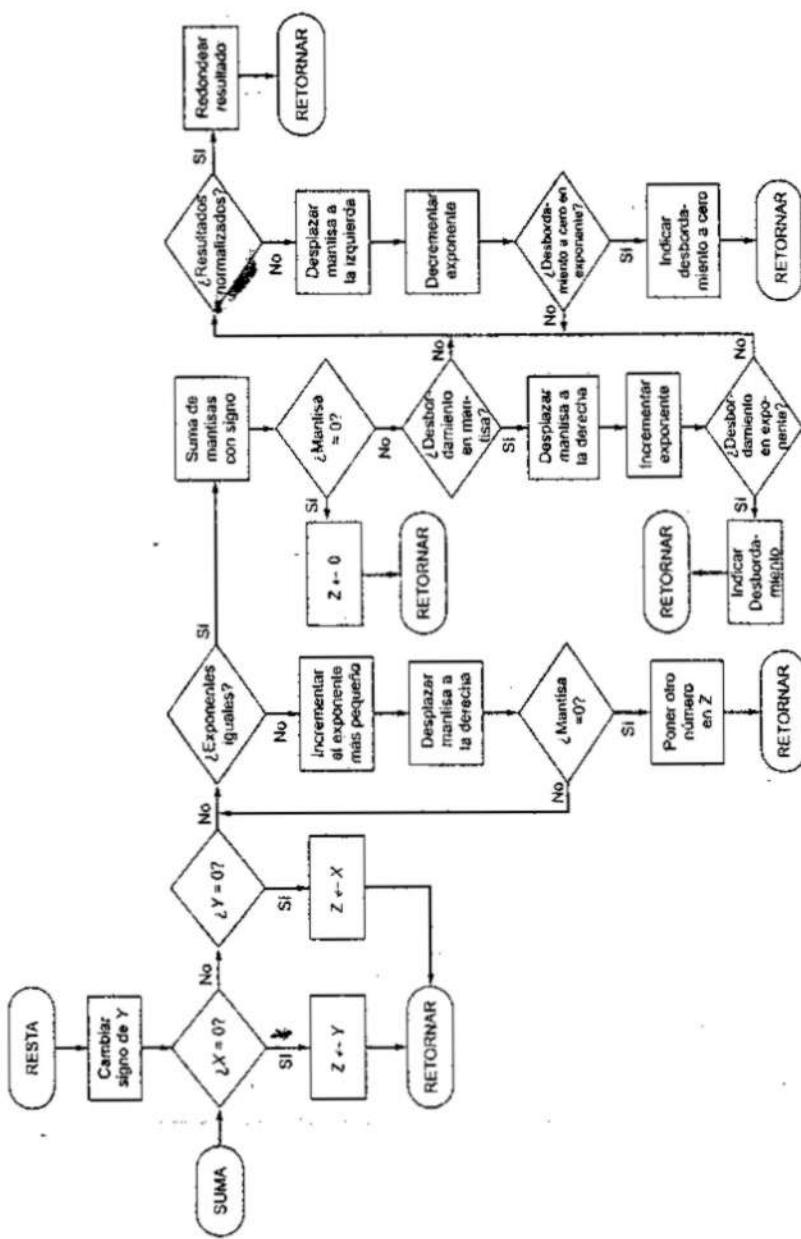


Figura 8.22. Suma y resta en coma flotante  $[Z - X \pm Y]$ .

consigue repitiendo desplazamientos de un dígito a la derecha de la mantisa, e incrementando el exponente hasta igualarlo con el otro (observe que, si la base implícita es 16, un desplazamiento de un dígito equivale a desplazar 4 bits). Si en el proceso se obtiene una mantisa 0 se da como resultado el otro número. Así, cuando dos números tienen exponentes muy diferentes, se pierde el menor de los números.

A continuación se suman las dos mantisas, teniendo en cuenta sus signos. Ya que los signos pueden diferir, el resultado puede ser 0. Existe también la posibilidad de desbordamiento de la mantisa en un dígito. Si es así, se desplaza a la derecha la mantisa del resultado, y se incrementa su exponente. Como resultado podría producirse un desbordamiento en el exponente; esto se debe indicar y la operación se detendría.

La siguiente fase normaliza el resultado. La normalización consiste en desplazar a la izquierda los dígitos de la mantisa, hasta que el más significativo (1 bit o 4 bits para exponentes de base 16) sea distinto de cero. Cada desplazamiento causa un decremento del exponente, lo que podría producir un desbordamiento a cero del mismo. Finalmente, el resultado

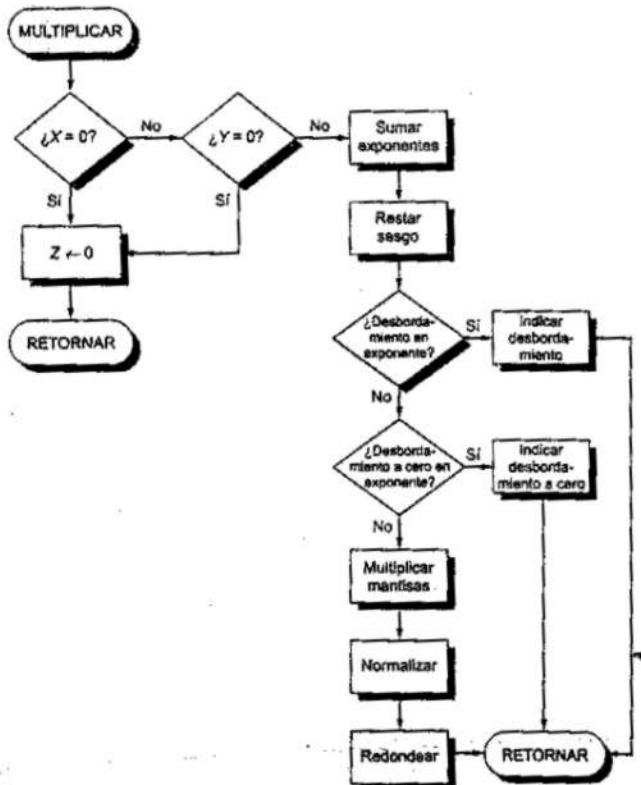


Figura 8.23. Multiplicación en coma flotante ( $Z \leftarrow X \times Y$ ).

debe redondearse y proporcionarse. Posponemos el tema del redondeo para después del estudio de la multiplicación y la división.

## MULTIPLICACIÓN Y DIVISIÓN

La multiplicación y la división en coma flotante son procesos mucho más sencillos que la suma y la resta, como se indica en la discusión que sigue.

Consideremos primero la multiplicación, que se ilustra en la Figura 8.23. En primer lugar, si cualquiera de los operandos es 0, se indica como resultado 0. El siguiente paso es sumar los exponentes. Si los exponentes están almacenados en forma sesgada, su suma tendría un sesgo doble. Por ello, debe restarse de la suma el valor de sesgo. El resultado podría dar lugar a un desbordamiento o a un desbordamiento a cero del exponente, lo que debe indicarse, y concluir el algoritmo.

Si el exponente del producto está dentro del rango apropiado, el paso siguiente es multiplicar las mantisas teniendo en cuenta sus signos. Dicha multiplicación se realiza como en el caso de los enteros. En este caso, estamos tratando con una representación en signo-magnitud, pero los detalles son similares a los de complemento a dos. El producto doblará la longitud del multiplicando y del multiplicador. Los bits extra se perderán durante el redondeo.

Tras calcular el producto, se normaliza y redondea el resultado, como se hizo para la suma y la resta. Observe que la normalización podría producir un desbordamiento a cero del exponente.

Consideremos finalmente el diagrama de flujo para la división, mostrado en la Figura 8.24. De nuevo, el primer paso es comprobar ceros. Si el divisor es 0, se da una indicación de error, o se pone el resultado a infinito, dependiendo de la implementación en cuestión. Un dividendo 0 da como resultado 0. A continuación, el exponente del divisor se resta al del dividendo. Esto elimina el sesgo, que debe añadirse de nuevo. Se comprueban entonces posibles desbordamientos (a cero o no) del exponente.

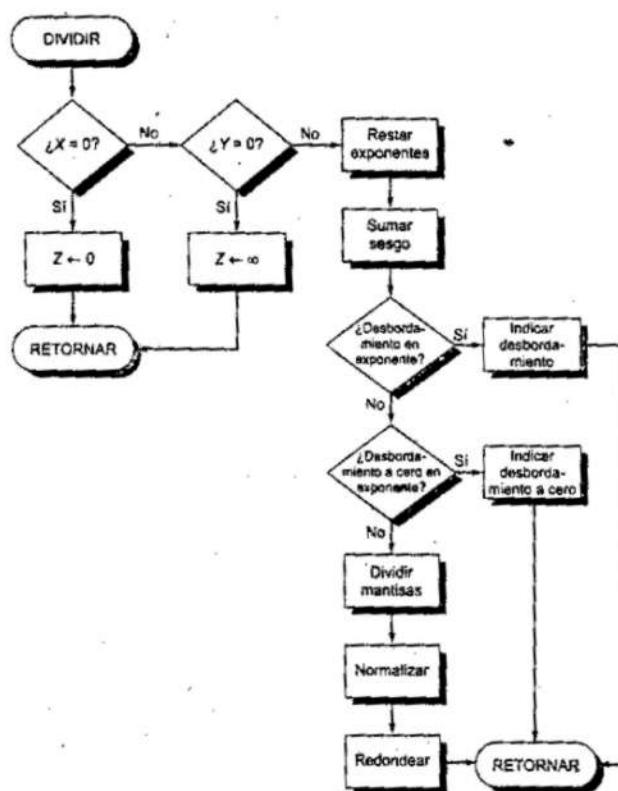
El siguiente paso es dividir las mantisas. A este paso sigue la normalización y redondeo usuales.

## CONSIDERACIONES SOBRE PRECISIÓN

### Bits de guarda

Hemos mencionado que, como paso previo a una operación en coma flotante, se cargan el exponente y la mantisa en registros de la ALU. En el caso de la mantisa, el tamaño del registro es casi siempre mayor que la longitud de la mantisa más el bit implícito (si se usa éste). El registro contiene bits adicionales, llamados bits de guarda o de respaldo, que se añaden a la derecha de la mantisa en forma de ceros.

La razón de utilizar estos bits se ilustra en la Figura 8.25. Considere números en el formato IEEE, que tiene una mantisa de 24 bits, incluyendo un 1 implícito a la izquierda de la coma binaria. Dos números de valor muy próximo son  $X = 1.00\ldots 00 \times 2^1$  e  $Y = 1.11\ldots 11 \times 2^0$ . Para restar el menor de ellos del mayor, debe desplazarse aquél un bit a la derecha para igualar los exponentes. Esto se muestra en la Figura 8.25a. En este proceso Y pierde 1 bit significativo; el resultado que se obtiene es  $2^{-22}$ . En la parte b de la figura se repite la misma operación, pero con bits de guarda añadidos. Ahora, el bit menos significativo no se pierde al alinear, y el resultado es  $2^{-23}$ , diferente en un factor de 2 respecto del anterior resultado.

Figura 8.24. División en coma flotante ( $Z \leftarrow X/Y$ ).

### Redondeo

Otro detalle que afecta a la precisión del resultado es la política de redondeo empleada. El resultado de cualquier operación sobre las mantisbras se almacena generalmente en un registro más grande. Cuando el resultado se pone de nuevo en el formato de coma flotante, hay que deshacerse de los bits extra.

Se han explorado diversas técnicas para realizar el redondeo. De hecho, el estándar del IEEE enumera cuatro aproximaciones alternativas:

- Redondeo al más próximo: el resultado se redondea al número más próximo representable.
- Redondeo hacia  $+\infty$ : el resultado se redondea por exceso hacia más infinito.
- Redondeo hacia  $-\infty$ : el resultado se redondea por defecto hacia menos infinito.
- Redondeo hacia 0: el resultado se redondea hacia cero.

$$\begin{array}{r} x = 1,000 \dots 00 \times 2^1 \\ -y = 0,111 \dots 11 \times 2^1 \\ \hline z = 0,000 \dots 01 \times 2^1 \\ = 1,000 \dots 00 \times 2^{-22} \end{array}$$

(a) Ejemplo binario, sin bits de guarda

$$\begin{array}{r} x = 1,000 \dots 00 \quad 0000 \times 2^1 \\ -y = 0,111 \dots 11 \quad 1000 \times 2^1 \\ \hline z = 0,000 \dots 00 \quad 1000 \times 2^1 \\ = 1,000 \dots 00 \quad 0000 \times 2^{-23} \end{array}$$

(b) Ejemplo binario, con bits de guarda

$$\begin{array}{r} x = 0,100000 \times 16^1 \\ -y = 0,0FFFFF \times 16^1 \\ \hline z = 0,000001 \times 16^1 \\ = 0,100000 \times 16^{-4} \end{array}$$

(c) Ejemplo hexadecimal, sin bits de guarda

$$\begin{array}{r} x = 0,100000 \quad 00 \times 16^1 \\ -y = 0,0FFFFF \quad F0 \times 16^1 \\ \hline z = 0,000000 \quad 10 \times 16^1 \\ = 0,100000 \quad 00 \times 16^{-5} \end{array}$$

(d) Ejemplo hexadecimal, con bits de guarda

Figura 8.25. Utilización de bits de guarda.

Consideremos cada uno de los casos anteriores. El redondeo al más próximo es el modo implícito contemplado en el estándar, y se define como sigue: debe tomarse el valor representable más próximo al resultado exacto; si los dos valores representables están igualmente próximos se daría aquél que produjese un 0 como bit menos significativo.

Por ejemplo, si los bits extra, aparte de los 23 bits que pueden almacenarse, son 10010, entonces la cantidad indicada por los bits extra supera la mitad del valor correspondiente a la última posición de bit representable. En este caso, la respuesta correcta es sumar 1 al último bit representable, redondeando por exceso al número siguiente representable. Considere ahora que los bits extra valen 01111. En este caso, representan una cantidad menor que la mitad de la última posición de bit representable. La respuesta correcta es, simplemente, ignorar los bits extra (truncar), lo que tiene como efecto un redondeo por defecto al número representable precedente.

El estándar también contempla el caso especial de bits extra en la forma 10000..., en que el resultado está exactamente en la mitad de los dos valores representables posibles. Una posible solución sería truncar siempre, que es la operación más sencilla. Sin embargo, la dificultad de esta aproximación simple es que introduce un sesgo pequeño, pero acumulativo, en una secuencia de cálculos. Se necesita un método de redondeo que no introduzca esta tendencia de los resultados. Una posibilidad sería redondear por exceso o por defecto basándose en un número aleatorio, de manera que, en promedio, el resultado no sería sesgado. Un argumento en contra de esta aproximación es que no produciría resultados predecibles deterministas. La aproximación tomada en la norma del IEEE es forzar que el resultado sea par: si el

resultado de un cálculo está exactamente en la mitad de dos números representables, el valor se redondea por exceso cuando el último bit representable actual es 1, y se deja como está si es 0.

Las dos siguientes opciones, redondeo a más o menos infinito, son útiles en la implementación de una técnica conocida como «aritmética de intervalos».\* La aritmética de intervalos proporciona un método eficiente para monitorizar y controlar errores en los cálculos en coma flotante, produciendo dos valores por cada resultado. Los dos valores se corresponden con los límites inferior y superior de un intervalo, que contiene el resultado exacto. La longitud del intervalo, que es la diferencia entre los límites superior e inferior, indica la precisión del resultado. Si los extremos del intervalo no son representables, se redondean por defecto y por exceso, respectivamente. Aunque la anchura del intervalo puede variar de unas implementaciones a otras, se han diseñado diversos algoritmos al objeto de conseguir intervalos estrechos. Si el rango de variación entre los límites superior e inferior es suficientemente estrecho se obtiene un resultado bastante preciso. Si no, al menos lo sabemos y podemos realizar análisis adicionales.

La última de las técnicas especificadas en el estándar es el redondeo hacia cero. Consiste, de hecho, en un simple truncamiento: se ignoran los bits extra. Esta es, ciertamente, la técnica más sencilla. Sin embargo, el resultado es que la magnitud del valor truncado es siempre menor o igual que el valor original más exacto, introduciéndose un sesgo hacia cero en la operación. Este sesgo o tendencia es más serio que el discutido antes, ya que afecta a todas las operaciones para las que haya algún bit extra distinto de cero.

## ESTÁNDAR DEL IEEE PARA LA ARITMÉTICA BINARIA EN COMA FLOTANTE

El IEEE 754 va más allá de la simple definición de un formato, detallando cuestiones prácticas específicas y procedimientos para que la aritmética en coma flotante produzca resultados uniformes y predecibles, independientemente de la plataforma hardware. Uno de estos aspectos ha sido ya discutido: el redondeo. En esta subsección se ven otros tres aspectos: el infinito, los NaN, y los números sin normalizar o «denormalizados».

### Infinito

Las operaciones aritméticas con infinito son tratadas como casos límite de la aritmética real, dándose la siguiente interpretación a los valores de infinito:

$$-\infty < (\text{todo número finito}) < +\infty$$

Exceptuando los casos especiales, discutidos posteriormente, cualquier operación aritmética que involucre al infinito produce el resultado obvio conocido. Por ejemplo:

|                                |                                   |
|--------------------------------|-----------------------------------|
| $5 + (+\infty) = +\infty$      | $5 + (+\infty) = +\infty$         |
| $5 - (+\infty) = -\infty$      | $(+\infty) + (+\infty) = +\infty$ |
| $5 + (-\infty) = -\infty$      | $(-\infty) + (-\infty) = -\infty$ |
| $5 - (-\infty) = +\infty$      | $(-\infty) - (+\infty) = -\infty$ |
| $5 \times (+\infty) = +\infty$ | $(+\infty) - (-\infty) = +\infty$ |

### NaN indicadores y silenciosos

Un NaN es una entidad simbólica codificada en formato de coma flotante, de la que hay dos tipos: indicadores y silenciosos. Un NaN indicador señala una condición de operación no válida siempre que aparece como operando. Los NaN indicadores permiten representar valores de variables no inicializadas y tratamientos de tipo aritmético, que no están contemplados en el estándar. Un NaN silencioso se propaga en la mayoría de las operaciones sin señalar excepción alguna. La Tabla 8.6 indica operaciones que producirían un NaN silencioso.

Obsérvese que ambos tipos de NaN tienen el mismo formato general: un exponente con todo unos y una parte fraccionaria distinta de cero. El patrón de bits real de dicha fracción depende de cada implementación concreta; sus valores pueden utilizarse para distinguir entre NaN indicadores y silenciosos, y para especificar condiciones de excepción particulares.

### Números denormalizados

Los números denormalizados se incluyen en el IEEE 754 para reducir las situaciones de desbordamiento hacia cero de exponentes. Cuando el exponente del resultado es demasiado pequeño (un exponente negativo con magnitud muy grande), el resultado se denormaliza desplazando a la derecha la parte fraccionaria e incrementando el exponente, a cada desplazamiento, hasta que dicho exponente esté dentro de un rango representable.

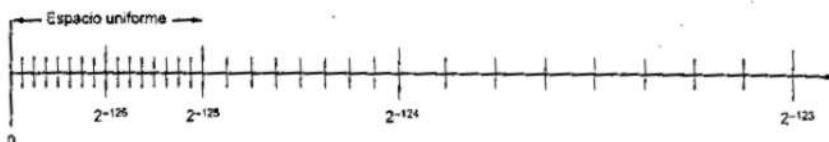
La Figura 8.26 ilustra el efecto que tiene añadir los números denormalizados. Los números que son representables pueden agruparse en intervalos de la forma  $[2^n, 2^{n+1}]$ . Dentro de cada intervalo, la parte de exponente del número es la misma y varía la parte fraccionaria, produciéndose un espacio uniforme de los números representables en el intervalo. A medida que nos aproximamos a cero, cada intervalo sucesivo es la mitad de ancho que el precedente, pero contiene la misma cantidad de números representables. Por tanto, la densidad de números representables aumenta a medida que nos aproximamos a cero. Sin embargo, si sólo se emplean números normalizados, hay una zona inutilizada entre cero y el menor de los números normalizados. En el caso del IEEE 754 hay  $2^{23}$  números representables en cada inter-

Tabla 8.6. Operaciones que producen un NaN silencioso

| Operación      | NaN silencioso producido por                                                                                                     |
|----------------|----------------------------------------------------------------------------------------------------------------------------------|
| Cualquiera     | Cualquier operación con un NaN indicador                                                                                         |
| Suma o resta   | Restas con infinito:<br>$(+\infty) + (-\infty)$<br>$(-\infty) + (+\infty)$<br>$(+\infty) - (+\infty)$<br>$(-\infty) - (-\infty)$ |
| Multiplicación | $0 \times \infty$                                                                                                                |
| División       | $\frac{0}{0} = \infty$                                                                                                           |
| Resto          | $x \text{ RE } 0 \quad 0 \text{ RE } y$                                                                                          |
| Raíz cuadrada  | $\sqrt{x}$ donde $x < 0$                                                                                                         |



(a) Formato de 32 bits sin números denormalizados



(b) Formato de 32 bits con números denormalizados

Figura 8.26. Efecto de los números denormalizados del IEEE 754.

valo, y el número positivo más pequeño representable es  $2^{-126}$ . Al considerar también los números denormalizados, se añaden  $2^{23}$  números uniformemente distribuidos entre 0 y  $2^{-126}$ .

El uso de números denormalizados se denomina *desbordamiento hacia cero gradual* [COON81]. Sin números denormalizados, la zona entre cero y el número distinto de cero más pequeño representable es mucho más grande que la zona entre dicho número y el que le sigue. El desbordamiento a cero gradual cubre esta zona y reduce el efecto de desbordamiento a cero del exponente hasta un nivel comparable con el redondeo de números normalizados.

## 8.6. LECTURAS Y SITIOS WEB RECOMENDADOS

Para un estudio serio de la aritmética del computador, es imprescindible el trabajo en dos volúmenes [SWAR90]. El Volumen I fue inicialmente publicado en 1980 y contiene artículos clave publicados (algunos de ellos difíciles de encontrar por otras vías) sobre fundamentos de aritmética del computador. El Volumen II contiene artículos más recientes que tratan sobre aspectos teóricos, de diseño, y de implementación. [OMON94], [KORE93], y [MORG92] son buenos ejemplos de libros sobre aritmética del computador.

Sobre aritmética en coma flotante, el libro [GOLD91] tiene un título apropiado: «What Every Computer Scientist Should Know About Floating-Point Arithmetic» (Lo que todo informático debe conocer sobre aritmética en coma flotante). Un tratamiento más avanzado puede encontrarse en [WALL90]. Otro excelente tratado sobre el tema se incluye en [KNUT81], que cubre también la aritmética con enteros. También merecen la pena los siguientes tratados de mayor profundidad: [OBER97a, OBER97b, SODE96].

**GOLD91** Goldberg, D. «What Every Computer Scientist Should Know About Floating-Point Arithmetic.» *ACM Computing Surveys*, March 1991. Available at <http://www.validgh.com/>

**KNUT98** Knuth, D. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Reading, MA: Addison-Wesley, 1998.

- KORE93 Koren, I. *Computer Arithmetic Algorithms*. Englewood Cliffs, NJ: Prentice Hall, 1993.
- MORG92 Morgan, D. *Numerical Methods*. San Mateo, CA: M&T Books, 1992.
- OBER97a Oberman, S., y Flynn, M. «Design Issues in Division and Other Floating-Point Operations.» *IEEE Transactions on Computers*, February, 1997.
- OBER97b Oberman, S., y Flynn, M. «Division Algorithms and Implementations.» *IEEE Transactions on Computers*, August, 1997.
- OMON94 Omondi, A. *Computer Arithmetic Systems: Algorithms, Architecture and Implementations*. Englewood Cliffs, NJ: Prentice Hall, 1994.
- SODE96 Soderquist, P., y Leeser, M. «Area and Performance Tradeoffs in Floating-Point Divide and Square-Root Implementations.» *ACM Computing Surveys*, September, 1996.
- SWAR90 Swartzlander, E., editor. *Computer Arithmetic, Volumes I and II*. Los Alamitos, CA: IEEE Computer Society Press, 1990.
- WALL90 Wallis, P. *Improving Floating-Point Programming*. New York: Wiley, 1992.



#### SITIO WEB RECOMENDADO:

- IEEE 754: Incluye los documentos del IEEE 754, publicaciones y artículos relacionados, y múltiples enlaces útiles relacionados con la aritmética del computador.

#### 8.7.3 PROBLEMAS

- 8.1. Otra representación utilizada a veces para los números enteros es el complemento a 1. Los enteros positivos se representan de la misma forma que en signo-magnitud. Un entero negativo se representa tomando el complemento booleano de cada bit del correspondiente número positivo.
  - Expresa una definición de los números en complemento a uno utilizando una suma ponderada de bits, similar a las Ecuaciones (8.1) y (8.2).
  - ¿Cuál es el rango de números que puede representarse en complemento a uno?
  - Defina un algoritmo que efectúe la suma en aritmética de complemento a uno.
- 8.2. Añada columnas a la Tabla 8.1 para signo-magnitud y para complemento a uno.
- 8.3. Considere la siguiente operación con una palabra binaria. Comenzar con el bit menos significativo. Copiar todos los bits que son 0 hasta que se encuentra el primer 1, que también se copia. A partir de éste, tomar el complemento booleano de los bits siguientes. ¿Cuál es el resultado?
- 8.4. En la Sección 8.3 se define la operación de complemento a dos como sigue: para calcular el complemento a dos de  $X$ , tomar el complemento booleano de cada bit de  $X$ , y después sumar 1.

- a) Compruebe que la siguiente definición es equivalente. Para un entero  $X$  de  $n$  bits, el complemento a dos de  $X$  se obtiene considerando  $X$  como entero sin signo y calculando  $(2^n - X)$ .
- b) Demuestre que la Figura 8.2 puede utilizarse para ilustrar gráficamente el punto anterior, mostrando cómo se usa el desplazamiento en el sentido de las agujas del reloj para realizar la substracción.
- 8.5. Calcule las siguientes diferencias utilizando complemento a dos:
- |           |             |                 |             |
|-----------|-------------|-----------------|-------------|
| a) 111000 | b) 11001100 | c) 111100001111 | d) 11000011 |
| -110011   | -101110     | -110011110011   | -11101000   |
- 8.6. ¿Es válida la siguiente definición alternativa de desbordamiento en aritmética de complemento a dos?
- Si la OR exclusiva de los bits de acarreo anterior y posterior a la columna más a la izquierda es 1, hay desbordamiento. En caso contrario no hay desbordamiento.
- 8.7. Compare las Figuras 8.9 y 8.12. ¿Por qué no se utiliza el bit C en la segunda?
- 8.8. Dados  $x = 0101$  e  $y = 1010$  en notación de complemento a dos (es decir,  $x = 5$  e  $y = -6$ ), calcule el producto  $p = x \times y$  con el algoritmo de Booth.
- 8.9. Demuestre que el producto de dos números de  $n$  dígitos en base B produce un resultado de no más de  $2n$  dígitos.
- 8.10. Verifique la validez del algoritmo de división de binarios sin signo de la Figura 8.16, mostrando los pasos implicados en el cálculo de la división de la Figura 8.15. Utilice una presentación similar a la empleada en la Figura 8.17.
- 8.11. El algoritmo de división entera descrito en la Sección 8.3 se conoce con el nombre de *método de división con restablecimiento*, ya que el valor del registro A debe restablecerse tras cada resta sin éxito. Una aproximación ligeramente más compleja, denominada *sin restablecimiento*, evita las restas y sumas innecesarias. Proponga un algoritmo para este método.
- 8.12. En operaciones aritméticas con enteros, el cociente  $J/K$  de dos enteros  $J$  y  $K$  es menor o igual que el cociente normal. ¿Verdadero o falso?
- 8.13. Divida  $-145$  entre  $13$  en notación binaria de complemento a dos, utilizando palabras de  $12$  bits. Utilice el algoritmo descrito en la Sección 8.3.
- 8.14. Suponga que el exponente  $e$  está restringido al rango  $0 \leq e \leq X$ , con un sesgo  $g$ , que la base es  $b$ , y que el formato tiene una longitud de  $p$  dígitos.
- ¿Cuáles son los números positivos mayor y menor que pueden expresarse?
  - ¿Cuáles son los números positivos mayor y menor que pueden expresarse como números normalizados en coma flotante?
- 8.15. Exprese, en formato de coma flotante IEEE de 32 bits los siguientes números:
- 5
  - 6
  - 1,5
  - 384
  - 1/16
  - 1/32

- 8.16. Exprese los siguientes números en formato de coma flotante IBM de 32 bits, que utiliza un exponente de 7 bits con 16 de base implícita:

|        |         |                          |                         |
|--------|---------|--------------------------|-------------------------|
| a) 1,0 | c) 1/64 | e) -15,0                 | g) $7,2 \times 10^{-5}$ |
| b) 0,5 | d) 0,0  | f) $5,4 \times 10^{-79}$ |                         |

- 8.17. ¿Cuál sería el valor de sesgo para:
- Un exponente de base 2 ( $B = 2$ ) en un campo de 6 bits?
  - Un exponente de base 8 ( $B = 8$ ) en un campo de 7 bits?
- 8.18. Dibuje una representación de la recta real, similar a la Figura 8.19b, para el formato de coma flotante de la Figura 8.21b.
- 8.19. Considere un formato de coma flotante con 8 bits para el exponente sesgado y 23 bits para la mantisa. Obtenga los patrones de bits de los siguientes números expresados con dicho formato:
- 720
  - 0,645
- 8.20. Cuando la gente habla de la imprecisión de la aritmética en coma flotante, normalmente asocia los errores a la cancelación que tiene lugar al restar cantidades muy próximas entre sí. Pero cuando  $X$  e  $Y$  son aproximadamente iguales, la diferencia  $X - Y$  se obtiene con exactitud, sin error. ¿Qué es lo que quiere decir exactamente la gente?
- 8.21. Cualquier representación en coma flotante utilizada en computadores representa con exactitud sólo ciertos números; todos los demás deben aproximarse. Si  $A'$  es el valor almacenado del valor real  $A$ , el error relativo,  $r$ , se expresa como:
- $$r = \frac{A - A'}{A}$$
- Represente la cantidad decimal +0,4 en el siguiente formato de coma flotante: base: 2; exponente: sesgado, 4 bits; mantisa: 7 bits. ¿Cuál es el error relativo?
- 8.22. Los valores numéricos  $A$  y  $B$  se almacenan en un computador como los aproximados  $A'$  y  $B'$ . Despreciando cualesquiera errores de truncamiento o redondeo posteriores, pruebe que el error relativo del producto es, aproximadamente, la suma de los errores relativos de los factores.
- 8.23. Siendo  $A = 1,427$ , encuentre el error relativo cuando  $A$  se trunca a 1,42 y cuando se redondea a 1,43.
- 8.24. Uno de los errores más serios en los cálculos con computadores se produce al restar dos números casi iguales. Considere  $A = 0,22288$  y  $B = 0,22211$ . El computador trunca todos los valores a 4 dígitos decimales. Por tanto  $A' = 0,2228$  y  $B' = 0,2221$ .
- ¿Cuáles son los errores relativos de  $A'$  y  $B'$ ?
  - ¿Cuál es el error relativo de  $C' = A' - B'$ ?

- 8.25. Muestre cómo se realizan las siguientes sumas en coma flotante (en las que las mantis se truncan a 4 dígitos decimales):
- $0.5566 \times 10^3 + 0.7777 \times 10^3$
  - $0.3344 \times 10^2 + 0.8877 \times 10^{-1}$
- 8.26. Muestre cómo se realizan las siguientes restas en coma flotante (en las que las mantis se truncan a 4 dígitos decimales):
- $0.7744 \times 10^{-2} - 0.6666 \times 10^{-2}$
  - $0.8844 \times 10^{-2} - 0.2233 \times 10^0$
- 8.27. Muestre cómo se realizan los siguientes cálculos en coma flotante (en los que las mantis se truncan a 4 dígitos decimales):
- $(0.2255 \times 10^3) \times (0.1234 \times 10^1)$
  - $(0.8833 \times 10^3) \div (0.5555 \times 10^5)$
- 8.28. Convierta los siguientes números expresados en octal a notación hexadecimal:
- 12
  - 5655
  - 2550276
  - 76545336
  - 3726755
- 8.29. Demuestre que todo número real con una representación binaria restringida (número finito de dígitos a la derecha de la coma binaria) también tiene una representación decimal restringida (número finito de dígitos a la derecha de la coma decimal).

## ÍNDICE 8. SISTEMAS DE NUMERACIÓN

### SISTEMA DECIMAL

A diario, utilizamos un sistema basado en los dígitos decimales (0, 1, 2, 3, 4, 5, 6, 7, 8, 9) para representar los números (el denominado sistema decimal). Considere lo que significa el número 83. Significa ocho veces diez, más tres:

$$83 = (8 \times 10) + 3$$

El número 4.728 significa cuatro veces mil, siete veces cien, dos veces diez, más ocho:

$$4.728 = (4 \times 1.000) + (7 \times 100) + (2 \times 10) + 8$$

El sistema decimal se dice que usa la *base 10*. Esto significa que cada dígito del número se multiplica por diez elevado a la potencia correspondiente a la posición de dicho dígito. Así:

$$83 = (8 \times 10^1) + (3 \times 10^0)$$

$$4.728 = (4 \times 10^3) + (7 \times 10^2) + (2 \times 10^1) + (8 \times 10^0)$$

Los valores fraccionarios se representan de la misma manera. Así:

$$472.83 = (4 \times 10^2) + (7 \times 10^1) + (2 \times 10^0) + (8 \times 10^{-1}) + (3 \times 10^{-2})$$

En general, para la representación decimal de  $X = \dots x_1x_1x_0\dots x_{-1}x_{-2}x_{-3}\dots$ , el valor de  $X$  es:

$$X = \sum_i x_i 10^i$$

## SISTEMA BINARIO

En el sistema decimal se emplean 10 dígitos diferentes para representar números en base diez. En el sistema binario se tienen sólo los dígitos 1 y 0. Por tanto, los números en el sistema binario se representan en base dos.

Para evitar confusiones, pondremos a veces un subíndice en los números, que indica su base. Por ejemplo,  $83_{10}$  y  $4.728_{10}$  son números representados en notación decimal, o simplemente números decimales. El 1 y el 0 en notación binaria tienen el mismo significado que en notación decimal:

$$0_2 = 0_{10}$$

$$1_2 = 1_{10}$$

Para representar números mayores, como ocurre en la notación decimal, cada dígito de un número binario tiene un valor que depende de su posición:

$$10_2 = (1 \times 2^1) + (0 \times 2^0) = 2_{10}$$

$$11_2 = (1 \times 2^1) + (1 \times 2^0) = 3_{10}$$

$$100_2 = (1 \times 2^2) + (0 \times 2^1) + (0 \times 2^0) = 4_{10}$$

Y así sucesivamente. De forma similar, los valores fraccionarios se representan con potencias negativas de la base:

$$1001.101_2 = 2^3 + 2^0 + 2^{-1} + 2^{-3} = 9.625_{10}$$

## CONVERSIÓN ENTRE BINARIO Y DECIMAL

Es fácil convertir un número de notación binaria a decimal. En la subsección anterior hemos mostrado diversos ejemplos. Todo lo que se necesita es multiplicar cada dígito binario por la potencia de 2 apropiada, y sumar los resultados.

Para convertir de decimal a binario, las partes fraccionarias se tratan por separado. Suponga que se quiere convertir un entero decimal  $N$  a forma binaria. Si dividimos  $N$  entre 2 en el sistema decimal, y obtenemos un cociente  $N_1$  y un resto  $R_1$ , podemos escribir:

$$N = 2 \times N_1 + R_1 \quad R_1 = 0 \text{ o } 1$$

A continuación, dividimos el cociente  $N_1$  por 2. Suponga que el nuevo cociente es  $N_2$ , y el nuevo resto es  $R_2$ . Entonces:

$$N_1 = 2 \times N_2 + R_2 \quad R_2 = 0 \text{ o } 1$$

De manera que:



$$N = 2(2N_2 + R_2) + R_1 = 2^2N_2 + R_2 \times 2^1 + R_1 \times 2^0$$

Si sustituimos:

$$N_2 = 2N_3 + R_3$$

tenemos:

$$N = 2^3N_3 + R_3 \times 2^2 + R_2 \times 2^1 + R_1 \times 2^0$$

Continuando este proceso, ya que  $N > N_1 > N_2 \dots$  llegará a producirse un cociente  $N_k = 1$  (excepto para los enteros decimales 0 y 1, cuyos equivalentes binarios son 0 y 1 respectivamente), y un resto  $R_k$  que es 0 o 1. Entonces:

$$N = (1 \times 2^k) + (R_k \times 2^{k-1}) + \dots + (R_3 \times 2^3) + (R_2 \times 2^2) + (R_1 \times 2^1)$$

Es decir, convertimos de base 10 a base 2 mediante divisiones repetidas por 2. Los restos y el cociente final, 1, nos dan los dígitos binarios de  $N$  en orden de significancia creciente. La Figura 8.27 muestra un par de ejemplos.

La parte fraccionaria implica repetidas multiplicaciones por 2, como se ilustra en la Figura 8.28. En cada paso se multiplica por 2 la parte fraccionaria del número decimal. El dígito del producto a la izquierda de la coma decimal será 1 o 0 y contribuye a la representación binaria, comenzando por el bit más significativo. La parte fraccionaria del producto se utiliza como multiplicando en el siguiente paso. Para ver que esto es correcto, tomemos una fracción decimal positiva  $F < 1$ . Podemos expresar  $F$  como

$$F = \left( a_{-1} \times \frac{1}{2} \right) + \left( a_{-2} \times \frac{1}{2^2} \right) + \left( a_{-3} \times \frac{1}{2^3} \right) + \dots$$

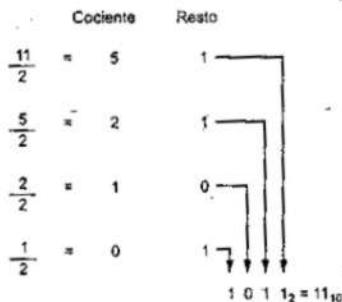
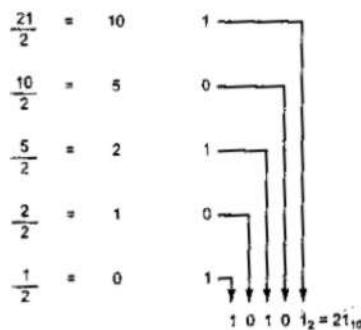
(a)  $11_{10}$ (b)  $21_{10}$ 

Figura 8.27. Ejemplos de conversión de números enteros de la notación decimal a la binaria.

| Producto               | Parte entera | .1 1 0 0 1 1 |
|------------------------|--------------|--------------|
| $0.81 \times 2 = 1.62$ | 1            | ↑            |
| $0.62 \times 2 = 1.24$ | 1            | ↑            |
| $0.24 \times 2 = 0.48$ | 0            | ↑            |
| $0.48 \times 2 = 0.96$ | 0            | ↑            |
| $0.96 \times 2 = 1.92$ | 1            | ↑            |
| $0.92 \times 2 = 1.84$ | 1            | ↑            |

(a)  $0.81_{10} \approx 0.110011_2$  (aproximado)(b)  $0.25_{10} = 0.01_2$  (exacto)

Figura 8.28. Ejemplos de conversión de números fraccionarios de la notación decimal a la binaria.

donde cada  $a_{-i}$  es 0 o 1. Si multiplicamos por 2, se obtiene:

$$2F = a_{-1} + \left( a_{-2} \times \frac{1}{2} \right) + \left( a_{-3} \times \frac{1}{2^2} \right) + \left( a_{-4} \times \frac{1}{2^3} \right) + \dots$$

Las partes enteras de estas dos expresiones deben ser iguales. Por tanto la parte entera de  $2F$ , que debe ser 0 o 1 puesto que  $0 < F < 1$ , es simplemente  $a_{-1}$ . Así,  $2F = a_{-1} + F_1$ , donde  $0 < F_1 < 1$ , y donde:

$$F_1 = \left( a_{-2} \times \frac{1}{2} \right) + \left( a_{-3} \times \frac{1}{2^2} \right) + \left( a_{-4} \times \frac{1}{2^3} \right) + \dots$$

Para encontrar  $a_{-2}$  se repite el proceso. Este proceso no es necesariamente exacto; es decir, una parte fraccionaria decimal con un número finito de dígitos puede requerir una parte fraccionaria binaria con infinitos dígitos. En tales casos, el algoritmo de conversión es normalmente interrumpido después de un número de pasos preestablecido, que depende de la precisión deseada.

## NOTACIÓN HEXADECIMAL

Dada la naturaleza binaria de los componentes de los computadores digitales, todos los tipos de datos son representados en los computadores mediante diversos códigos binarios. Hemos visto ejemplos del uso de códigos binarios para texto y la notación binaria para enteros. Posteriormente veremos ejemplos de utilización de códigos binarios para otros tipos de datos. Sin embargo, aunque el sistema binario sea muy conveniente para los computadores, resulta bastante engorroso para los humanos. En consecuencia, la mayoría de los profesionales de la Informática, que tienen que trabajar a menudo con los datos «en bruto» del computador, prefieren una notación más compacta.

¿Qué notación utilizar? Una posibilidad es la decimal. Esta es, ciertamente, más compacta que la notación binaria, pero es engorroso por lo tedioso de convertir de base 2 a base 10, y viceversa.

En su lugar se ha optado por una notación conocida como hexadecimal. Los dígitos binarios son agrupados en conjuntos de 4. A cada combinación posible de 4 dígitos binarios se le asocia un símbolo de la siguiente manera:

|          |          |
|----------|----------|
| 0000 = 0 | 1000 = 8 |
| 0001 = 1 | 1001 = 9 |
| 0010 = 2 | 1010 = A |
| 0011 = 3 | 1011 = B |
| 0100 = 4 | 1100 = C |
| 0101 = 5 | 1101 = D |
| 0110 = 6 | 1110 = E |
| 0111 = 7 | 1111 = F |

La notación se denomina hexadecimal por utilizar 16 símbolos, y a cada uno de ellos se le llama dígito hexadecimal.

Una secuencia de dígitos hexadecimales puede considerarse como un entero representado en base 16. Así:

$$\begin{aligned} 1A_{16} &= (1_{16} \times 16^1) + (A_{16} \times 16^0) \\ &= (1_{10} \times 16^1) + (10_{10} \times 16^0) = 26 \end{aligned}$$

Pero la notación hexadecimal no se utiliza sólo para representar enteros. Se emplea como notación concisa para representar cualquier secuencia de dígitos binarios, ya represente texto, números, o cualquier otro tipo de datos. Las razones para utilizar la notación hexadecimal son:

1. Es más compacta que la notación binaria.
2. En la mayoría de los computadores, los datos binarios ocupan múltiplos de 4 bits y, por tanto, múltiplos de un dígito decimal.
3. Es extremadamente fácil la conversión entre binario y hexadecimal.

Como ejemplo del último punto, considere la cadena binaria 110111100001. Su equivalente es:

$$\begin{array}{ccc} 1101 & 1110 & 0001 = DEI_{16} \\ D & E & I \end{array}$$

Este proceso se realiza de forma tan natural, que un programador experimentado puede convertir mentalmente las representaciones visuales de los datos binarios a su equivalente hexadecimal sin necesidad de escribirlos. Es bastante probable que usted no necesite nunca esta habilidad particular. A pesar de ello, hemos incluido esta discusión en el texto, porque puede que se encuentre en ocasiones con datos en notación hexadecimal.

## CAPÍTULO 9

# **Repertorios de instrucciones: características y funciones**

- 9.1. Características de las instrucciones máquina
- 9.2. Tipos de operandos
- 9.3. Tipos de datos en el Pentium II y el PowerPC
- 9.4. Tipos de operaciones
- 9.5. Tipos de operaciones en el Pentium II y el PowerPC
- 9.6. Lenguaje ensamblador
- 9.7. Lecturas recomendadas
- 9.8. Problemas
- Apéndice 9A. Pilas
- Apéndice 9B. «Little-, Big y Bi-Endian»

| Dirección<br>de byte | Asignación de direcciones Big-endian |     |     |    |     |     |     |     |  |  |  |  |
|----------------------|--------------------------------------|-----|-----|----|-----|-----|-----|-----|--|--|--|--|
|                      | 11                                   | 12  | 13  | 14 |     |     |     |     |  |  |  |  |
| 00                   | 00                                   | 01  | 02  | 03 | 04  | 05  | 06  | 07  |  |  |  |  |
| 08                   | 21                                   | 22  | 23  | 24 | 25  | 26  | 27  | 28  |  |  |  |  |
| 10                   | 08                                   | 09  | 0A  | 0B | 0C  | 0D  | 0E  | 0F  |  |  |  |  |
| 18                   | 31                                   | 32  | 33  | 34 | 'A' | 'B' | 'C' | 'D' |  |  |  |  |
| 20                   | 10                                   | 11  | 12  | 13 | 14  | 15  | 16  | 17  |  |  |  |  |
|                      | 'E'                                  | 'F' | 'G' |    | 51  | 52  |     |     |  |  |  |  |
|                      | 18                                   | 19  | 1A  | 1B | 1C  | 1D  | 1E  | 1F  |  |  |  |  |
|                      | 61                                   | 62  | 63  | 64 |     |     |     |     |  |  |  |  |
|                      | 20                                   | 21  | 22  | 23 |     |     |     |     |  |  |  |  |

- Los elementos esenciales de una instrucción del computador son: el código de operación, que especifica la operación a realizar; las referencias a operandos fuente y destino, que especifican la ubicación de las entradas y salidas para la operación; y la referencia a la siguiente instrucción, que usualmente está implícita.
- Los códigos de operación especifican las operaciones dentro de una de las siguientes categorías: operaciones aritméticas y lógicas; transferencia de datos entre dos registros, entre registros y memoria o entre dos posiciones de memoria; entrada/salida (E/S); y control.
- Las referencias a operandos especifican registros o posiciones de memoria de datos operandos. Los datos pueden ser de diversos tipos: direcciones, números, caracteres o datos lógicos.
- Una característica arquitectural común de los procesadores es la utilización de una pila, que puede estar visible o no al programador. Las pilas se emplean para gestionar las llamadas y retornos de procedimientos, y pueden contemplarse como una forma alternativa de direccionar memoria. Las operaciones básicas con la pila son PUSH (introducir), POP (extraer), y operaciones con una o dos posiciones de la cabecera de la pila. Las pilas normalmente se implementan de manera que crecen de las direcciones más altas hacia las más bajas.
- Los procesadores pueden clasificarse como big-endian, little-endian, y bi-endian. Un dato numérico multi-byte que se almacena con el byte más significativo en la dirección numérica más baja, se memoriza en la forma big-endian; si se memoriza con el byte más significativo en la dirección más alta, lo hace en la forma little-endian. Un procesador bi-endian puede manejar ambos estilos de memorización.

**G**ran parte de lo tratado en este libro no es fácilmente visible para el usuario o programador de un computador. Si un programador está usando un lenguaje de alto nivel, como el Pascal o el Ada, muy poco de la arquitectura de la máquina está visible.

Un punto de encuentro en que el diseñador del computador y el programador pueden ver la misma máquina, es el repertorio de instrucciones. Desde el punto de vista del diseñador, el conjunto de instrucciones máquina informa de las especificaciones funcionales de la CPU: implementar la CPU es una tarea que, en buena parte, implica implementar el repertorio de instrucciones máquina. Desde el punto de vista del usuario, quien elige programar en lenguaje máquina (realmente en lenguaje ensamblador; véase Sección 9.6) se hace consciente de la estructura de registros y de memoria, de los tipos de datos que soporta directamente la máquina y del funcionamiento de la ALU.

La descripción del repertorio de instrucciones máquina de un computador es un paso más hacia la explicación de la CPU del computador. De acuerdo con esto, dedicaremos este capítulo y el siguiente a las instrucciones máquina, para volver después sobre la estructura y funcionamiento de las CPU.

## 9.1. CARACTERÍSTICAS DE LAS INSTRUCCIONES MÁQUINA

El funcionamiento de la CPU está determinado por las instrucciones que ejecuta. Estas instrucciones se denominan *instrucciones máquina* o *instrucciones del computador*. Al conjunto de instrucciones distintas que puede ejecutar la CPU se le denomina *repertorio de instrucciones* de la CPU.

### ELEMENTOS DE UNA INSTRUCCIÓN MÁQUINA

Cada instrucción debe contener la información que necesita la CPU para su ejecución. La Figura 9.1, que es una repetición de la Figura 3.6, muestra los pasos involucrados en la ejecución de instrucciones e, implícitamente, define los elementos constitutivos de una instrucción máquina. Dichos elementos son:

- **Código de operación:** Especifica la operación a realizar (suma, E/S, etc.). La operación se indica mediante un código binario, denominado «código de operación» o, abreviadamente, «codop».
- **Referencia a operandos fuente:** La operación puede implicar a uno o más operandos fuente, es decir, operandos que son entradas para la instrucción.
- **Referencia al operando resultado:** La operación puede producir un resultado.
- **Referencia a la siguiente instrucción:** Dice a la CPU de dónde captar la siguiente instrucción tras completarse la ejecución de la instrucción actual.

La siguiente instrucción a captar está en memoria principal o, en el caso de un sistema de memoria virtual, bien en memoria principal o en memoria secundaria (disco). En la mayoría de los casos, la siguiente instrucción a captar sigue inmediatamente a la instrucción en ejecución. En tales casos no hay referencia explícita a la siguiente instrucción. Cuando sea necesaria una referencia explícita, debe suministrarse la dirección de memoria principal o de memoria virtual. La forma en que se da dicha dirección se discute en el Capítulo 10.

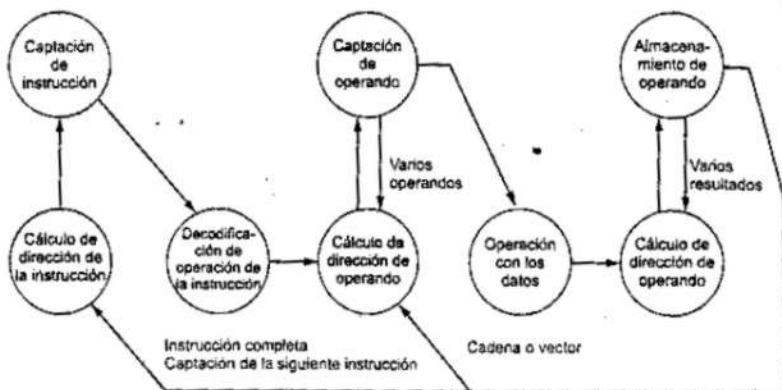


Figura 9.1. Diagrama de estados de un ciclo de instrucción.

Los operandos fuente y resultado pueden estar en alguna de las siguientes áreas:

- Memoria principal o virtual: Como en las referencias a instrucciones siguientes, debe indicarse la dirección de memoria principal o de memoria virtual.
- Registro de la CPU: Salvo raras excepciones, una CPU contiene uno o más registros que pueden ser referenciados por instrucciones máquina. Si sólo existe un registro, la referencia a él puede ser implícita. Si existe más de uno, cada registro tendrá asignado un número único, y la instrucción debe contener el número del registro deseado.
- Dispositivo de E/S: La instrucción debe especificar el módulo y dispositivo de E/S para la operación. En el caso de E/S asignadas en memoria, se dará otra dirección de memoria principal o virtual.

## REPRESENTACIÓN DE LAS INSTRUCCIONES

Dentro del computador, cada instrucción se representa por una secuencia de bits. La instrucción está dividida en campos, correspondientes a los elementos constitutivos de la misma. La Figura 9.2 muestra un ejemplo sencillo de formato de instrucción. Otro ejemplo, el formato de instrucciones del IAS, se mostró en la Figura 2.2. En la mayoría de los repertorios de instrucciones se emplea más de un formato. Durante su ejecución, la instrucción se escribe en un registro de instrucción (IR) de la CPU. La CPU debe ser capaz de extraer los datos de los distintos campos de la instrucción para realizar la operación requerida.

Es difícil, tanto para los programadores como para los lectores de un libro de texto, manejar las representaciones binarias de las instrucciones máquina. Por ello, es una práctica

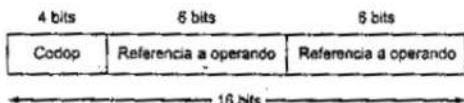


Figura 9.2. Un formato de instrucciones sencillo.

común utilizar *representaciones simbólicas* de las instrucciones máquina. Un ejemplo se dio en la Tabla 2.1 para el repertorio de instrucciones del IAS.

Los codops se representan mediante abreviaturas, denominadas *nemotécnicos*, que indican la operación en cuestión. Ejemplos usuales son:

|      |                                        |
|------|----------------------------------------|
| ADD  | Sumar                                  |
| SUB  | Restar                                 |
| MPY  | Multiplicar                            |
| DIV  | Dividir                                |
| LOAD | Cargar datos de memoria                |
| STOR | Almacenar datos en memoria (memorizar) |

Los operandos también suelen representarse simbólicamente. Por ejemplo, la instrucción

ADD R, Y

puede significar «sumar el valor contenido en la posición de datos Y al contenido del registro R». En este ejemplo, Y hace referencia a la dirección de una posición de memoria, y R a un registro particular. Observe que la operación se realiza con el contenido de la posición, no con su dirección.

Es posible pues, escribir un programa en lenguaje máquina de forma simbólica. Cada codop simbólico tiene una representación binaria fija, y el programador especifica la posición de cada operando simbólico. Por ejemplo, el programador podría comenzar con una lista de definiciones:

X = 513  
Y = 514

Y así sucesivamente. Un sencillo programa aceptaría como entrada esta información simbólica, convertiría los codops y referencias a operandos a forma binaria y construiría las instrucciones máquina binarias.

Es raro encontrar ya programadores en lenguaje máquina. La mayoría de los programas actuales se escriben en un lenguaje de alto nivel o, en ausencia del mismo, en lenguaje ensamblador, sobre el que trataremos al final de este capítulo. No obstante, el lenguaje máquina simbólico sigue siendo útil para describir las instrucciones máquina, y con ese fin lo utilizaremos.

## TIPOS DE INSTRUCCIONES

Considere una instrucción de alto nivel, tal y como se expresaría en un lenguaje como el BASIC o el FORTRAN. Por ejemplo,

X = X + Y

Esta sentencia ordena al computador sumar los valores almacenados en X y en Y, y poner el resultado en X. ¿Cómo se podría realizar lo mismo con instrucciones máquina? Supongamos que las variables X e Y corresponden a las posiciones 513 y 514. Considerando un repertorio simple de instrucciones máquina, la operación podría llevarse a cabo con tres instrucciones:

1. Cargar un registro con el contenido de la posición de memoria 513.
2. Sumar al registro el contenido de la posición de memoria 514.
3. Memorizar el contenido del registro en la posición de memoria 513.

Como se observa, una sola instrucción BASIC puede necesitar de tres instrucciones máquina. Este es un caso típico de relación entre un lenguaje de alto nivel y un lenguaje máquina. Un lenguaje de alto nivel expresa las operaciones de forma algebraica concisa, utilizando variables. Un lenguaje máquina expresa las operaciones de una manera elemental, implicando operaciones de transferencia de datos a, o desde, registros.

Con el sencillo ejemplo anterior como guía, consideremos qué tipos de instrucciones deben incluirse en un computador real. Debiera tener un conjunto de instrucciones que permitieran al usuario formular cualquier tarea de procesamiento de datos. Otra forma de verlo sería considerar las posibilidades de un lenguaje de programación de alto nivel. Cualquier programa escrito en alto nivel, debe traducirse a lenguaje máquina para ser ejecutado. Por tanto, el repertorio de instrucciones máquina debe ser suficientemente amplio como para expresar cualquiera de las instrucciones de un lenguaje de alto nivel. Teniendo esto presente, los tipos de instrucciones se pueden clasificar de la siguiente manera:

- De procesamiento de datos: Instrucciones aritméticas y lógicas
- De almacenamiento de datos: Instrucciones de memoria.
- De transferencia de datos: Instrucciones de E/S.
- De control: Instrucciones de comprobación y de bifurcación.

Las instrucciones *aritméticas* proporcionan capacidad computacional para procesar datos numéricos. Las instrucciones *lógicas* (booleanas) operan sobre los bits de una palabra, en lugar de considerarlos como números, proporcionando, por tanto, capacidad para el procesamiento de cualquier otro tipo de datos que el usuario quiera emplear. Estas operaciones se realizan principalmente con datos en registros de la CPU. Por lo tanto, debe haber instrucciones de *memoria* para transferir los datos entre la memoria y los registros. Las instrucciones de *E/S* se necesitan para transferir programas y datos a memoria, y devolver los resultados de los cálculos al usuario. Las instrucciones de *comprobación* o test se emplean para comprobar el valor de una palabra de datos o el estado de un cálculo. Las de *bifurcación* se usan entonces para bifurcar a diferentes conjuntos de instrucciones dependiendo de la decisión tomada.

Examinaremos los distintos tipos de instrucciones con mayor detalle más adelante, en este mismo capítulo.

## NÚMERO DE DIRECCIONES

Una de las formas tradicionales de describir la arquitectura de un procesador es en términos del número de direcciones contenidas en cada instrucción. Esta dimensión se va haciendo menos significativa a medida que aumenta la complejidad del diseño de la CPU. A pesar de ello, merece la pena extenderse y analizar dicha distinción.

¿Cuál es el número máximo de direcciones que serían necesarias en una instrucción? evidentemente, las instrucciones aritméticas y lógicas son las que requieren más operandos. Prácticamente todas las operaciones aritméticas y lógicas son, o bien unarias (un operando), o bien binarias (dos operandos). Así pues, necesitaríamos un máximo de dos direcciones para referenciar operandos. El resultado de una operación debe almacenarse, lo que sugiere una tercera dirección.

Finalmente, tras completar una instrucción, debe captarse la siguiente, y su dirección es, pues, necesaria.

El razonamiento anterior sugiere como plausible que una instrucción incluyera cuatro referencias a direcciones: dos operandos, un resultado y la dirección de la instrucción siguiente. En la práctica, es muy raro encontrar instrucciones que contengan cuatro direcciones. La

mayoría de las CPU trabajan con instrucciones de una, dos o tres direcciones, siendo implícita la dirección de la instrucción siguiente (obtenida a partir del contador de programa).

La Figura 9.3 compara instrucciones típicas de una, dos y tres direcciones, que podrían utilizarse para calcular  $Y = (A - B) + (C + D \times E)$ . Con tres direcciones, cada instrucción especifica dos posiciones de operandos y la posición del resultado. Dado que no queremos alterar el valor de ninguna posición de operando, se utiliza una posición temporal, T, para almacenar resultados intermedios. Observe que hay cuatro instrucciones, y que la expresión original tenía cinco operandos.

Las instrucciones con tres direcciones no son comunes, ya que requieren formatos relativamente largos para albergar las tres referencias. Con instrucciones de dos direcciones, y para operaciones binarias, una de las direcciones debe hacer el servicio doble de uno de los operandos y del resultado. Así pues, la instrucción SUB Y, B realiza el cálculo  $Y - B$  y guarda el resultado en Y. El formato de dos direcciones reduce el espacio necesario, pero resulta algo engorroso. Para evitar que se altere el valor de un operando, se utiliza una instrucción MOVE para transferir uno de los valores a una posición temporal o de resultados, antes de realizar el cálculo. Nuestro programa ejemplo se amplía a seis instrucciones.

La instrucción de una sola dirección es aún más simple. Para que funcione, una segunda dirección debe estar implícita. Esto fue lo usual en las primeras máquinas, en las que la dirección implícita era un registro de la CPU conocido como *acumulador*, o AC. El acumulador contiene uno de los operandos, y se emplea para almacenar el resultado. En nuestro ejemplo se necesitan 8 instrucciones para realizar la tarea.

Es, de hecho, posible arreglárselas con cero direcciones para algunas instrucciones. Las instrucciones con cero direcciones son aplicables a una organización especial de memoria, denominada *pila* (stack). Una pila es un conjunto de posiciones del tipo *last-in-first-out* (el último en entrar es primero en salir). La pila está en una posición conocida y, a menudo, al menos los dos elementos de su cabecera están en registros de la CPU. Así pues, las instrucciones con cero direcciones referenciarían dichos elementos de la cabecera. Las memorias pila se describen en el Apéndice 9A, y su uso se trata también en el Capítulo 10.

La Tabla 9.1 resume posibles interpretaciones a considerar para las instrucciones de 0, 1, 2 o 3 direcciones. En todos los casos se supone que la dirección de la siguiente instrucción está implícita, y que se va a realizar una operación con dos operandos fuente y un resultado.

| Instrucción | Comentario       |
|-------------|------------------|
| SUB Y, A, B | $Y - A - B$      |
| MPY T, D, E | $T - D \times E$ |
| ADD T, T, C | $T - T + C$      |
| DIV Y, Y, T | $Y - Y \div T$   |

(a) Instrucciones de tres direcciones

| Instrucción | Comentario       |
|-------------|------------------|
| MOVE Y, A   | $Y - A$          |
| SUB Y, B    | $Y - Y - B$      |
| MOVE T, D   | $T - D$          |
| MPY T, E    | $T - T \times E$ |
| ADD T, C    | $T - T + C$      |
| DIV Y, T    | $Y - Y \div T$   |

(b) Instrucciones de dos direcciones

| Instrucción | Comentario         |
|-------------|--------------------|
| LOAD D      | $AC - D$           |
| MPY E       | $AC - AC \times E$ |
| ADD C       | $AC - AC + C$      |
| STOR Y      | $Y - AC$           |
| LOAD A      | $AC - A$           |
| SUB B       | $AC - AC - B$      |
| DIV Y       | $AC - AC \div Y$   |
| STOR Y      | $Y - AC$           |

(c) Instrucciones de una dirección

Figura 9.3. Programas para calcular  $Y = (A - B) + (C + D \times E)$ .

Tabla 9.1. Utilización de las direcciones de las instrucciones  
(instrucciones sin bifurcación)

| Número de direcciones | Representación simbólica | Interpretación                       |
|-----------------------|--------------------------|--------------------------------------|
| 3                     | OP A, B, C               | $A \leftarrow B \text{ OP } C$       |
| 2                     | OP A, B                  | $A \leftarrow A \text{ OP } B$       |
| 1                     | OP A                     | $AC \leftarrow AC \text{ OP } A$     |
| 0                     | OP                       | $T \leftarrow T \text{ OP } (T - 1)$ |

AC = acumulador

T = cabecera de la pila

A, B, C = posiciones de memoria o registros

El número de direcciones por instrucción es una decisión básica de diseño. Menos direcciones por instrucción significa instrucciones más primarias, lo que requiere una CPU menos compleja. También da lugar a instrucciones más cortas. Por otra parte, los programas contienen más instrucciones, lo que normalmente supone mayor tiempo de ejecución y programas más largos y complejos. Hay también un umbral importante entre instrucciones de una y de múltiples direcciones. Con instrucciones de una sola dirección, el programador tiene generalmente a su disposición sólo un registro de uso general: el acumulador. Con instrucciones de múltiples direcciones suele disponerse de múltiples registros de uso general. Esto permite que algunas operaciones se realicen sólo con registros. Ya que los accesos a registros son más rápidos que a memoria, se acelera la ejecución. Por razones de flexibilidad y facilidad para utilizar varios registros, la mayoría de las máquinas contemporáneas emplean una combinación de instrucciones de dos y de tres direcciones.

Las decisiones de diseño asociadas con la elección del número de direcciones por instrucción son complicadas debido a otros factores. Un aspecto a considerar es si una dirección hace referencia a una posición de memoria o a un registro. Ya que hay menos registros, se necesitan menos bits para referenciarlos. Además, como veremos en el capítulo siguiente, una máquina puede permitir diversos modos de direccionamiento, y la especificación del modo consume uno o más bits. El resultado es que la mayoría de los diseños de las CPU hacen uso de varios formatos de instrucciones.

## DISEÑO DEL REPERTORIO DE INSTRUCCIONES

Uno de los aspectos más interesantes y más analizados del diseño de un computador, es el diseño del repertorio de instrucciones del lenguaje máquina. El diseño de un repertorio de instrucciones es muy complejo, ya que afecta a muchos aspectos del computador. El repertorio de instrucciones define muchas de las funciones realizadas por la CPU y tiene, por tanto, un efecto significativo sobre la implementación de la misma. El repertorio de instrucciones es el medio que tiene el programador para controlar la CPU. En consecuencia, deben considerarse las necesidades del programador a la hora de diseñar el repertorio de instrucciones.

Puede sorprender saber que algunos de los aspectos más básicos relativos al diseño de repertorios de instrucciones siguen siendo temas de controversia. Los más importantes entre dichos aspectos de diseño son:

- **Repertorio de operaciones:** Cuántas y qué operaciones considerar, y cuán complejas deben ser.
- **Tipos de datos:** Los distintos tipos de datos con los que se efectúan operaciones.
- **Formatos de instrucciones:** Longitud de la instrucción (en bits), número de direcciones, tamaño de los distintos campos, etc.

- **Registros:** Número de registros de la CPU que pueden ser referenciados por instrucciones, y su uso.
- **Direccionamiento:** El modo o modos de direccionamiento mediante los cuales puede especificarse la dirección de un operando.

Estos aspectos están fuertemente interrelacionados, y deben considerarse conjuntamente en el diseño de un repertorio de instrucciones. Este libro, por supuesto, debe considerarlos en secuencia, pero se intentará mostrar las relaciones entre ellos.

Dada la importancia del tema, un bloque amplio de la Parte III del libro se dedica al diseño del repertorio de instrucciones. A continuación de esta sección de repaso, el capítulo examina los tipos de datos y repertorios de operaciones. El Capítulo 10 trata los modos de direccionamiento (considerando también los registros) y los formatos de instrucciones. El Capítulo 12 se dedica al estudio del «computador de conjunto reducido de instrucciones» (RISC, Reduced Instruction Set Computer). La arquitectura RISC cuestiona muchas de las decisiones tomadas sobre repertorios de instrucciones de computadores comerciales contemporáneos. Un ejemplo de máquina RISC es el PowerPC. En este capítulo y en el que sigue utilizaremos este procesador como ejemplo. Sin embargo, la importancia del PowerPC en el contexto de los RISC se discutirá en el Capítulo 12.

## TIPOS DE OPERANDOS

Las instrucciones máquina operan con datos. Las categorías generales más importantes de datos son:

- Direcciones
- Números
- Caracteres
- Datos lógicos

Cuando tratemos los modos de direccionamiento en el Capítulo 10, veremos que las direcciones son, de hecho, un tipo de datos. En muchos casos, debe realizarse algún cálculo sobre la referencia a un operando de una instrucción a fin de determinar la dirección de memoria principal o virtual. En este contexto, las direcciones pueden considerarse como números enteros sin signo.

Otros tipos de datos comunes son los números, los caracteres y los datos lógicos, y cada uno de ellos se analiza brevemente en esta sección. Aparte de éstos, en algunas máquinas se utilizan tipos o estructuras de datos específicos. Por ejemplo, puede haber operaciones máquina que trabajan directamente con listas o cadenas de caracteres.

### NÚMEROS

Todos los lenguajes máquina incluyen tipos de datos numéricos. Incluso en el procesamiento de datos no numéricos se necesitan números que actúen como contadores, longitudes de campos, etc. Una distinción importante entre los números utilizados en las matemáticas ordinarias y los almacenados en un computador, es que estos últimos están limitados. Esto es cierto en dos sentidos. En primer lugar, hay un límite para la magnitud de los números representables en una máquina; en segundo lugar, en el caso de números en coma flotante, su pre-

cisión está limitada. Por tanto, el programador debe ser consciente de las consecuencias del redondeo, el desbordamiento, o el desbordamiento a cero.

En los computadores son usuales tres tipos de datos numéricos:

- Enteros o en coma fija
- En coma flotante
- En decimal

Los dos primeros se vieron con detalle en el Capítulo 8. Falta comentar algo sobre los números decimales.

Aunque todas las operaciones internas del computador son en esencia binarias, los usuarios del sistema utilizamos números decimales. Es necesario pues, convertir de decimal a binario las entradas, y de binario a decimal las salidas. Para aplicaciones en las que hay muchas entradas/salidas frente a pocos cálculos, y cálculos comparativamente simples, es preferible memorizar y operar con los números directamente en su forma decimal. La representación más común para ello es la de decimal empaquetado.

En decimal empaquetado, cada dígito decimal se representa mediante un código de 4 bits, según la manera habitual. Es decir: 0 = 0000; 1 = 0001; ...; 8 = 0100 y 9 = 1001. Observe que resulta un código bastante ineficiente, ya que sólo se emplean 10 de las 16 posibles combinaciones de 4 bits. Para construir números, se van encadenando códigos de 4 bits, normalmente formando múltiplos de 8 bits. Así, el código de 246 es 0000001001000110. Este código es, claramente, menos compacto que la representación binaria directa, pero evita la necesidad de conversiones. Los números negativos pueden representarse incluyendo un dígito de signo de 4 bits, bien a la izquierda o bien a la derecha de la cadena de dígitos decimales empaquetados. Por ejemplo, el código 1111 podría representar el signo menos.

Muchas máquinas tienen instrucciones aritméticas destinadas a operar directamente con números en decimal empaquetado. Los algoritmos correspondientes son bastante similares a los descritos en la Sección 8.3, pero deben tener en cuenta la operación de acarreo decimal.

## CARACTERES

Una forma bastante común de datos es el texto o secuencias de caracteres. Aunque la información textual sea más conveniente para las personas, no puede ser almacenada o transmitida fácilmente en forma de caracteres por los sistemas de comunicación y de procesamiento de datos. Tales sistemas están diseñados para datos binarios. Por lo tanto, se han ideado diversos códigos que permiten representar caracteres mediante secuencias de bits. Tal vez el primer ejemplo fue el código Morse. Hoy en día, el código de caracteres más utilizado es el Alfabeto de Referencia Internacional (IRA, International Reference Alphabet), conocido en los Estados Unidos como ASCII (American Standard Code for Information Interchange); Código Estándar Americano para Intercambio de Información (véase Tabla 6.1). Cada carácter es representado en este código por un patrón distinto de 7 bits. Pueden representarse por tanto 128 caracteres diferentes. Este número es mayor que el necesario para representar los caracteres impresos, utilizando algunos de los patrones para representar caracteres de control. Algunos de éstos se emplean para controlar la impresión de caracteres en una página. Otros caracteres de control están dedicados para procedimientos de comunicación. Los caracteres codificados en el IRA se memorizan y transmiten utilizando casi siempre 8 bits por carácter. El octavo bit puede fijarse a 0, o utilizarse como bit de paridad para la detección de errores. En el segundo caso, el valor del bit se pone de manera que el número total de unos en cada octeto sea o siempre impar (paridad impar) o siempre par (paridad par).

Observe en la Tabla 6.1 que, con el IRA, los patrones de bits 011XXXX representan los dígitos 0 a 9, mediante sus valores binarios equivalentes, 0000 a 1001, en los 4 bits de la derecha. Esta codificación coincide con la empleada en decimal empaquetado, facilitándose así la conversión entre IRA de 7 bits y decimal empaquetado de 4 bits.

Otro código utilizado para caracteres es el Código de Intercambio Decimal Codificado en Binario Ampliado (EBCDIC, Extended Binary Coded Decimal Interchange Code). Este código de 8 bits se emplea en las máquinas IBM S/370. Como el IRA, el EBCDIC es compatible con el decimal empaquetado. En el caso del EBCDIC, los códigos 11110000 a 11111001 representan los dígitos 0 a 9.

## DATOS LÓGICOS

Normalmente, cada palabra o cualquier otra unidad direccionable (byte, media palabra, etc.) es tratada como una unidad de datos individual. Sin embargo, a veces es útil considerar una unidad de  $n$  bits como  $n$  elementos o datos de 1 bit, donde cada elemento tiene un valor 1 o 0. Cuando los datos son vistos de esta manera, se consideran datos *lógicos*.

Esta representación orientada a bits tiene dos ventajas. La primera es que a veces puede interesarlos almacenar una matriz de elementos binarios o booleanos, en la que cada elemento pueda tomar sólo los valores 1 (verdadero) o 0 (falso). Con datos lógicos, la memoria puede ser entonces utilizada más eficientemente. En segundo lugar, hay ocasiones en las que queremos manipular bits individuales de un dato. Por ejemplo: cuando se implementan operaciones de coma flotante en software, en algunos casos necesitamos desplazar bits de las mantis. Otro ejemplo es al convertir de ASCII a decimal empaquetado, donde se necesita extraer los 4 bits de la derecha de cada byte.

Obsérvese cómo, en los ejemplos precedentes, los mismos datos son tratados unas veces como datos lógicos y otras como numéricos o como texto. El «tipo» de una unidad de datos viene determinado por la operación que se esté realizando con ella. Aunque esto no sea lo normal con los lenguajes de alto nivel (como por ejemplo el Pascal), es casi siempre así en el caso de lenguajes máquina.

## TIPOS DE DATOS EN EL PENTIUM II Y EL PowerPC

### TIPOS DE DATOS EN EL PENTIUM II

El Pentium II puede tratar tipos de datos de 8 (byte), 16 (palabra), 32 (palabra doble) y 64 (palabra cuádruple) bits de longitud. Para posibilitar una flexibilidad máxima en las estructuras de datos, y una utilización eficiente de la memoria, las palabras no tienen por qué estar alineadas con las direcciones pares de memoria, ni las palabras dobles alineadas con las direcciones divisibles por 4, ni las cuádruples con direcciones divisibles por 8. Sin embargo, cuando se accede a los datos a través de un bus de 32 bits, su transferencia tiene lugar en unidades de palabras dobles, empezando en direcciones divisibles por cuatro. El procesador convierte las peticiones con valores no alineados, en una secuencia de peticiones adaptada a la forma de trasferencia en el bus. Como en todas las máquinas Intel 80X86, el Pentium II emplea el estilo «little-endian»; es decir: el byte menos significativo es almacenado en la dirección más baja (véase el Apéndice 9A).

El byte, la palabra, la palabra doble y la cuádruple, son referidas como tipos generales de datos. Además, el Pentium II admite una variedad impresionante de tipos de datos específicos:

Tabla 9.2. Tipos de datos del Pentium II

| Tipo de datos                                      | Descripción                                                                                                                                                                                                                  |
|----------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| General                                            | Posiciones de byte, de palabra (16 bits), de palabra doble (32 bits), y de palabra cuádruple (64 bits), con contenido binario arbitrario.                                                                                    |
| Entero                                             | Un valor binario con signo contenido en un byte, una palabra, o una palabra doble, representado en complemento a dos.                                                                                                        |
| Ordinal                                            | Un entero sin signo contenido en un byte, una palabra, o una palabra doble.                                                                                                                                                  |
| Decimal codificado en binario (BCD) desempaquetado | Representación de un dígito BCD en el rango de 0 a 9, con un dígito en cada byte.                                                                                                                                            |
| BCD empaquetado                                    | Representación empaquetada de dos dígitos BCD en un byte; valor en el rango de 0 a 99.                                                                                                                                       |
| Puntero de proximidad (puntero cercano)            | Una dirección efectiva de 32 bits que representa el desplazamiento dentro de un segmento. Utilizado para todos los punteros en una memoria no segmentada y para referencias dentro de un segmento en una memoria segmentada. |
| Campo de bits                                      | Una secuencia contigua de bits en la que cada posición de bit se considera como unidad independiente. Una cadena de bits puede comenzar en cualquier posición de cualquier byte y puede contener hasta $2^{32} - 1$ bits.    |
| Cadena de bytes                                    | Una secuencia contigua de bytes, de palabras, o de palabras dobles, que contiene de 0 a $2^{32} - 1$ bytes.                                                                                                                  |
| Coma flotante                                      | Véase la Figura 9.4.                                                                                                                                                                                                         |

cos, que son reconocidos y procesados mediante instrucciones concretas. La Tabla 9.2 resume estos tipos.

El tipo coma flotante se refiere realmente a un conjunto de tipos utilizados por la unidad de coma flotante, y que son procesados mediante instrucciones de coma flotante. Como ilustra la Figura 9.4, las instrucciones de coma flotante pueden operar con enteros y con enteros decimales empaquetados, así como con números en coma flotante. Los enteros están en representación de complemento a dos y pueden ser de 16, 32 o 64 bits. Los enteros decimales empaquetados se almacenan en representación signo-magnitud con 18 dígitos en el rango de 0 a 9. Las tres representaciones en coma flotante se ajustan al estándar IEEE 754.

## TIPOS DE DATOS EN EL PowerPC

El PowerPC puede manejar tipos de datos de 8 (byte), 16 (media palabra), 32 (palabra) y 64 (palabra doble) bits de longitud. Algunas instrucciones requieren que los operandos de memoria estén alineados con una frontera de 32 bits. Sin embargo, en general no es necesario el alineamiento. Una característica interesante del PowerPC es que puede utilizar bien el estilo «little-endian» o el «big-endian»; es decir, el byte menos significativo puede estar almacenado, bien en la dirección más baja, o bien en la más alta (véase el Apéndice 9A).

El byte, la media palabra, la palabra y la palabra doble, son tipos de datos generales. El procesador interpreta el contenido de un elemento de datos dado dependiendo de la instrucción. El procesador de coma fija reconoce los siguientes tipos de datos:

| Formatos de datos     | Rango       | Precisión  | Byte más significativo |                   |          |          |          |          |          |          |          |          | Byte con dirección más alta |       |       |       |       |       |       |       |       |       |                     |
|-----------------------|-------------|------------|------------------------|-------------------|----------|----------|----------|----------|----------|----------|----------|----------|-----------------------------|-------|-------|-------|-------|-------|-------|-------|-------|-------|---------------------|
|                       |             |            | 7                      | 0                 | 7        | 0        | 7        | 0        | 7        | 0        | 7        | 0        | 7                           | 0     | 7     | 0     | 7     | 0     | 7     | 0     | 7     | 0     |                     |
| Entero de una palabra | $10^4$      | 16 bits    |                        |                   |          |          |          |          |          |          |          |          |                             |       |       |       |       |       |       |       |       |       | (Complemento a dos) |
|                       |             |            | 15                     | 0                 |          |          |          |          |          |          |          |          |                             |       |       |       |       |       |       |       |       |       |                     |
| Entero corto          | $10^9$      | 32 bits    |                        |                   |          |          |          |          |          |          |          |          |                             |       |       |       |       |       |       |       |       |       | (Complemento a dos) |
|                       |             |            | 31                     | 0                 |          |          |          |          |          |          |          |          |                             |       |       |       |       |       |       |       |       |       |                     |
| Entero largo          | $10^{18}$   | 64 bits    |                        |                   |          |          |          |          |          |          |          |          |                             |       |       |       |       |       |       |       |       |       | (Complemento a dos) |
|                       |             |            | 63                     | 0                 |          |          |          |          |          |          |          |          |                             |       |       |       |       |       |       |       |       |       |                     |
| BCD empacado          | $10^{18}$   | 18 dígitos | s                      | X                 | $d_{17}$ | $d_{16}$ | $d_{15}$ | $d_{14}$ | $d_{13}$ | $d_{12}$ | $d_{11}$ | $d_{10}$ | $d_9$                       | $d_8$ | $d_7$ | $d_6$ | $d_5$ | $d_4$ | $d_3$ | $d_2$ | $d_1$ | $d_0$ |                     |
|                       |             |            | 79                     | 71                |          |          |          |          |          |          |          |          |                             |       |       |       |       |       |       |       |       |       | 0                   |
| Precisión simple      | $10^{-38}$  | 24 bits    | s                      | Exponente sesgado | Mantisa  |          |          |          |          |          |          |          |                             |       |       |       |       |       |       |       |       |       |                     |
|                       |             |            | 31                     | 23                | 0        |          |          |          |          |          |          |          |                             |       |       |       |       |       |       |       |       |       |                     |
| Doble precisión       | $10^{-308}$ | 3 bits     | s                      | Exponente sesgado | Mantisa  |          |          |          |          |          |          |          |                             |       |       |       |       |       |       |       |       |       |                     |
|                       |             |            | 63                     | 51                | 0        |          |          |          |          |          |          |          |                             |       |       |       |       |       |       |       |       |       |                     |
| Precisión ampliada    | $10^{4932}$ | 64 bits    | s                      | Exponente sesgado | 1        | Mantisa  |          |          |          |          |          |          |                             |       |       |       |       |       |       |       |       |       |                     |
|                       |             |            | 79                     | 63 D              | 0        |          |          |          |          |          |          |          |                             |       |       |       |       |       |       |       |       |       |                     |

s = bit de signo (0 = positivo; 1 = negativo).

$d_n$  = dígito decimal (dos por cada byte).

X = bits sin significado; se ignoran cuando se carga; ceros cuando se memoria.

D = posición del punto binario implícito.

I = bit de la parte entera de la mantisa; en reales temporales se almacena; en simple y doble precisión está implícito.

Figura 9.4. Formatos de datos numéricos en el Pentium II.

- **Byte sin signo:** Puede utilizarse para operaciones lógicas o para aritméticas con enteros. Se carga de memoria en un registro general, completando con ceros hacia la izquierda hasta la longitud total del registro.
- **Media palabra sin signo:** Como antes, pero para cantidades de 16 bits.
- **Media palabra con signo:** Utilizado para operaciones aritméticas; cargado en memoria completando el signo hacia la izquierda hasta la longitud total del registro (es decir, se replica el bit de signo en todas las posiciones vacantes).
- **Palabra sin signo:** Utilizado para operaciones lógicas y como puntero de direcciones.
- **Palabra con signo:** Utilizado para operaciones aritméticas.

- **Palabra doble sin signo:** Utilizado como puntero de direcciones.
- **Cadena de bytes:** Desde 0 hasta 128 bytes de longitud.

Además, el PowerPC admite tipos de datos en coma flotante de precisión simple y doble definidos en el estándar IEEE 754.

## 9.4 TIPOS DE OPERACIONES

El número de códigos de operación (codops) diferentes varía ampliamente de una máquina a otra. Sin embargo, en todas las máquinas podemos encontrar los mismos tipos generales de operaciones. Una clasificación típica y útil es la siguiente:

- Transferencias de datos
- Aritméticas
- Lógicas
- De conversión
- De E/S
- De control del sistema
- De control de flujo

La Tabla 9.3 (basada en [HAYE88]) enumera tipos de instrucciones comunes de cada clase. Esta sección proporciona una revisión somera de los distintos tipos de operaciones, junto con una breve discusión sobre las acciones que realiza la CPU para ejecutar un tipo particular de las mismas (resumidas en la Tabla 9.4). Este último punto se analiza con más detalle en el Capítulo 11.

Tabla 9.3. Operaciones usuales de repertorios de instrucciones

| Tipo                    | Nombre de la operación                 | Descripción                                                       |
|-------------------------|----------------------------------------|-------------------------------------------------------------------|
| Transferencias de datos | Move (transferir)                      | Transfiere una palabra o un bloque desde una fuente a un destino. |
|                         | Store (memorizar)                      | Transfiere una palabra desde el procesador a memoria.             |
|                         | Load (cargar o captar)                 | Transfiere una palabra desde memoria al procesador.               |
|                         | Exchange (intercambiar)                | Intercambia los contenidos de la fuente y el destino.             |
|                         | Clear (reiniciar o poner a 0)          | Transfiere una palabra de ceros al destino.                       |
|                         | Set (poner a 1)                        | Transfiere una palabra de unos al destino.                        |
|                         | Push (introducir en la pila, «apilar») | Transfiere una palabra desde una fuente a la cabecera de la pila. |
|                         | Pop (extraer de la pila, «desapilar»)  | Transfiere una palabra desde la cabecera de la pila a un destino. |
| Aritméticas             | Add (sumar)                            | Calcula la suma de dos operandos.                                 |
|                         | Subtract (restar)                      | Calcula la diferencia de dos operandos.                           |
|                         | Multiply (multiplicar)                 | Calcula el producto de dos operandos.                             |
|                         | Divide (dividir)                       | Calcula el cociente de dos operandos.                             |
|                         | Absolute (valor absoluto)              | Sustituye el operando por su valor absoluto.                      |
|                         | Negate (opuesto)                       | Cambia el signo del operando.                                     |
|                         | Increment (incrementar)                | Suma 1 al operando.                                               |
|                         | Decrement (decrementar)                | Resta 1 del operando.                                             |

Tabla 9.3. Operaciones usuales de repertorios de instrucciones (continuación)

| ipo              | Nombre de la operación                                                                              | Descripción                                                                                                                                        |
|------------------|-----------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------|
| lógicas          | AND (producto lógico, Y)<br>OR (suma lógica, O)<br>NOT (complemento)<br>Exclusive-OR (OR-exclusiva) | Realizan, bit a bit, la operación lógica indicada.                                                                                                 |
|                  | Test (comprobar)                                                                                    | Comprueba la condición especificada; pone los indicadores («flags») en función del resultado.                                                      |
|                  | Compare (comparar)                                                                                  | Realiza la comparación lógica o aritmética de dos o más operandos; pone los indicadores («flags») en función del resultado.                        |
|                  | Set Control Variables<br>(fijar variables de control)                                               | Instrucciones que fijan controles para protección, gestión de interrupciones, control del temporizador, etc.                                       |
|                  | Shift (desplazamiento)                                                                              | Desplaza el operando a la izquierda (derecha), introduciendo valores constantes por el otro extremo.                                               |
|                  | Rotate (rotar)                                                                                      | Desplaza el operando a la izquierda (derecha) de forma cíclica.                                                                                    |
| Control de flujo | Jump (bifurcación o salto)                                                                          | Ruptura incondicional de flujo; carga el PC con la dirección especificada.                                                                         |
|                  | Jump Conditional<br>(salto condicional)                                                             | Comprueba la condición especificada; dependiendo de la condición, carga el PC con la dirección indicada o no hace nada.                            |
|                  | Jump to subroutine<br>(llamada a subrutina)                                                         | Guarda la información de control del programa en una posición conocida y salta a la dirección indicada.                                            |
|                  | Return (retorno)                                                                                    | Sustituye el contenido del PC y de otros registros por los de la posición conocida.                                                                |
|                  | Execute (ejecutar)                                                                                  | Capta el operando de la dirección indicada y lo ejecuta como una instrucción; no modifica el PC.                                                   |
|                  | Skip (salto implícito)                                                                              | Incrementa el PC, de manera que se salte la instrucción siguiente.                                                                                 |
|                  | Skip conditional (salto implícito condicional)                                                      | Comprueba la condición indicada; realiza el salto implícito o no hace nada, dependiendo de la condición.                                           |
|                  | Halt (parar)                                                                                        | Detiene la ejecución del programa.                                                                                                                 |
|                  | Wait (esperar)                                                                                      | Detiene la ejecución del programa; comprueba de forma repetitiva la condición especificada; reanuda la ejecución cuando se satisface la condición. |
|                  | No operation (no operación)                                                                         | No se ejecuta operación alguna, pero la ejecución del programa continúa.                                                                           |
| Entrada/salida   | Input (entrada)                                                                                     | Transfiere datos desde un puerto o dispositivo de E/S al destino (memoria principal o registro del procesador).                                    |
|                  | Output (salida)                                                                                     | Transfiere datos desde la fuente especificada a un puerto o dispositivo de E/S.                                                                    |
|                  | Start I/O (iniciar E/S)                                                                             | Transfiere instrucciones al procesador de E/S para iniciar operaciones de E/S.                                                                     |
|                  | Test I/O (comprobar E/S)                                                                            | Transfiere información de estado desde el sistema de E/S al destino especificado.                                                                  |
| Conversión       | Translate (traducir)                                                                                | Traducción de los valores de una sección de memoria, basada en una tabla de correspondencia.                                                       |
|                  | Convert (convertir)                                                                                 | Convierte el contenido de una palabra de un formato a otro (por ejemplo, de decimal empaquetado a binario).                                        |

Tabla 9.4. Acciones de la CPU para varios tipos de operaciones

|                        |                                                                                                                                                                                                                                                           |
|------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Transferencia de datos | Transfiere datos de una posición a otra.<br>Si se implica a la memoria:<br>Determina la dirección de memoria.<br>Realiza la transformación de direcciones de memoria virtual a real.<br>Comprueba la cache.<br>Inicia la lectura/escritura en/ds memoria. |
| Aritmética             | Puede implicar transferencias de datos, antes y/o después.<br>Realiza la operación en la ALU.<br>Actualiza códigos e indicadores de condición.                                                                                                            |
| Lógica                 | Lo mismo que en una aritmética.                                                                                                                                                                                                                           |
| Conversión             | Similar a la aritmética y a la lógica. Puede implicar a lógica especial para realizar la conversión.                                                                                                                                                      |
| Control de flujo       | Actualiza el contador de programa. En el caso de llamadas y retornos de subrutinas, gestiona la transferencia y enlace de parámetros.                                                                                                                     |
| E/S                    | Cursa una orden a un módulo de E/S.<br>En el caso de E/S asignada en memoria, determina la dirección de memoria correspondiente.                                                                                                                          |

## TRANSFERENCIA DE DATOS

El tipo de instrucción máquina más básico es la transferencia de datos. La instrucción de transferencia de datos debe especificar varias cosas. En primer lugar, deben especificarse las posiciones de los operandos fuente y destino. Cada posición podría ser de memoria, un registro, o la cabecera de la pila. En segundo lugar, debe indicarse la longitud de los datos a transferir. En tercer lugar, como en todas las instrucciones con operandos, debe especificarse el modo de direccionamiento para cada operando. Este último punto se trata en el Capítulo 10.

La elección de las instrucciones de transferencia de datos a incluir en un repertorio de instrucciones, es un ejemplo del tipo de compromisos a los que debe llegar un diseñador. Por ejemplo, la posición en general (memoria o registro) de un operando puede indicarse, bien en la especificación del codop, bien en la del operando. La Tabla 9.5 muestra ejemplos de las instrucciones de transferencia de datos más comunes del IBM S/370. Observe cómo hay variantes para indicar la cantidad de datos a transferir (8, 16, 32, o 64 bits). También hay diferentes instrucciones para transferencias entre registros, de registro a memoria y de memoria a registro. Por contra, el VAX tiene una instrucción de transferencia (MOV) con variantes según la cantidad de datos a transferir, pero que especifica como parte del operando si éste está en un registro o en memoria. La aproximación considerada en el VAX es algo más fácil para el programador, quien tendría que manejar menos nemotécnicos. Sin embargo, es menos compacta que la considerada en el IBM S/370, ya que la posición (registro en lugar de memoria) de cada operando debe especificarse separadamente en la instrucción. Volveremos sobre esta cuestión cuando veamos, en el siguiente capítulo, los formatos de instrucciones.

En términos de la acción de la CPU, las operaciones de transferencia de datos son quizás las más sencillas. Cuando, tanto el origen como el destino, son registros, la CPU simplemente hace que los datos se transfieran de un registro a otro; ésta es una operación interna a la CPU. Si uno o ambos operandos están en memoria, la CPU debe realizar alguna o todas las siguientes tareas:

Tabla 9.5. Ejemplo de operaciones de transferencia de datos del IBM S/370

| Nemotécnico de la operación | Nombre          | Número de bits transferidos | Descripción                                                   |
|-----------------------------|-----------------|-----------------------------|---------------------------------------------------------------|
| L                           | Load            | 32                          | Transferencia de memoria a registro.                          |
| LH                          | Load Halfword   | 16                          | Transferencia de memoria a registro (media palabra).          |
| LR                          | Load            | 32                          | Transferencia de registro a registro.                         |
| LER                         | Load (Short)    | 32                          | Transferencia (corta) entre dos registros de coma flotante.   |
| LE                          | Load (Short)    | 32                          | Transferencia (corta) de memoria a registro de coma flotante. |
| LDR                         | Load (Long)     | 64                          | Transferencia (larga) entre dos registros de coma flotante.   |
| LD                          | Load (Long)     | 64                          | Transferencia (larga) de memoria a registro de coma flotante. |
| ST                          | Store           | 32                          | Transferencia de registro a memoria.                          |
| STH                         | Store Halfword  | 16                          | Transferencia de registro a memoria (media palabra).          |
| STC                         | Store Character | 8                           | Transferencia de registro a memoria (un carácter).            |
| STE                         | Store (Short)   | 32                          | Transferencia (corta) de registro de coma flotante a memoria. |
| STD                         | Store (Long)    | 64                          | Transferencia (larga) de registro de coma flotante a memoria. |

1. Calcular la dirección de memoria basándose en el modo de direccionamiento utilizado (Capítulo 10).
2. Si la dirección hace referencia a memoria virtual, traducir de dirección virtual a real.
3. Determinar si el elemento direccionado está en la cache.
4. Si no, cursar la orden al módulo de memoria.

## ARITMÉTICAS

La mayoría de las máquinas proporcionan las operaciones aritméticas básicas de suma, resta, multiplicación y división. Estas se tienen siempre para números enteros con signo (coma fija). A menudo, las proporcionan también para números en coma flotante y para decimales empacados.

Entre otras operaciones posibles hay varias instrucciones de un solo operando: por ejemplo:

- **Absolute:** obtiene el valor absoluto del operando.
- **Negate:** cambia el signo del operando.
- **Increment:** incrementa en 1 el operando.
- **Decrement:** decrementa en 1 el operando.

La ejecución de una instrucción aritmética puede implicar operaciones de transferencia de datos para ubicar los operandos como entradas a la ALU, y para almacenar la salida de la ALU. La Figura 3.5 ilustra las transferencias involucradas tanto en operaciones de transferencia como en aritméticas. Por supuesto que, además, la parte ALU de la CPU debe realizar la operación deseada.

Tabla 9.6. Operaciones lógicas básicas

| P | Q | NOT P | P AND Q | P OR Q | P XOR Q | P = Q |
|---|---|-------|---------|--------|---------|-------|
| 0 | 0 | 1     | 0       | 0      | 0       | 1     |
| 0 | 1 | 1     | 0       | 1      | 1       | 0     |
| 1 | 0 | 0     | 0       | 1      | 1       | 0     |
| 1 | 1 | 0     | 1       | 1      | 0       | 1     |

## LÓGICAS

La mayoría de las máquinas también disponen de diversas operaciones para manipular bits individuales dentro de una palabra o de otra unidad direccionable. Están basadas en operaciones booleanas (véase Apéndice A).

La Tabla 9.6 muestra algunas de las operaciones lógicas básicas que pueden realizarse con datos booleanos o binarios. La operación NOT invierte un bit. Las funciones lógicas más comunes con dos operandos son la AND, la OR y la OR-exclusiva (XOR). La EQUAL (igual) es una comprobación binaria bastante útil.

Las operaciones lógicas pueden aplicarse desde bit a bit hasta unidades lógicas de datos de  $n$  bits. Así, si dos registros contienen los datos:

$$(R1) = 10100101$$

$$(R2) = 00001111$$

entonces:

$$(R1) \text{ AND } (R2) = 00000101$$

donde la notación (X) significa el contenido de la posición X. Por lo tanto, la operación AND puede utilizarse como *máscara* para seleccionar ciertos bits de una palabra, poniendo a cero los restantes bits. En otro ejemplo, si dos registros contienen:

$$(R1) = 10100101$$

$$(R2) = 11111111$$

entonces:

$$(R1) \text{ XOR } (R2) = 01011010$$

Con una palabra puesta a todo unos, la operación XOR invierte todos los bits de la otra palabra (complemento a uno).

Además de las operaciones lógicas bit a bit, la mayoría de las máquinas ofrecen diversas funciones de desplazamiento y rotación. Las operaciones más básicas se ilustran en la Figura 9.5. En un *desplazamiento lógico* se desplazan a la derecha o a la izquierda los bits de la palabra. En un extremo, el bit saliente se pierde al desplazarse. En el otro extremo se introduce un 0. Los desplazamientos lógicos son útiles principalmente para aislar campos dentro de una palabra. Los ceros que se van introduciendo en la palabra desplazan la información no deseada, que se va perdiendo por el otro extremo.

Como ejemplo, suponga que queremos transmitir caracteres de datos, carácter a carácter, a un dispositivo de E/S. Si las palabras de memoria son de 16 bits y contienen 2 caracteres,

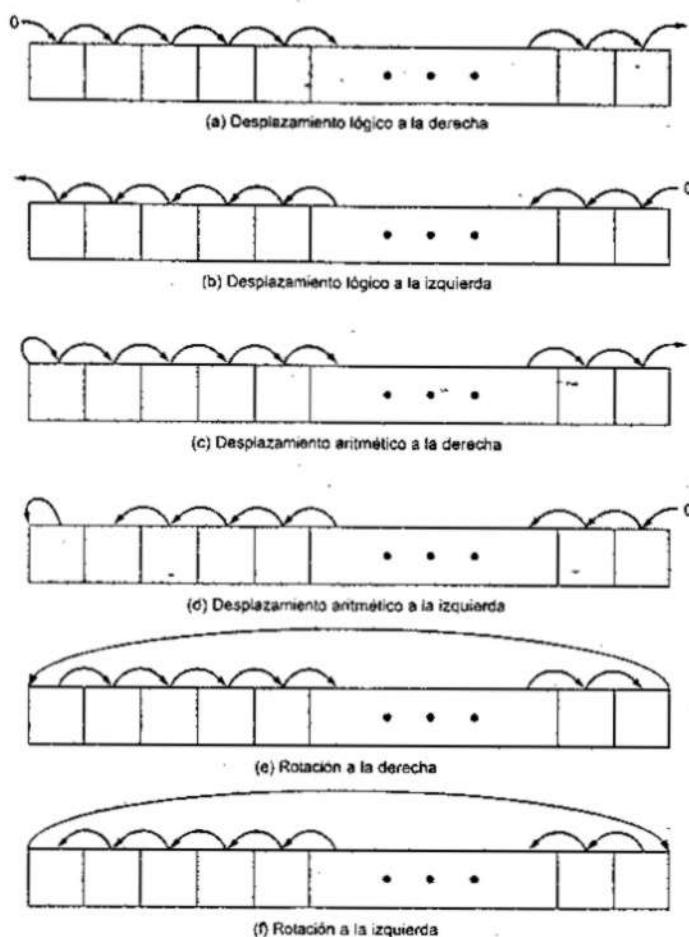


Figura 9.5. Operaciones de desplazamiento y de rotación.

debemos *desempaquetar* los caracteres antes de enviarlos. Para enviar los 2 caracteres de una palabra, procedemos como sigue:

1. Cargar la palabra en un registro.
2. Realizar la AND con el valor 1111111000000000. Esto enmascara el carácter de la derecha.
3. Desplazar el registro ocho veces. Esto desplaza el otro carácter a la mitad derecha del registro.
4. Realizar la E/S. El módulo de E/S leerá los 8 bits de orden más bajo del bus de datos.

Los pasos anteriores (el 2 puede en realidad suprimirse) producen el envío del carácter de izquierda. Para enviar el carácter de la derecha:

1. Cargar de nuevo la palabra en el registro.
2. AND con 000000011111111.
3. Realizar la E/S.

La operación de *desplazamiento aritmético* trata el dato como entero con signo, y no desplaza el bit de signo. En un desplazamiento aritmético a la derecha, el bit de signo normalmente se replica en la posición de bit de su derecha. Estos desplazamientos pueden acelerar ciertas operaciones aritméticas. Con números en notación de complemento a dos, un desplazamiento a izquierda o a derecha equivale a multiplicar o a dividir por 2 respectivamente, asumiendo que no hay desbordamiento ni desbordamiento a cero.

La *rotación*, o desplazamiento cíclico, preserva todos los bits con los que se está operando. Un posible uso de la rotación es ir volteando sucesivamente cada bit en la posición más a la izquierda, donde pueda ser identificado comprobando el bit de signo del dato (tratado éste como número).

Como en el caso de operaciones aritméticas, las operaciones lógicas implican actividad de la ALU y pueden involucrar operaciones de transferencia de datos.

## CONVERSIÓN

Las instrucciones de conversión son aquellas que cambian el formato u operan sobre el formato de los datos. Un ejemplo es la conversión decimal a binario.

Un ejemplo de instrucción de conversión más compleja es la Translate (TR) del S/370. Esta instrucción puede utilizarse para convertir de un código de 8 bits a otro, y tiene tres operandos:

TR R1, R2, L

El operando R2 contiene la dirección de comienzo de una tabla de códigos de 8 bits. Se traducen los L bytes que comienzan en la dirección especificada en R1, sustituyéndose cada byte por el contenido del elemento de la tabla indexado por dicho byte. Por ejemplo, para traducir de EBCDIC a ASCII, primero creamos una tabla de 256 bytes en posiciones de memoria (por ejemplo, desde la 1000 a la 10FF). Esta tabla debe contener los caracteres del código ASCII en la secuencia de la representación binaria del código EBCDIC; es decir, el código de cada carácter ASCII se coloca en la tabla en una posición relativa igual al valor binario del código EBCDIC del mismo carácter. Así pues, las posiciones 10F0 a 10F9 contendrán los valores 30 a 39, ya que F0 es el código EBCDIC del dígito 0, cuyo código ASCII es el 30; y así hasta la posición correspondiente al dígito 9. Suponga ahora que tenemos la secuencia de dígitos 1984, en EBCDIC, a partir de la posición 2100, y queremos convertirla a ASCII. Suponga lo siguiente:

- Las posiciones 2100-2103 contienen F1 F9 F8 F4.
- R1 contiene 2100.
- R2 contiene 1000.

Entonces, si ejecutamos:

TR R1, R2, 4

los contenidos de las posiciones 2100 a 2103 serían 31 39 38 34.

## ENTRADA/SALIDA

Las instrucciones de entrada/salida se trataron con cierto detalle en el Capítulo 6. Como vimos, existen aproximaciones muy diversas, incluyendo entradas/salidas programadas aisladas, entradas/salidas programadas asignadas en memoria, DMA, y el uso de un procesador de E/S. Muchas implementaciones ofrecen sólo unas pocas instrucciones de E/S, con acciones específicas indicadas mediante parámetros, códigos o palabras de órdenes.

## CONTROL DEL SISTEMA

Las instrucciones de control del sistema son, por lo general, instrucciones privilegiadas que pueden ejecutarse sólo mientras el procesador está en un estado privilegiado concreto o está ejecutando un programa de una zona privilegiada específica de memoria. Normalmente, estas instrucciones están reservadas para que las use el sistema operativo.

Algunos ejemplos de operaciones de control son los siguientes: una instrucción de control del sistema puede leer o alterar un registro de control (los registros de control se verán en el Capítulo 11); otro ejemplo es una instrucción para leer o modificar una clave de protección de memoria, tal como la utilizada en el sistema de memoria del S/370; otro ejemplo es acceder a bloques de control de procesos en un sistema con multiprogramación.

## CONTROL DE FLUJO

En todos los tipos de operaciones discutidos hasta ahora, la siguiente instrucción a ejecutar es la inmediatamente posterior, en memoria, a la instrucción en curso. Sin embargo, una fracción significativa de las instrucciones de cualquier programa tiene como misión cambiar la secuencia de ejecución de instrucciones. Para estas instrucciones, la operación que realiza la CPU es actualizar el contador de programa para que contenga la dirección de alguna de las instrucciones que hay en memoria.

Hay varias razones por las que se necesitan las operaciones de control de flujo o de transferencia del control. Entre las más importantes están:

1. En el uso práctico de los computadores es esencial poder ejecutar cada instrucción más de una vez, y puede que muchos miles de veces. Implementar una aplicación puede de requerir miles, o incluso millones, de instrucciones. Esto sería impensable si hubiera que escribir cada instrucción por separado. Si se va a procesar una tabla o una lista de elementos, lo normal es utilizar un bucle de programa. Así, una secuencia de instrucciones se ejecutaría repetidas veces para procesar todos los datos.
2. Prácticamente todos los programas implican algunas tomas de decisiones. Queremos que el computador haga algo cuando se cumple una condición, y otra cosa distinta si se cumple otra condición. Por ejemplo, una secuencia de instrucciones calcula la raíz cuadrada de un número. Al principio de la secuencia se comprueba el signo del número. Si éste es negativo, el cálculo no se realiza, sino que se señala una condición de error.
3. Redactar correctamente un programa largo, o incluso uno de extensión moderada, es una tarea excesivamente compleja. Es de gran ayuda partir la tarea en trozos más pequeños, con los que se trabaje por separado.

Retomamos la discusión sobre las operaciones de control de flujo que se pueden encontrar en repertorios de instrucciones: bifurcación, salto implícito y llamada a procedimiento.

### Instrucciones de bifurcación

Una instrucción de bifurcación, también llamada «de salto», tiene como uno de sus operandos la dirección de la siguiente instrucción a ejecutar. Las más frecuentes son las instrucciones de *salto condicional*. Es decir, se efectúa la bifurcación (se actualiza el contador de programa con la dirección especificada en el operando) sólo si se cumple una condición dada. En caso contrario, se ejecuta la instrucción siguiente de la secuencia (se incrementa el contador de programa de la forma habitual).

Hay dos formas comunes de generación de la condición a comprobar en una instrucción de salto condicional. En primer lugar, la mayoría de las máquinas proporcionan un código de condición de uno o varios bits, que se actualiza cuando se ejecutan algunas operaciones. Este código puede imaginarse como un pequeño registro visible para el usuario. Como ejemplo, una operación aritmética (ADD, SUBTRACT, etc.) podría fijar un código de condición de 2 bits con uno de los cuatro valores siguientes: 0, positivo, negativo o desbordamiento. En tal caso, la máquina podría tener cuatro instrucciones de bifurcación o salto condicional:

- BRP X Saltar a la posición X si el resultado es positivo
- BRN X Saltar a la posición X si el resultado es negativo
- BRZ X Saltar a la posición X si el resultado es cero
- BRO X Saltar a la posición X si se ha producido desbordamiento

En todos los casos anteriores, el resultado al que se hace referencia es el de la última operación ejecutada que afecte al código de condición.

Otra aproximación que puede utilizarse con un formato de instrucción de tres direcciones consiste en realizar la comparación y especificar la bifurcación en la misma instrucción. Por ejemplo:

BRE R1, R2, X Saltar a X si el contenido de R1 es igual al de R2

La Figura 9.6 muestra ejemplos de estas operaciones. Observe que un salto puede ser bien *hacia adelante* (a una instrucción con dirección más alta) o *hacia atrás* (dirección más baja). El ejemplo muestra cómo se pueden utilizar un salto incondicional y otro condicional para crear un bucle de repetición de instrucciones. Las instrucciones en las posiciones 202 a 210 se ejecutarán repetidas veces, hasta que el resultado de restar Y de X sea 0.

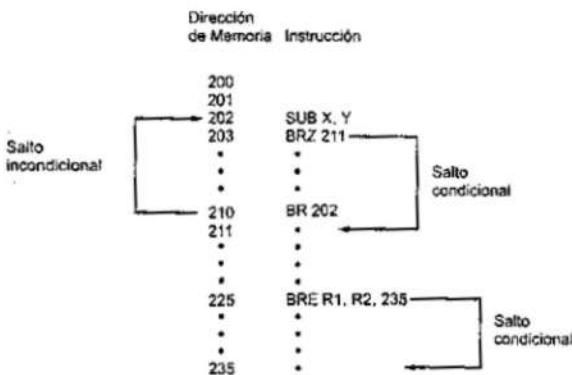


Figura 9.6. Instrucciones de bifurcación.

### Instrucciones de salto implícito

Otra forma común de instrucciones de control de flujo es la instrucción de salto implícito («skip»). Esta instrucción incluye una dirección de manera implícita. Normalmente, el salto implícito implica que se va a saltar una instrucción; por lo tanto, la dirección implícita es igual a la dirección de la siguiente instrucción más la longitud de una instrucción.

Dado que la instrucción de salto implícito no requiere un campo de dirección de destino, éste queda libre para otras cosas. Un ejemplo típico es la instrucción «incrementar y saltar si es cero» (ISZ). Considere el fragmento de programa siguiente:

```

301
.
.
.
309 ISZ R1
310 BR 301
311

```

En él, las dos instrucciones de control de flujo se emplean para implementar un bucle iterativo. R1 se fija a un valor negativo, el opuesto del número de iteraciones a realizar. Al final del bucle, R1 se incrementa. Si es distinto de cero, el programa bifurca hacia el comienzo del bucle. Si no, se salta la instrucción de bifurcación y el programa continúa con la instrucción siguiente a la de fin del bucle.

### Instrucciones de llamada a procedimiento

El *procedimiento* (procedure) fue quizás la innovación más importante en el desarrollo de los lenguajes de programación. Un procedimiento es un programa autoconsistente que se incorpora en un programa más grande. En cualquier punto del programa se puede invocar o *llamar* al procedimiento. Es decir, en ese punto se ordena al computador que pase a ejecutar el procedimiento completo y que retorne después al punto en que tuvo lugar la llamada.

Las dos razones principales para el uso de los procedimientos son la economía y la modularidad. Un procedimiento permite que la misma porción de código se utilice muchas veces. Esto es importante para economizar en esfuerzo de programación y para hacer un uso muy eficiente del espacio de memoria del sistema (el programa hay que almacenarlo). Los procedimientos permiten también que los programas largos se puedan subdividir en unidades más pequeñas. Este uso de la *modularidad* facilita enormemente la tarea de programar.

El uso de procedimientos requiere de dos instrucciones básicas: una instrucción de llamada («Call»), que produce una bifurcación desde la posición actual al procedimiento; y una instrucción de retorno del procedimiento (Return) al lugar desde el que se llamó. Ambas son modalidades de instrucciones de bifurcación.

La Figura 9.7a ilustra el uso de procedimientos para construir un programa. En este ejemplo, hay un programa principal que empieza en la posición 4000. Este programa incluye una llamada al procedimiento PROC1, que comienza en la posición 4500. Cuando se encuentra esta instrucción de llamada, la CPU interrumpe la ejecución del programa principal e inicia la ejecución de PROC1, captando la siguiente instrucción de la posición 4500. Dentro de PROC1, hay dos llamadas a PROC2, el cual comienza en la posición 4800. En cada caso, se interrumpe la ejecución de PROC1 y se ejecuta PROC2. La sentencia RETURN hace que la CPU vuelva al programa de llamada y continúe con la ejecución de la instrucción que sigue a la correspondiente CALL. Este comportamiento se ilustra en la Figura 9.7b.

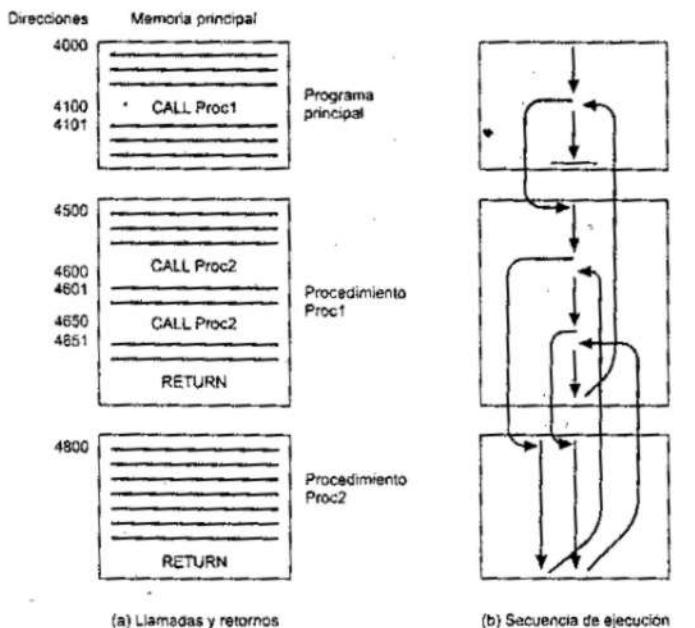


Figura 9.7. Procedimientos anidados.

Merece la pena resaltar varios puntos:

1. Un procedimiento puede llamarse desde distintas posiciones.
2. Un procedimiento puede contener llamadas a otros procedimientos. Esto posibilita el *anidamiento* de procedimientos hasta una profundidad arbitraria.
3. Cada llamada a procedimiento está emparejada con un retorno en el programa llamado.

Ya que debe permitirse que el procedimiento se llame desde distintos puntos, la CPU debe preservar la dirección de retorno en algún sitio, para que éste pueda realizarse correctamente. Hay tres lugares habituales para guardar la dirección de retorno:

- Un registro
- Al principio del procedimiento
- En la cabecera de la pila

Consideremos la instrucción en lenguaje máquina `CALL X`, que significa *llamada al procedimiento de la posición X*. Si se utiliza un registro, la ejecución de `CALL X` produce las siguientes acciones:

$$\begin{aligned} RN &\leftarrow PC + \Delta \\ PC &\leftarrow X \end{aligned}$$

donde RN es un registro que se utiliza siempre para este fin. PC es el contador de programa y  $\Delta$  es la longitud de la instrucción. El procedimiento llamado puede ahora consultar el contenido de RN para utilizarlo en el retorno posterior.

Una segunda posibilidad es almacenar la dirección de retorno al comienzo del procedimiento. En este caso, CALL X hace que:

$$\begin{aligned} X &\leftarrow PC + \Delta \\ PC &\leftarrow X + 1 \end{aligned}$$

La dirección de retorno queda almacenada en un lugar seguro.

Las dos aproximaciones anteriores son correctas y se han utilizado en la práctica. La única limitación que presentan es que impiden el uso de procedimientos *reentrantes*. Un procedimiento reentrant es aquel para el que es posible tener abiertas varias llamadas al mismo tiempo. Los procedimientos recursivos son un ejemplo del uso de esta característica.

Una aproximación más general y potente es utilizar una pila (véase el Apéndice 9A para una definición de la pila). Cuando la CPU ejecuta una llamada, coloca la dirección de retorno en la pila. Cuando ejecuta un retorno, utiliza la dirección almacenada en la pila. La Figura 9.8 ilustra la utilización de la pila.

Además de aportar la dirección de retorno, a menudo es necesario pasar o transferir ciertos parámetros en la llamada a un procedimiento. Estos se pueden transferir mediante registros. Otra posibilidad es almacenar estos parámetros en memoria justo después de la instrucción CALL. En este caso, el retorno debe hacerse a la posición siguiente a los parámetros. De nuevo, estas dos aproximaciones presentan limitaciones. Si se emplean los registros, el programa llamado y el que hace la llamada deben escribirse de manera que se asegure el uso correcto de los registros. El almacenamiento de los parámetros en memoria hace difícil transferir un número variable de ellos. Y ambas alternativas impiden el uso de procedimientos reentrantes.

Una alternativa más flexible para el paso de parámetros es la pila. Cuando el procesador ejecuta una llamada, no sólo introduce en la pila la dirección de retorno, sino también los parámetros que deben transferirse al procedimiento llamado. El procedimiento invocado puede acceder a los parámetros en la pila. Para el retorno, los parámetros de retorno pueden introducirse también en la pila. El conjunto de parámetros completo que se almacena en la llamada a un procedimiento, incluyendo la dirección de retorno, se denomina *marco de pila*.

En la Figura 9.9 se muestra un ejemplo. En este ejemplo se hace referencia al procedimiento P (en el que se declaran las variables  $x_1$  y  $x_2$ ) y al procedimiento Q, que puede ser llamado por P, y en el que se declaran las variables locales  $y_1$  e  $y_2$ . En la figura, el punto de

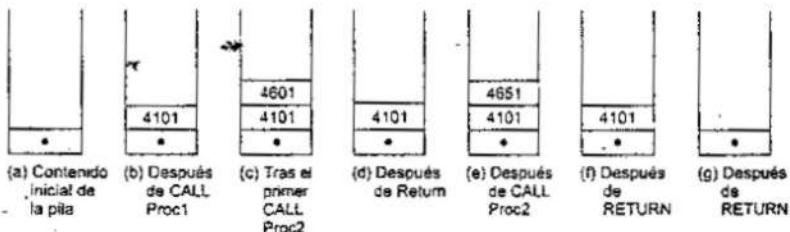


Figura 9.8. Uso de una pila para implementar el anidamiento de procedimientos de la Figura 9.7

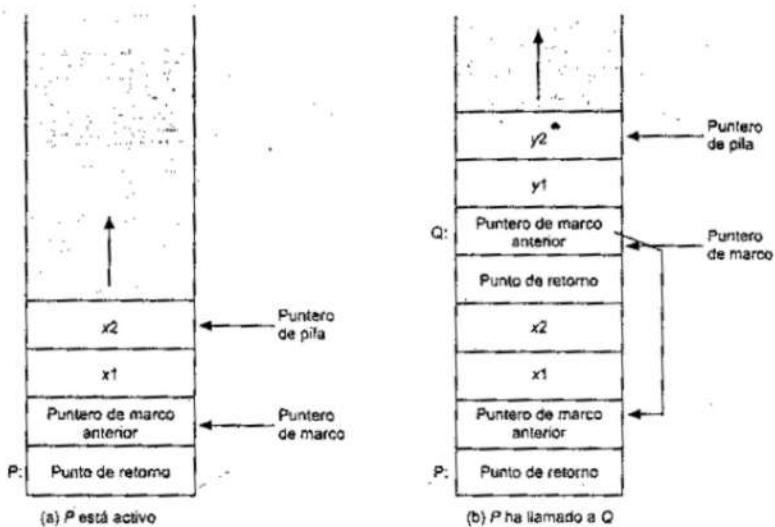


Figura 9.9. Crecimiento del marco de pila utilizando los procedimientos de ejemplo  $P$  y  $Q$  [DEWA90].

retorno para cada procedimiento está en el primer elemento memorizado del correspondiente marco de pila. A continuación, se almacena un puntero hacia el comienzo del marco precedente. Esto es necesario cuando el número o la longitud de los parámetros a introducir en la pila son variables.

### 3.5. TIPOS DE OPERACIONES EN EL PENTIUM II Y EL PowerPC

#### TIPOS DE OPERACIONES DEL PENTIUM II

El Pentium II ofrece un amplio abanico de tipos de operaciones, incluyendo diversas instrucciones especializadas. Con ello se ha intentado dotar de medios a los programadores de compiladores para producir traducciones óptimas a lenguaje máquina de los programas en lenguaje de alto nivel. La Tabla 9.7 resume los distintos tipos y da ejemplos de cada uno. La mayoría coinciden con instrucciones convencionales, que se pueden encontrar en la mayoría de los repertorios de instrucciones máquina, pero algunos de los tipos de instrucciones están adaptados a las arquitecturas 80x86/Pentium y son de particular interés.

#### Instrucciones de llamada/retorno

El Pentium II contiene cuatro instrucciones para ejecutar llamadas/retornos a/de procedimientos: CALL, ENTER, LEAVE y RETURN. Es instructivo analizar las posibilidades que ofrecen estas instrucciones. Recuerde (Figura 9.9) que una forma habitual de implemen-

Tabla 9.7. Tipos de operaciones del Pentium II (con ejemplos de operaciones típicas)

| Instrucción             | Descripción                                                                                                                                                                                                                                                                                          |
|-------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Transferencias de datos |                                                                                                                                                                                                                                                                                                      |
| MOV                     | Transferir el operando entre registros o entre registro y memoria.                                                                                                                                                                                                                                   |
| PUSH                    | Apilar el operando.                                                                                                                                                                                                                                                                                  |
| PUSHA                   | Apilar todos los registros.                                                                                                                                                                                                                                                                          |
| MOVSX                   | «Move byte, word, dword, sign extended». Transfiera un byte a una palabra, o una palabra a una palabra doble, extendiendo hacia la izquierda el signo según la representación en complemento a dos.                                                                                                  |
| LEA                     | «Load effective address». Carga el desplazamiento del operando fuente, en lugar de su valor, en el operando destino.                                                                                                                                                                                 |
| XLAT                    | «Table lookup translation». Sustituye un byte de AL con otro de una tabla de traducción codificada por el usuario. Cuando se ejecuta XLAT, AL debe tener un índice sin signo de la tabla. XLAT cambia el índice que contiene AL por el correspondiente elemento de la tabla.                         |
| IN, OUT                 | Entrada o salida de operando desde el (o al) espacio de E/S.                                                                                                                                                                                                                                         |
| Aritméticas             |                                                                                                                                                                                                                                                                                                      |
| ADD                     | Sumar los operandos.                                                                                                                                                                                                                                                                                 |
| SUB                     | Restar los operandos.                                                                                                                                                                                                                                                                                |
| MUL                     | Multiplicación de enteros sin signo, con operandos de un byte, una palabra o una palabra doble, y como resultado una palabra, una palabra doble o una cuádruple.                                                                                                                                     |
| IDIV                    | División con signo.                                                                                                                                                                                                                                                                                  |
| Lógicas                 |                                                                                                                                                                                                                                                                                                      |
| AND                     | Producto lógico de los operandos.                                                                                                                                                                                                                                                                    |
| BTS                     | «Bit test and set». Opera con un operando de campo de bits. La instrucción copia el valor actual de un bit en un indicador CF y pone a 1 el bit original.                                                                                                                                            |
| BSF                     | «Bit scan forward». Busca en una palabra o en una palabra doble el primer bit a 1 y almacena su número de posición en un registro.                                                                                                                                                                   |
| SHL/SHR                 | Desplazamiento lógico a izquierda o a derecha.                                                                                                                                                                                                                                                       |
| SAL/SAR                 | Desplazamiento aritmético a izquierda o a derecha.                                                                                                                                                                                                                                                   |
| ROL/ROR                 | Rotar a izquierda o a derecha.                                                                                                                                                                                                                                                                       |
| SETcc                   | Pone un byte a cero o a uno, dependiendo de alguna de las 16 condiciones definidas por los indicadores de estado.                                                                                                                                                                                    |
| Control de flujo        |                                                                                                                                                                                                                                                                                                      |
| JMP                     | Salto incondicional.                                                                                                                                                                                                                                                                                 |
| CALL                    | Transfiere el control a otra posición. Antes de transferirlo, se introduce en la pila la dirección de la instrucción siguiente a la CALL.                                                                                                                                                            |
| JE/JZ                   | Salto si igual/cero.                                                                                                                                                                                                                                                                                 |
| LOOP/LOOPZ              | Bucle si igual/cero. Se trata de una bifurcación condicional que utiliza un valor almacenado en el registro ECX. La instrucción primero decremente ECX antes de comprobar ECX para la condición del salto.                                                                                           |
| INT/INTO                | Interrupción/interrupción-si-desbordamiento. Transfiere el control a una rutina de servicio de interrupciones.                                                                                                                                                                                       |
| Operaciones con cadenas |                                                                                                                                                                                                                                                                                                      |
| MOVS                    | «Move byte, word, dword string». Esta instrucción opera con un elemento de una cadena, indexada por los registros ESI y EDI. Después de cada operación con un elemento de la cadena, dichos registros se incrementan o decrementan automáticamente, para apuntar al siguiente elemento de la cadena. |
| LODS                    | Carga un byte, una palabra o una palabra doble de una cadena.                                                                                                                                                                                                                                        |

Tabla 9.8. Códigos de condición del Pentium II

| Bit de estado | Nombre           | Descripción                                                                                                                                                                                  |
|---------------|------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| C             | Acarreo          | Indica un acarreo o un acarreo negativo en la posición de bit más significativa, tras una operación aritmética. También se modifica por algunas operaciones de desplazamiento y de rotación. |
| P             | Paridad          | Paridad del resultado de una operación aritmética o lógica. El 1 indica paridad par, y el 0 paridad impar.                                                                                   |
| A             | Acarreo auxiliar | Representa un acarreo o un acarreo negativo entre las dos mitades de una operación aritmética o lógica de 8 bits utilizando el registro AL.                                                  |
| Z             | Cero             | Indica que el resultado de una operación aritmética o lógica es 0.                                                                                                                           |
| S             | Signo            | Indica el signo del resultado de una operación aritmética o lógica.                                                                                                                          |
| O             | Desbordamiento   | Indica un desbordamiento aritmético después de una suma o una resta.                                                                                                                         |

( $-1 < 0$ ). Muchos lenguajes ensamblador introducen dos denominaciones diferentes, que permiten distinguir entre los dos casos anteriores: Si se están comparando dos números como enteros con signo, se emplean los términos *menor que* y *mayor que*; si se comparan como enteros sin signo, se emplean los términos *inferior* y *superior*.

Una segunda observación afecta a la complejidad de comparar enteros con signo. Un resultado con signo es mayor o igual que cero si (a) el bit de signo es cero y no ha habido

Tabla 9.9. Condiciones de bifurcación para las instrucciones de salto condicional y SETcc del Pentium II

| Símbolo    | Condición comprobada                                                                       | Comentario                                                         |
|------------|--------------------------------------------------------------------------------------------|--------------------------------------------------------------------|
| A, NBE     | $C = 0 \text{ AND } Z = 0$                                                                 | Superior; no inferior o igual (en números con signo: mayor que).   |
| AE, NB, NC | $C = 0$                                                                                    | Superior o igual; no inferior (mayor que o igual); no hay acarreo. |
| B, NAE, C  | $C = 1$                                                                                    | Inferior; no superior o igual (menor que); hay acarreo.            |
| BE, NA     | $C = 1 \text{ OR } Z = 1$                                                                  | Inferior o igual; no superior (menor que o igual).                 |
| E, Z       | $Z = 1$                                                                                    | Igual; cero (para números con o sin signo).                        |
| G, NLE     | $((S = 1 \text{ AND } O = 1) \text{ OR } (S = 0 \text{ AND } O = 0)) \text{ AND } (Z = 0)$ | Mayor que; no menor que o igual (con signo).                       |
| GE, NL     | $(S = 1 \text{ AND } O = 1) \text{ OR } (S = 0 \text{ AND } O = 0)$                        | Mayor que o igual; no menor que (con signo).                       |
| L, NGE     | $(S = 1 \text{ AND } O = 0) \text{ OR } (S = 0 \text{ AND } O = 1)$                        | Menor que; no mayor que o igual (con signo).                       |
| LE, NG     | $(S = 1 \text{ AND } O = 0) \text{ OR } (S = 0 \text{ AND } O = 1) \text{ OR } (Z = 1)$    | Menor que o igual; no mayor que (con signo).                       |
| NE, NZ     | $Z \neq 0$                                                                                 | Distinto; distinto de cero (con o sin signo).                      |
| NO         | $O \neq 0$                                                                                 | No desbordamiento.                                                 |
| NS         | $S \neq 0$                                                                                 | No signo (no negativo).                                            |
| NP, PO     | $P = 0$                                                                                    | No paridad; paridad impar.                                         |
| O          | $O \neq 1$                                                                                 | Desbordamiento.                                                    |
| P          | $P = 1$                                                                                    | Paridad; paridad par.                                              |
| S          | $S = 1$                                                                                    | Signo (negativo).                                                  |

desbordamiento ( $S = 0$  AND  $O = 0$ ), o (b) el bit de signo vale uno y ha habido desbordamiento. Un análisis de la Figura 8.5 le convencerá de que son correctas las condiciones que se comprueban para las distintas operaciones (véase el Problema 9.12).

### Instrucciones MMX del Pentium II

Intel introdujo en 1996 la tecnología MMX en su línea de procesadores Pentium. MMX comprende instrucciones muy optimizadas para tareas multimedia. Hay 57 instrucciones nuevas que tratan los datos en un modo SIMD (single-instruction, multiple-data), es decir una secuencia de instrucciones y múltiples secuencias de datos, que posibilita efectuar la misma operación, tal como una suma o una multiplicación, con varios elementos de datos a la vez. Cada instrucción suele ejecutarse en un ciclo de reloj. Para ciertas aplicaciones, estas operaciones rápidas en paralelo pueden conducir a una velocidad entre dos y ocho veces superior a la de algoritmos equiparables que no utilicen las instrucciones MMX [ATKI96].

El conjunto de instrucciones MMX está orientado a programación multimedia. Los datos de vídeo y audio suelen representarse mediante vectores o matrices grandes, compuestos por datos de longitud reducida (8 o 16 bits), mientras que las instrucciones convencionales operan normalmente con datos de 32 o 64 bits. Estos son algunos ejemplos: En gráficos y en video, cada escena consiste en una matriz de puntos de imagen<sup>1</sup>, y hay 8 bits para cada punto de imagen, u 8 bits para cada componente de color (rojo, verde, azul) del punto de imagen. Las muestras de audio suelen estar cuantizadas con 16 bits. Para algunos algoritmos de gráficos 3D es común emplear 32 bits para los tipos de datos básicos. Para posibilitar el procesamiento paralelo con estos tamaños de datos, en MMX se definen tres nuevos tipos de datos. Tienen una longitud de 64 bits, y constan de varios campos de datos más pequeños, cada uno de los cuales contiene un entero en coma fija. Estos tipos son los siguientes:

- **Byte empaquetado:** Ocho bytes en una cantidad de 64 bits.
- **Palabra empaquetada:** Cuatro palabras de 16 bits empaquetadas en 64 bits.
- **Palabra doble empaquetada:** Dos palabras dobles de 32 bits empaquetadas en 64 bits.

La Tabla 9.10 resumé el repertorio de instrucciones MMX. La mayoría de las instrucciones produce un procesamiento paralelo de bytes, palabras o palabras dobles. Por ejemplo, la instrucción PSLLW efectúa un desplazamiento lógico hacia la izquierda de cada una de las cuatro palabras del operando (una palabra empaquetada); la instrucción PADDB toma como operando un byte empaquetado, y realiza en paralelo sumas con cada posición de byte para producir un byte empaquetado de salida.

Una característica inusual que presenta el nuevo conjunto de instrucciones es la introducción de la aritmética con saturación. Con la aritmética sin signo ordinaria, cuando una operación produce un desbordamiento (es decir, se produce un acarreo en la posición de bit más significativa), el bit extra se trunca. Este tipo de truncamiento se conoce con el término inglés «wraparound», y puede, por ejemplo, hacer que el resultado de una suma sea menor que los dos operandos de entrada. Considere, por ejemplo, las dos palabras, en hexadecimal, F000h y 3000h. Su suma se expresaría como:

$$\begin{array}{r}
 \text{F000h} = 1111 0000 0000 0000 \\
 +3000h = \underline{\underline{1111}} 0000 0000 0000 \\
 \hline
 10010 0000 0000 0000 = 2000h
 \end{array>$$

<sup>1</sup> Un pixel o punto de imagen es el elemento más pequeño de una imagen digital al que se puede asignar un nivel de gris. En otras palabras, un pixel es cada punto individual de una imagen representada mediante una matriz de puntos.

Tabla 9.10. Repertorio de instrucciones MMX

| Categoría              | Instrucción          | Descripción                                                                                                                                                                            |
|------------------------|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Aritméticas            | PADD [B, W, D]       | Suma paralela de ocho bytes empaquetados, cuatro palabras de 16 bits, o dos palabras dobles de 32 bits, con truncamiento.                                                              |
|                        | PADDS [B, W]         | Suma con saturación.                                                                                                                                                                   |
|                        | PADDUS [B, W]        | Suma sin signo con saturación.                                                                                                                                                         |
|                        | PSUB [B, W, D]       | Resta con truncamiento.                                                                                                                                                                |
|                        | PSUBS [B, W]         | Resta con saturación.                                                                                                                                                                  |
|                        | PSUBUS [B, W]        | Resta sin signo con saturación.                                                                                                                                                        |
|                        | PMULHW               | Multiplicación paralela de cuatro palabras con signo de 16 bits, con elección de los 16 bits más significativos del resultado de 32 bits.                                              |
|                        | PMULLW               | Multiplicación paralela de cuatro palabras con signo de 16 bits, con elección de los 16 bits menos significativos del resultado de 32 bits.                                            |
| Comparación            | PMADDWD              | Multiplicación paralela de cuatro palabras con signo de 16 bits; suma a la vez pares adyacentes de los resultados de 32 bits.                                                          |
|                        | PCMPEQ [B, W, D]     | Comparación paralela de igualdad; el resultado es una máscara de unos, cuando es verdadero, o una máscara de ceros, si es falso.                                                       |
| Conversión             | PCMPGT [B, W, D]     | Comparación paralela de magnitud; mayor que; el resultado es una máscara de unos, cuando es verdadero, o una máscara de ceros, si es falso.                                            |
|                        | PACKUSWB             | Empaqueta palabras en bytes con saturación sin signo.                                                                                                                                  |
|                        | PACKSS [WB, DW]      | Empaqueta palabras en bytes, o palabras dobles en palabras, con saturación con signo.                                                                                                  |
|                        | PUNPCKH [BW, WD, DQ] | Desempaquetar en paralelo (mezcla entrelazada) los bytes más significativos, palabras, o dobles palabras, del registro MMX.                                                            |
| Lógicas                | PUNPCKL [BW, WD, DQ] | Desempaquetar en paralelo (mezcla entrelazada) los bytes menos significativos, palabras, o dobles palabras, del registro MMX.                                                          |
|                        | PAND                 | Producto lógico (AND bit a bit) de 64 bits.                                                                                                                                            |
|                        | PNDN                 | Producto lógico y complemento (NAND bit a bit) de 64 bits.                                                                                                                             |
|                        | POR                  | Suma lógica (OR bit a bit) de 64 bits.                                                                                                                                                 |
| desplazamiento         | PXOR                 | OR exclusiva (XOR bit a bit) de 64 bits.                                                                                                                                               |
|                        | PSLL [W, D, Q]       | Desplazamiento lógico a la izquierda en paralelo de palabras, palabras dobles, o palabra cuádruple, tantas veces como se especifique en el registro MMX o mediante un valor inmediato. |
|                        | PSRL [W, D, Q]       | Desplazamiento lógico a la derecha en paralelo de palabras, palabras dobles, o palabra cuádruple.                                                                                      |
| Transferencia de datos | PSRA [W, D]          | Desplazamiento lógico a la derecha en paralelo de palabras, palabras dobles, o palabra cuádruple.                                                                                      |
|                        | MOV [D, Q]           | Transfiere una palabra doble o una cuádruple a/desde el registro MMX.                                                                                                                  |
| gestión de estado      | EMMS                 | Descarga el estado MMX (descarga los bits de etiqueta de los registros FP).                                                                                                            |

ta: Para aquellas instrucciones que admiten varios tipos de datos, éstos se indican entre corchetes: [byte (B), palabra (W), palabra doble (D), palabra cuádruple (Q)].

Si los dos números representaban intensidad de imagen, el resultado de la suma hace que la combinación de zonas sombreadas oscuras aparezca como más clara. Esto no es lo que se pretende normalmente. Mediante la aritmética con saturación, cuando la suma produce un desbordamiento, o la resta produce un desbordamiento negativo, el resultado se fija respectivamente al mayor o al menor valor representable. Para el ejemplo anterior, la aritmética con saturación daría como resultado:

$$\begin{array}{r}
 F000h = 1111\ 0000\ 0000\ 0000 \\
 +3000h = \underline{1111}\ 0000\ 0000\ 0000 \\
 \hline
 10010\ 0000\ 0000\ 0000 \\
 1111\ 1111\ 1111\ 1111 = FFFFh
 \end{array}$$

Para apreciar el uso de las instrucciones MMX, vamos a considerar un ejemplo tomado de [PELE97] e [INTE98b]. Una función típica en video es el efecto de desvanecimiento o extinción progresiva («fade-out») y reaparición («fade-in»), mediante el cual una imagen A se deshace y convierte gradualmente en otra B. Las dos imágenes se combinan mediante una media ponderada:

$$\text{Pixel\_resultado} = \text{Pixel\_A} \times \text{fade} + \text{Pixel\_B} \times (1 - \text{fade})$$

Este cálculo se efectúa para cada posición de punto de imagen en A y B. Si se produce una secuencia de video mientras «fade» está cambiando progresivamente desde 1 a 0 (con una escala ajustada a un entero de 8 bits), el resultado es una transformación paulatina de la imagen A en la imagen B.

La Figura 9.10 muestra, para un conjunto de puntos de imagen, la secuencia de pasos necesaria. Las componentes de pixel de 8 bits son transformadas en elementos de 16 bits para adaptarlas al tamaño de las multiplicaciones MMX de 16 bits. Si estas imágenes tienen una resolución de  $640 \times 480$ , y en el cambio de imagen se emplean los 255 valores posibles de «fade», se ejecutarían 535 millones de instrucciones cuando se emplea MMX. El mismo procesamiento realizado sin las instrucciones MMX requeriría 1.4 millones de instrucciones [INTE98b].

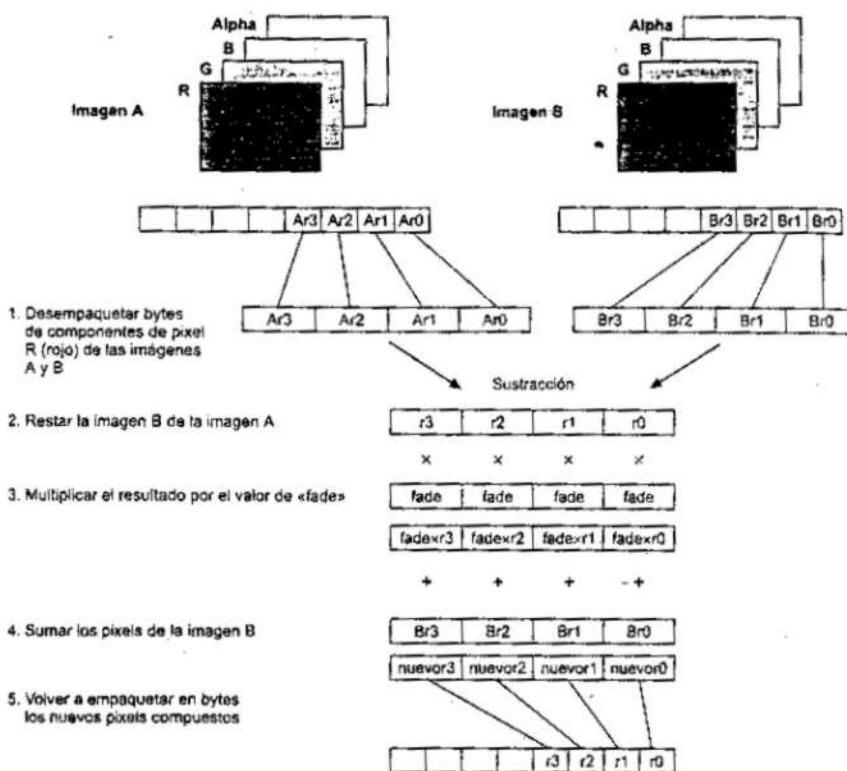
## TIPOS DE OPERACIONES DEL PowerPC

El PowerPC ofrece una gran variedad de tipos de operaciones. La Tabla 9.11 los enumera y proporciona algunos ejemplos. Merece la pena mencionar varias características.

### Instrucciones dedicadas a bifurcaciones

En el PowerPC se contemplan las posibilidades usuales de bifurcación condicional e incondicional. Las instrucciones de bifurcación comprueban un solo bit del registro de condición para determinar un estado de verdadero, falso o indiferencia, y el contenido de un registro de cuenta para comprobar condiciones de cero, distinto de cero o indiferencia. Así pues, en instrucciones de bifurcación condicional se admiten nueve posibles condiciones. Cuando se comprueba el estado de cero o distinto de cero del registro de cuenta, antes se decrementa en 1 su valor. Esto es conveniente al objeto de implementar bucles iterativos.

Las instrucciones de bifurcación pueden también indicar que la dirección de la posición siguiente a la bifurcación se guarde en el registro de enlace, descrito en el Capítulo 13. Esto facilita el procesamiento de llamadas/retornos.



Código MMX que ejecuta las operaciones anteriores:

|          |              |                                                             |
|----------|--------------|-------------------------------------------------------------|
| pxor     | mm7, mm7     | :poner a cero mm7                                           |
| movq     | mm3, fad_val | :cargar el valor de fade replicado 4 veces                  |
| movd     | mm0, imagenA | :cargar las componentes de rojo de 4 pixeles de la imagen A |
| movd     | mm1, imagenB | :cargar las componentes de rojo de 4 pixeles de la imagen B |
| punpkbhw | mm0, mm7     | :desempaquetar a 16 bits 4 pixeles                          |
| punpkbhw | mm1, mm7     | :desempaquetar a 16 bits 4 pixeles                          |
| psubw    | mm0, mm1     | :restar la imagen B de la imagen A                          |
| pmulhw   | mm0, mm3     | :multiplicar los valores de la resta por el valor de «fade» |
| padddw   | mm0, mm1     | :sumar el resultado a la imagen B                           |
| packuswb | mm0, mm7     | :empaquetar en bytes los resultados de 16 bits              |

Figura 9.10. Composición de imagen en una representación de planos de colores (PELE97).

### Instrucciones de carga/memorización

En la arquitectura PowerPC, sólo acceden a memoria las instrucciones de carga y memorización; las instrucciones aritméticas y lógicas se realizan sólo con contenidos de registros. Esto es característico de un diseño RISC, y se verá con mayor detalle en el Capítulo 12.

Tabla 9.11. Tipos de operaciones del PowerPC (con ejemplos de operaciones típicas)

| Instrucción                                   | Descripción                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|-----------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Dedicadas a bifurcaciones</b>              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| b<br>bl<br>bc<br>sc<br>trap                   | Bifurcación o salto incondicional.<br>Saltar a la dirección destino y colocar en el registro de enlace la dirección efectiva de la instrucción siguiente a la bifurcación.<br>Salto condicional en función del registro de cuenta y/o bit del registro de condición.<br>Llamada al sistema para invocar un servicio del sistema operativo.<br>Comparar dos operandos y, si se cumple la condición especificada, invocar al gestor de interrupciones del sistema.                                                                                                                                                                                                                                                                                       |
| <b>Carga/memorización</b>                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| lwzu<br>ld<br>lmw<br>lswx                     | Cargar una palabra y poner a cero los restantes bits de la izquierda; actualizar el registro fuente.<br>Cargar una palabra doble.<br>Cargar una palabra múltiple; cargar palabras consecutivas en registros contiguos, desde el especificado hasta el registro 31 de uso general.<br>Cargar una cadena de bytes en registros, comenzando por el registro indicado; cuatro bytes por registro; y con conexión cíclica del registro 31 al 0.                                                                                                                                                                                                                                                                                                             |
| <b>Aritmética de enteros</b>                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| add<br>subf -<br>mullw<br>divd                | Sumar los contenidos de dos registros y ubicar el resultado en un tercero.<br>Restar los contenidos de dos registros y ubicar el resultado en un tercero.<br>Multiplicar los 32 bits menos significativos de los contenidos de dos registros y guardar el producto de 64 bits en un tercer registro.<br>Dividir los contenidos de 64 bits de dos registros, y guardar el cociente en un tercer registro.                                                                                                                                                                                                                                                                                                                                               |
| <b>Lógicas y desplazamientos</b>              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| cmp<br>crand<br>and<br>cntlzd<br>ridic<br>std | Comparar dos operandos y fijar cuatro bits de condición en el campo del registro de condición que se especifica.<br>AND del registro de condición: se calcula el producto lógico de dos bits del registro de condición y el resultado se guarda en una de las dos posiciones de bit.<br>Producto lógico de los contenidos de dos registros y guardar el resultado en un tercero.<br>Cuenta el número de bits a 0 consecutivos del registro fuente, empezando por el bit cero, y guarda el resultado de la cuenta en el registro destino.<br>Rotar a la izquierda un registro de palabra doble, realizar la AND con una máscara, y almacenar el registro destino.<br>Desplazar a la izquierda el registro fuente y almacenar en el registro de destino. |
| <b>Coma flotante</b>                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| fls<br>fadd<br>fmadd<br>fcmpu                 | Cargar de memoria un número en coma flotante de 32 bits, convertir al formato de 64 bits, y memorizar en un registro de coma flotante.<br>Sumar los contenidos de dos registros y ubicar el resultado en un tercero.<br>Multiplicar los contenidos de dos registros, sumar el contenido de un tercero, y colocar el resultado en un cuarto registro.<br>Comparar dos operandos en coma flotante y fijar los bits de condición.                                                                                                                                                                                                                                                                                                                         |
| <b>Gestión de cache</b>                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| dcbf<br>icbi                                  | Limpia un bloque de la cache de datos; busca en las direcciones concretas de la cache y realiza la operación.<br>Invalide un bloque de la cache de instrucciones.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |

Hay dos características que distinguen a las diferentes instrucciones de carga/memorización:

- **Tamaño del dato:** Los datos pueden transferirse en unidades de un byte, media palabra, palabra o palabra doble. También se dispone de instrucciones para cargar o memorizar una cadena de bytes en, o de, un conjunto registros.
- **Extensión del signo:** Cuando se carga una palabra o una media palabra, los bits no utilizados de la parte izquierda del registro de 64 bits de destino se llenan, bien con ceros, o bien con el valor del bit de signo del dato cargado.

### **9.6. LENGUAJE DE ENSAMBLADOR**

Una CPU puede interpretar y ejecutar instrucciones máquina. Estas instrucciones son, simplemente, números binarios almacenados en el computador. Si un programador quisiera programar directamente en lenguaje máquina, necesitaría introducir los programas como datos binarios.

Considere la sencilla sentencia BASIC:

$$N = I + J + K$$

Suponga que queremos programar esta sentencia en lenguaje máquina y dar a I, J y K los valores iniciales 2, 3 y 4, respectivamente. La forma de hacer esto se muestra en la Figura 9.11a. El programa empieza en la posición 101 (hexadecimal). Se reserva memoria para las cuatro variables a partir de la posición 201. El programa consta de cuatro instrucciones:

1. Cargar el contenido de la posición 201 en el acumulador (AC).
2. Sumar a AC el contenido de la posición 202.
3. Sumar a AC el contenido de la posición 203.
4. Memorizar el contenido de AC en la posición 204.

Esto es, claramente, un proceso tedioso y muy susceptible a errores.

Una ligera mejora consiste en redactar el programa en hexadecimal, en lugar de en binario (Figura 9.11b). El programa se escribiría mediante una serie de líneas, en la que cada línea contiene la dirección de una posición de memoria y el código hexadecimal del valor binario a memorizar en tal posición. Necesitamos, entonces, un programa que acepte esta entrada, traduzca cada línea a un número binario y lo almacene en la posición especificada.

Para que la mejora sea más significativa, podemos hacer uso de nombres simbólicos o nemotécnicos de las instrucciones, en lugar de los códigos de operación reales. El resultado es el *programa simbólico* mostrado en la Figura 9.11c. Cada línea sigue representando una posición de memoria, y consta de tres campos separados por espacios. El primer campo contiene la dirección de una posición. En cada instrucción, el segundo campo contiene el símbolo de tres letras que representa su código de operación. Si se trata de una instrucción que hace referencia a memoria, un tercer campo contiene la dirección. Para memorizar un dato concreto en una posición dada, nos inventamos una *pseudoinstrucción* con el símbolo DAT. Ésta es meramente un indicador de que el tercer campo de la línea contiene un número hexadecimal a memorizar en la posición que especifica el primer campo.

Para este tipo de descripción de entrada necesitamos un programa ligeramente más complejo. El programa aceptaría como entrada cada línea, generaría un número binario, basán-

| Dirección | Contenido |      |      |      |  |     |     |     |
|-----------|-----------|------|------|------|--|-----|-----|-----|
| 101       | 0010      | 0010 | 0000 | 0001 |  | 101 | LDA | 201 |
| 102       | 0001      | 0010 | 0000 | 0010 |  | 102 | ADD | 202 |
| 103       | 0001      | 0010 | 0000 | 0011 |  | 103 | ADD | 203 |
| 104       | 0011      | 0010 | 0000 | 0100 |  | 104 | STA | 204 |
| 201       | 0000      | 0000 | 0000 | 0010 |  | 201 | DAT | 2   |
| 202       | 0000      | 0000 | 0000 | 0011 |  | 202 | DAT | 3   |
| 203       | 0000      | 0000 | 0000 | 0100 |  | 203 | DAT | 4   |
| 204       | 0000      | 0000 | 0000 | 0000 |  | 204 | DAT | 0   |

|                         |                        |
|-------------------------|------------------------|
| (a) Programa en binario | (c) Programa simbólico |
|-------------------------|------------------------|

| Dirección | Contenido |   | Etiqueta | Operación | Operando |
|-----------|-----------|---|----------|-----------|----------|
| 101       | 2201      |   | FORMUL   | LDA       | I        |
| 102       | 1202      |   |          | ADD       | J        |
| 103       | 1203      |   |          | ADD       | K        |
| 104       | 3204      |   |          | STA       | N        |
| 201       | 0002      | I |          | DATA      | 2        |
| 202       | 0003      | J |          | DATA      | 3        |
| 203       | 0004      | K |          | DATA      | 4        |
| 204       | 0000      | N |          | DATA      | 0        |

(b) Programa en hexadecimal

(d) Programa en ensamblador

Figura 9.11. Cálculo de la fórmula  $N = I + J + K$ .

dose en los campos segundo y tercero (si lo hay), y lo memorizaría en la posición indicada por el primer campo.

El uso de programas simbólicos hace la vida mucho más fácil, pero es aún engorroso. En particular, hay que dar una dirección absoluta para cada palabra. Esto significa que el programa y los datos sólo pueden cargarse en posiciones concretas de memoria, y que debemos saber siempre cuáles son. Peor aún, supongamos que se quiere algún día cambiar el programa para añadir o borrar una línea. Esto cambiaría las direcciones de todas las palabras siguientes.

Un procedimiento mejor, y frecuentemente utilizado, es emplear direcciones simbólicas. Esto se ilustra en la Figura 9.11d. Cada línea sigue teniendo tres campos. El primero sigue siendo para la dirección, pero se utiliza un símbolo en lugar de una dirección numérica absoluta. Algunas líneas carecen de dirección, indicando que la dirección de dicha línea es uno más que la dirección de la precedente. Para las instrucciones que hacen referencia a memoria, el tercer campo contiene también una dirección simbólica.

Con este último refinamiento, hemos inventado un *lenguaje ensamblador*. Los programas escritos en lenguaje ensamblador (programas en ensamblador) se traducen a lenguaje máquina mediante un *ensamblador*. Este programa debe, no sólo realizar la traducción simbólica mencionada antes, sino también asignar direcciones de memoria a las direcciones simbólicas.

El desarrollo de los lenguajes ensambladores fue un logro importante en la evolución de la tecnología de los computadores. Fue el primer paso hacia los lenguajes de alto nivel utilizados hoy en día. Aunque son pocos los programadores que los utilicen, prácticamente todos los procesadores disponen de lenguajes ensambladores. Se utilizan para programas del sistema, tales como compiladores y rutinas de E.S.

**9.7. LECTURAS RECOMENDADAS**

Hay varios libros de texto que proporcionan una buena descripción de lenguajes máquina y diseño de repertorios de instrucciones; entre ellos están [HENN96], [TANE90] y [HAYE88]. El repertorio de instrucciones del Pentium está bien descrito en [BREY97] y [DEWA90]; y el del PowerPC en [IBM94] y [WEIS94].

BREY97 Brey, B. *The Intel Microprocessors: 8086/8066, 80186/80188, 80286, 80386, 80486, Pentium, and Pentium Processor*. Upper Saddle River, NJ: Prentice Hall, 1997.

DEWA90 Dewar, R., y Smosha, M. *Microprocessors: A Programmer's View*. New York: McGraw-Hill, 1990.

HAYE88 Hayes, J. *Computer Architecture and Organization*. 2nd edition. New York: McGraw-Hill, 1988.

HENN96 Hennessy, J., y Patterson, D. *Computer Architecture: A Quantitative Approach*. San Mateo, CA: Morgan Kaufmann, 1996.

IBM94 International Business Machines, Inc. *The PowerPC Architecture: A Specification for a New Family of RISC Processors*. San Francisco, CA: Morgan Kaufmann, 1994.

TANE90 Tanenbaum, A. *Structured Computer Organization*. Englewood Cliffs, NJ: Prentice Hall, 1990.

WEIS94 Weiss, S., y Smith, J. *POWER and PowerPC*. San Francisco: Morgan Kaufmann, 1994.

**9.8. PROBLEMAS**

- 9.1. Muchas CPU incluyen lógica para realizar operaciones aritméticas con números en decimal empaquetado. Aunque las reglas para la aritmética decimal son similares a las de las operaciones binarias, los resultados decimales pueden requerir algunas correcciones o ajustes de los dígitos individuales cuando se emplea lógica binaria.

Considere la suma en decimal de dos números sin signo. Si cada número consta de  $N$  dígitos, habrá  $4N$  bits en cada número. Los dos números se suman mediante un sumador binario. Sugiera una regla sencilla para corregir el resultado. Realice después la suma de los números 1.698 y 1.786.

- 9.2. El complemento a diez de un número decimal  $X$  se define como  $10N - X$ , donde  $N$  es el número de dígitos decimales del número. Describa el uso de la representación en complemento a diez para realizar la resta decimal. Ilustre el procedimiento restando  $(0326)_{10}$  de  $(0736)_{10}$ .
- 9.3. Compare las máquinas de cero, una, dos y tres direcciones, escribiendo programas que calculen

$$X = (A + B \times C) / (D - E \times F)$$

para cada una de las cuatro máquinas. Las instrucciones de que se dispone son:

| 0 Direcciones | 1 Dirección | 2 Direcciones   | 3 Direcciones   |
|---------------|-------------|-----------------|-----------------|
| PUSH M        | LOAD M      | MOVE (X ← Y)    | MOVE (X ← Y)    |
| POP M         | STORE M     |                 |                 |
| ADD           | ADD M       | ADD (X ← X + Y) | ADD (X ← Y + Z) |
| SUB           | SUB M       | SUB (X ← X - Y) | SUB (X ← Y - Z) |
| MUL           | MUL M       | MUL (X ← X × Y) | MUL (X ← Y × Z) |
| DIV           | DIV M       | DIV (X ← X/Y)   | DIV (X ← Y/Z)   |

- 9.4. Considere un computador hipotético con un repertorio de instrucciones formado por sólo dos instrucciones de  $n$  bits. El primer bit especifica el código de operación, y los bits restantes direccionan una de las  $2^n - 1$  palabras de  $n$  bits de la memoria principal. Las dos instrucciones son:

SUBS X Resta al acumulador el contenido de la posición X, y memoriza el resultado en la posición X y en el acumulador.

JUMP X Coloca la dirección X en el contador de programa.

Una palabra de memoria principal puede contener una instrucción, o un número binario en notación de complemento a dos. Demuestre que este repertorio de instrucciones es razonablemente completo, especificando cómo se podrían programar las siguientes operaciones:

- Transferencia de datos: Posición X al acumulador; acumulador a la posición X.
- Suma: suma el contenido de la posición X al acumulador.
- Operación lógica OR.
- Operaciones de E/S.

9.5. Muchos repertorios de instrucciones incluyen la instrucción NOOP (no operación), que no tiene otro efecto sobre el estado de la CPU que incrementar el contador de programa. Sugiera algunos usos de esta instrucción.

9.6. Suponga que la CPU va a utilizar una pila para gestionar las llamadas a procedimientos y los retornos. ¿Puede eliminarse el contador de programa utilizando como contador de programa la cabecera de la pila?

9.7. El Apéndice 9A puntualiza que no hay instrucciones específicas para la pila en un repertorio de instrucciones, si ésta va a ser utilizada por la CPU sólo para manejar procedimientos. ¿Cómo puede la CPU utilizar la pila para cualquier fin sin instrucciones específicas de gestión de la pila?

9.8. Convierta las siguientes fórmulas de notación polaca inversa a infija:

- AB + C + D
- AB/CD/+
- ABCDE + × × /
- ABCDE + F/ + G - H/ × +

9.9. Convierta las siguientes fórmulas de infija a polaca inversa:

- $A + B + C + D + E$
- $(A + B) \times (C + D) + E$
- $(A \times B) + (C \times D) + E$
- $(A - B) \times (((C - D) \times E)/F)/G) \times H$

9.10. Convierta la expresión  $A + B - C$  a notación postfija, utilizando el algoritmo de Dijkstra. Muestre los pasos seguidos. ¿Es el resultado equivalente a  $(A + B) - C$ , o a  $A + (B - C)$ ? ¿Importa?

9.11. La arquitectura Pentium II incluye una instrucción llamada «ajuste decimal tras la suma» (DAA). DAA realiza la siguiente secuencia de operaciones:

```
if ((AL AND 0FH) > 9) OR (AF = 1) then
 AL ← AL + 6;
 AF ← 1;
else
 AF ← 0;
endif;
if (AL > 9FH) OR (CF = 1) then
 AL ← AL + 60H;
 CF ← 1;
else
 AF ← 0;
endif.
```

«H» indica hexadecimal. AL es un registro de 8 bits que retiene el resultado de la suma de dos enteros sin signo de 8 bits. AF es un indicador que se pone a uno si hay un acarreo del bit 3 al 4 como resultado de la suma. CF es un indicador que se activa si hay un acarreo del bit 7 al 8. Explique la función realizada por la instrucción DAA.

9.12. La instrucción «compare» (CMP: comparar) del Pentium II resta el operando fuente del operando destino; actualiza los indicadores de estado (C, P, A, Z, S, O), pero no afecta a ninguno de los operandos. La instrucción CMP puede estar seguida de una instrucción de salto condicional Jump (Jcc) o por una SETcc (véase Tabla 9.7), donde cc hace referencia a una de las 16 condiciones listadas en la Tabla 9.9. Demuestre que son correctas las condiciones comprobadas para el caso de una comparación de números con signo.

9.13. La mayoría de los repertorios de instrucciones de microprocesadores incluyen una instrucción que comprueba una condición y, si se cumple, pone a uno un operando destino. Algunos ejemplos son la SETcc del Pentium II, la Scc del Motorola MC68000 y la Scond del National 32000.

a) Hay algunas diferencias entre estas instrucciones:

- SETcc y Scc actúan sobre un único byte, mientras que Scond lo hace sobre operandos de un byte, una palabra o una palabra doble.
- SETcc y Scond ponen el operando al valor entero 1 si se cumple la condición, y a cero si no. Scc pone todos los bits del byte a uno o a cero respectivamente.

Compare las ventajas y desventajas relativas de estas diferencias.

- b) Ninguna de estas instrucciones afecta a los indicadores de condición, y se requiere, por tanto, una comprobación explícita del resultado de la instrucción para determinar su valor. Discuta si los códigos de condición debieran activarse o no con esta instrucción.
- c) Una sencilla sentencia IF, como la IF a > b THEN, puede implementarse utilizando un método de representación numérica, es decir, haciendo evidente el valor booleano, en contraposición al método de *flujo del control*, que representa el valor de una expresión booleana por un punto alcanzado en el programa. Un compilador podría implementar IF a > b THEN con el siguiente código del 80×86:

```

SUB CX, CX :pone el registro CX a 0
MOV AX, B :transfiere el contenido de la posición B al registro AX
CMP AX, A :compara el contenido del registro AX y de la posición A
JLE TEST :saltar si A ≤ B
INC CX :suma 1 al contenido del registro CX
TEST JCXZ OUT :salta si el contenido de CX es 0
THEN
OUT

```

El resultado de ( $A > B$ ) es un valor booleano almacenado en un registro y disponible después, fuera del contexto del flujo del código mostrado más arriba. Es conveniente utilizar para ello el registro CX, porque muchos de los códigos de operación de bifurcación y de bucle incorporan la comprobación de CX.

Muestre una implementación alternativa utilizando la instrucción SETcc que ahorre memoria y tiempo de ejecución (Nota: no son necesarias otras instrucciones adicionales del 80×86, aparte de las SETcc).

- d) Considere ahora la sentencia en lenguaje de alto nivel:

$A := (B > C) \text{ OR } (D = F)$

Un compilador podría generar el siguiente código:

```

MOV EAX, B :transfiere el contenido de la posición B al registro EAX
CMP EAX, C :compara el contenido del registro EAX y de la posición C
MOV BL, 0 :0 representa falso
JLE N1 :salta si B ≤ C
MOV BL, 1 :1 representa falso
N1 MOV EAX, D
 CMP EAX, F
 MOV BH, 0
 JNE N2
 MOV BH, 1
N2 OR BL, BH

```

muestre una implementación alternativa utilizando la instrucción SETcc que ahorre memoria y tiempo de ejecución.

- 9.14. Utilizando el algoritmo para conversión de notación infix a postfija definido en el Apéndice 9A, muestre los pasos involucrados en la conversión a postfija de la expresión de la Figura 9.15. Utilice una representación similar a la empleada en la Figura 9.17.

- 9.15. Muestre el cálculo de la expresión de la Figura 9.17, empleando una representación similar a la Figura 9.16.
- 9.16. Redibuje el patrón «little-endian» de la Figura 9.18 de manera que los bytes aparezcan como si estuvieran numerados siguiendo el patrón «big-endian». Es decir, muestre la memoria en filas de 64 bits, con los bytes enumerados de izquierda a derecha y de arriba a abajo.
- 9.17. Utilizando el formato de la Figura 9.18, dibuje los patrones «little-endian» y «big-endian» para la siguiente estructura de datos, y comente los resultados.

```
a) struct {
 double i; //0x1112131415161718
} s1;
b) struct {
 int i; //0x11121314
 int j; //0x15161718
} s2;
c) struct {
 short i; //0x1112
 short j; //0x1314
 short k; //0x1516
 short l; //0x1718
} s3;
```

- 9.18. Las especificaciones de la arquitectura PowerPC no indican cómo se implementaría el modo «little-endian»; sólo indica cómo debe ver un procesador la memoria cuando se trabaja en modo «little-endian». Cuando se convierten estructuras de datos de «big-endian» a «little-endian», los procesadores son libres de implementar un mecanismo de intercambio de bytes verdadero o de utilizar algún mecanismo de modificación de direcciones. Los procesadores PowerPC actuales son por defecto máquinas «big-endian», y utilizan la modificación de direcciones para tratar los datos como «little-endian».

Considere la estructura definida en la Figura 9.18. El patrón de la parte inferior derecha de la figura muestra la estructura *s* como la vería el procesador. De hecho, si la estructura *s* se compila en modo «little-endian», su patrón en memoria es el que se muestra en la Figura 9.12. Explique la correspondiente asignación, describa una forma fácil para implementarla y discuta la efectividad de esta aproximación.

| Asignación de direcciones "little endian" |     |     |     |     |    |    |    |    |    |    |     |     |     |    |
|-------------------------------------------|-----|-----|-----|-----|----|----|----|----|----|----|-----|-----|-----|----|
| Dirección de byte                         | 00  |     | 01  |     | 02 |    | 03 |    | 04 |    | 05  |     | 06  |    |
| 00                                        | 00  | 01  | 02  | 03  | 04 | 05 | 06 | 07 | 21 | 22 | 23  | 24  | 25  | 26 |
| 08                                        | 08  | 09  | 0A  | 0B  | 0C | 0D | 0E | 0F | 31 | 32 | 33  | 34  |     |    |
| 10                                        | 'D' | 'C' | 'B' | 'A' | 31 | 32 | 33 | 34 | 10 | 11 | 12  | 13  | 14  | 15 |
| 18                                        | 10  | 11  | 12  | 13  | 14 | 15 | 16 | 17 | 51 | 52 | 'G' | 'F' | 'E' |    |
| 20                                        | 18  | 19  | 1A  | 1B  | 1C | 1D | 1E | 1F | 61 | 62 | 63  | 64  |     |    |
|                                           | 20  | 21  | 22  | 23  | 24 | 25 | 26 | 27 |    |    |     |     |     |    |

Figura 9.12. Estructura *s* «little-endian» en memoria del PowerPC.

- 9.19. Escriba un pequeño programa para determinar el tipo de endian de una máquina, e informe del resultado. Ejecute el programa en un computador al que tenga acceso y muestre el resultado.

## APÉNDICE 9A. PIAS

### PILAS

Una *pila* es un conjunto ordenado de elementos, en el que sólo uno de ellos es accesible en un instante dado. El punto de acceso se denomina *cabecera* de la pila. El número de elementos de la pila, o *longitud* de la pila, es variable. Sólo se pueden añadir o eliminar elementos en la cabecera de la pila. Por esta razón, una pila también se denomina *lista último en entrar-primeros en salir* (LIFO, last-in-first-out).

La Figura 9.13 ilustra las operaciones básicas con la pila. Comenzamos en un instante de tiempo en el que la pila contiene algunos elementos. Una operación PUSH (apilar) añade un nuevo elemento en la cabecera de la pila. Una operación POP (desapilar) elimina el elemento de la cabecera de la pila. En ambos casos, la cabecera experimenta el cambio apropiado. Las operaciones binarias, que requieren de dos operandos (por ejemplo, multiplicar, dividir, sumar o restar), utilizan dos elementos de la cabecera de la pila como operandos, desapilando ambos y apilando el resultado de nuevo en la pila. Las operaciones unarias, que requieren sólo un operando (por ejemplo la NOT), utilizan el elemento de la cabecera de la pila. Todas estas operaciones se resumen en la Tabla 9.12.

### IMPLEMENTACIÓN DE LA PILA

La pila es una estructura útil para la CPU. Un posible uso, discutido en la Sección 9.4, es la gestión de llamadas y retornos a/de procedimientos. Las pilas pueden ser también útiles para el programador. Un ejemplo es la evaluación de expresiones, discutida más adelante en esta Sección.

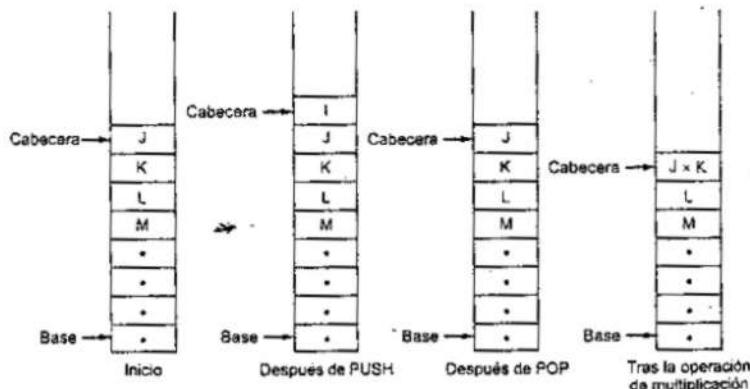


Figura 9.13. Operaciones básicas de la pila.

Tabla 9.12. Operaciones orientadas al uso de la pila

|                   |                                                                                                                                                                                                                                |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| PUSH              | Añade un nuevo elemento en la cabecera de la pila.                                                                                                                                                                             |
| POP               | Elimina el elemento de la cabecera de la pila.                                                                                                                                                                                 |
| Operación unaria  | Realiza una operación con el elemento de la cabecera de la pila.                                                                                                                                                               |
| Operación binaria | Sustituye el elemento de la cabecera con el resultado.<br>Realiza una operación con dos elementos de la cabecera de la pila. Elimina de la pila dichos elementos. Pone el resultado de la operación en la cabecera de la pila. |

La implementación de una pila depende en parte de sus usos potenciales. Si se desean ejecutar operaciones con la pila disponibles para el programador, el repertorio de instrucciones dispondría de operaciones orientadas al manejo de la pila, incluyendo PUSH, POP, y operaciones que utilicen uno o dos elementos de la cabecera de la pila como operandos. Ya que todas estas operaciones hacen referencia a una misma posición, la cabecera de la pila, la dirección del operando u operandos está implícita, y no necesita incluirse en la instrucción. Son pues instrucciones con cero direcciones, a las que nos referimos en la Sección 9.1.

Si el mecanismo de pila va a ser utilizado sólo por la CPU, con usos tales como el manejo de procedimientos, en el repertorio de instrucciones no se contemplarían instrucciones orientadas al uso de la pila. En cualquier caso, la implementación de una pila requiere que exista un cierto conjunto de posiciones, utilizadas para almacenar los elementos de la pila. Una aproximación típica se ilustra en la Figura 9.14a. En memoria principal (o en memoria vir-

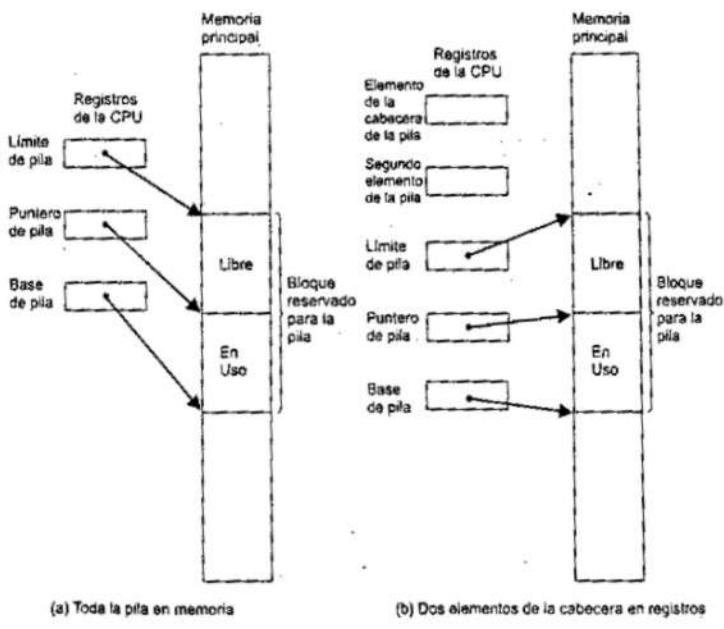


Figura 9.14. Organizaciones de pila usuales.

tual) se reserva un bloque de posiciones contiguas para la pila. La mayor parte del tiempo, el bloque está parcialmente lleno con elementos de la pila, y el resto del bloque está disponible para que crezca la pila. Para un funcionamiento correcto se necesitan tres direcciones, normalmente memorizadas en registros de la CPU:

- **Puntero de pila:** Contiene la dirección del tope o cabecera de la pila. Si se añade, o se elimina, un elemento de la pila, el puntero se incrementa, o decrementa, para que contenga la dirección de la nueva cabecera de la pila.
- **Base de la pila:** Contiene la dirección base del bloque reservado para la pila. Si se intenta un POP cuando la pila está vacía, se informa de un error.
- **Límite de la pila:** Contiene la dirección del otro extremo del bloque reservado. Si se intenta un PUSH cuando el bloque está utilizado en su totalidad, se informa de un error.

Tradicionalmente, en la mayoría de las máquinas actuales la base de la pila coincide con la dirección más alta del bloque reservado para la pila, mientras que el límite se corresponde con la dirección más baja. Por lo tanto, la pila crece desde direcciones más altas hacia más bajas.

Para acelerar las operaciones con la pila, también los dos elementos de la cabecera se almacenan a menudo en registros, como muestra la Figura 9.14b. En este caso, el puntero de pila contiene la dirección del tercer elemento de la pila.

## EVALUACIÓN DE EXPRESIONES

Las fórmulas matemáticas se expresan normalmente en la notación conocida con el nombre de *infija* (infix). En esta notación el operador binario aparece entre los operandos (por ejemplo,  $a + b$ ). En expresiones complejas se emplean paréntesis para determinar el orden de evaluación de las mismas. Por ejemplo,  $a + (b \times c)$  producirá un resultado diferente que  $(a + b) \times c$ . Para minimizar el número de paréntesis, las operaciones tienen una precedencia o prioridad implícita. Generalmente, la multiplicación tiene prioridad sobre la suma, de manera que  $a + b \times c$  es equivalente a:  $a + (b \times c)$ .

Una técnica alternativa es la notación denominada *polaca inversa*, o postfija. En esta notación, el operador se especifica después de los operandos. Por ejemplo,

$$\begin{array}{ll} a + b & \text{sería } a\ b\ + \\ a \div (b \times c) & \text{sería } a\ b\ c\ \times\ + \\ (a + b) \times c & \text{sería } a\ b\ +\ c\ \times \end{array}$$

Observe que, por compleja que sea la expresión, no se requieren paréntesis en esta notación.

La ventaja de la notación postfija es que una expresión con este formato es fácil de evaluar usando una pila. Una expresión en notación postfija se recorre de izquierda a derecha y, para cada elemento de la expresión, se aplican las siguientes reglas:

1. Si el elemento es una variable o una constante, se introduce en la pila.
2. Si el elemento es un operador, se extraen de la cabecera de la pila los dos operandos, se realiza la operación y se apila el resultado.

Tras recorrer la expresión completa, el resultado se encuentra en la cabecera de la pila.

La sencillez de este método lo hace muy apropiado para la evaluación de expresiones. Como consecuencia, muchos compiladores, partiendo de las expresiones descritas en lenguajes de alto nivel, las convierten a notación postfija y, entonces, generan las instrucciones máquina a partir de esta notación. La Figura 9.15 muestra la secuencia de instrucciones má-

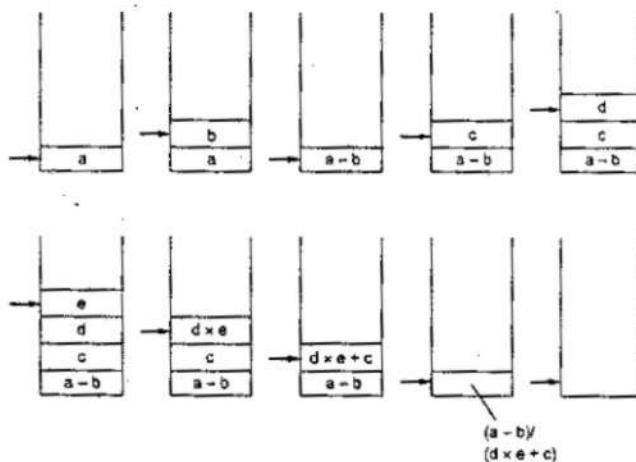
|                         | Pila                                                                                             | Registros generales                                                                                                       | Registro único                                                                          |
|-------------------------|--------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------|
|                         | Push a<br>Push b<br>Subtract<br>Push c<br>Push d<br>Push e<br>Multiply<br>Add<br>Divide<br>Pop f | Load G[1], a<br>Subtract G[1], b<br>Load G[2], d<br>Multiply G[2], e<br>Add G[2], c<br>Divide G[1], G[2]<br>Store G[1], f | Load d<br>Multiply e<br>Add c<br>Store f<br>Load a<br>Subtract b<br>Divide f<br>Store f |
| Número de instrucciones | 10                                                                                               | 7                                                                                                                         | 8                                                                                       |
| Accesos a memoria       | 10 op + 8 d                                                                                      | 7 op + 6 d                                                                                                                | 8 op + 8 d                                                                              |

Figura 9.15. Comparación de tres programas para calcular  $f = (a - b)/(c + d \times e)$ .

quina para evaluar  $f = (a - b)/(c + d \times e)$  utilizando instrucciones orientadas al uso de la pila. La figura muestra también el uso de instrucciones con una y dos direcciones. Observe que, aunque en estos últimos casos no se han utilizado las instrucciones dedicadas a la pila, la notación postfixa ha servido de guía para generar las instrucciones máquina. La secuencia de eventos producidos por el programa que emplea la pila se ilustra en la Figura 9.16.

El propio proceso de conversión de una expresión infija a otra postfixa se lleva a cabo más fácilmente si se utiliza una pila. El siguiente algoritmo se debe a Dijkstra [DIJK63]. La expresión infija se va recorriendo de izquierda a derecha, y se va generando a la vez la expresión postfixa como salida. Los pasos del algoritmo son los siguientes:

1. Examinar el siguiente elemento de la línea de entrada.

Figura 9.16. Utilización de la pila para calcular  $f = (a - b)/(c + d \times e)$ .

| Entrada                             | Salida               | Pila<br>(cabecera<br>a la derecha) |
|-------------------------------------|----------------------|------------------------------------|
| $A + B \times C + (D + E) \times F$ | vacia                | vacia                              |
| $+ B \times C + (D + E) \times F$   | A                    | vacia                              |
| $B \times C + (D + E) \times F$     | A                    | +                                  |
| $\times C + (D + E) \times F$       | AB                   | +                                  |
| $C + (D + E) \times F$              | AB                   | +\times                            |
| $+ (D + E) \times F$                | ABC                  | +\times                            |
| $(D + E) \times F$                  | ABC\times            | +                                  |
| $D + E) \times F$                   | ABC\times+           | +(                                 |
| $+ E) \times F$                     | ABC\times+D          | +(                                 |
| $E) \times F$                       | ABC\times+DE         | +(+                                |
| $) \times F$                        | ABC\times+DE+        | +(+                                |
| F                                   | ABC\times+DE+        | +\times                            |
| vacia                               | ABC\times+DE+F       | +\times                            |
| vacia                               | ABC\times+DE+F\times | vacia                              |

Figura 9.17. Conversión de una expresión de notación infija a postfija.

- Extraerlo como salida si se trata de un operando.
- Si es una apertura de paréntesis, se introduce en la pila.
- Si es un operador, entonces:
  - Si la cabecera de la pila contiene una apertura de paréntesis, apilar el operador.
  - Si tiene una prioridad más alta que el de la cabecera de la pila (la multiplicación y la división tienen mayor prioridad que la suma y la resta), apilar el operador.
  - Si tiene una prioridad menor o igual, efectuar un «pop» de la pila a la salida y repetir el paso 4.
- Si es un paréntesis de cierre, desapilar operadores hacia la salida hasta que se encuentre un paréntesis de apertura. Desapilar y descartar el paréntesis de apertura.
- Si hay más entradas, ir al paso 1.
- Si no hay más elementos de entrada, desapilar los restantes operadores.

La Figura 9.17 ilustra el uso de este algoritmo. Este ejemplo da al lector una idea de la potencia de los algoritmos basados en la pila.

#### APÉNDICE 9B. «LITTLE», «BIG» Y BI-ENDIAN»

Hay un aspecto curioso y molesto, relacionado con la forma en que se referencian y se representan los bytes dentro de una palabra y los bits dentro de un byte. Veremos primero el problema del orden de los bytes, y después consideraremos el de los bits.

#### ORDEN DE LOS BYTES

El concepto de «endian» fue introducido por Cohen [COHE81]. En relación con los bytes, este concepto tiene que ver con la ordenación de los bytes en un escalar multi-byte. Este tema

se introduce mejor con un ejemplo. Suponga que tenemos el valor hexadecimal de 32 bits 12345678, y que se almacena en una palabra de 32 bits de una memoria direccionable por bytes, en la posición de byte 184. El valor consta de cuatro bytes, con el contenido 78 para byte menos significativo, y el byte más significativo con el valor 12 como contenido. Hay dos maneras de memorizar este valor:

| Dirección | Valor | Dirección | Valor |
|-----------|-------|-----------|-------|
| 184       | 12    | 184       | 78    |
| 185       | 34    | 185       | 56    |
| 186       | 56    | 186       | 34    |
| 187       | 78    | 187       | 12    |

La asignación de la izquierda almacena el byte más significativo en la dirección de byte con valor numérico más bajo; este modo se denomina «big-endian», y es equivalente al orden de escritura de izquierda a derecha utilizado en las lenguas de las culturas occidentales. La asignación de la derecha almacena el byte menos significativo en la dirección con valor más bajo, y se conoce con el nombre de «little-endian», equivalente al orden de derecha a izquierda seguido para las operaciones aritméticas<sup>2</sup>. Para un valor escalar multi-byte dado, la «little-endian» y la «big-endian» son asignaciones inversas en términos de bytes.

El concepto de los «endians» surge cuando es necesario tratar una entidad multi-byte como un único dato con una sola dirección, aun cuando esté compuesto de unidades direccionables más pequeñas. Algunos procesadores, tales como el Intel 80×86, el Pentium II, el VAX y el Alpha, son máquinas «little-endian», mientras que otros, tales como el IBM System 370/390, el Motorola 680×0, el Sun SPARC y la mayoría de los RISC, son máquinas «big-endian». Esto presenta problemas cuando se transfieren datos de una máquina con un tipo de «endian» a otra de distinto tipo, y cuando un programador intenta manipular bytes o bits individuales de un escalar multi-byte.

La característica de «endians» se aplica sólo a unidades de datos individuales. En cualquier máquina, elementos tales como los ficheros, las estructuras de datos y las matrices, constan de múltiples unidades de datos, cada una de ellas con la característica de «endian». Así pues, la conversión de un bloque de memoria con un tipo de «endian» a otro, requiere conocer la estructura de los datos.

La Figura 9.18 ilustra cómo el tipo de «endian» determina el orden de direccionamiento y de los bytes. La estructura C de la parte superior contiene varios tipos de datos. El trazado de la memoria de la parte inferior izquierda es el resultado de la compilación de dicha estructura para una máquina «big-endian», y el de la parte inferior derecha para una «little-endian». En ambos casos se muestra la memoria mediante una serie de filas de 64 bits. Para el caso «big-endian», la memoria se representa de izquierda a derecha y de arriba a abajo, mientras que para el caso «little-endian» se hace de derecha a izquierda y de arriba a abajo. Observe que estos trazados son arbitrarios. Cualquiera de los esquemas podría emplear un trazado de izquierda a derecha o de derecha a izquierda dentro de una fila; esto es algo que depende de la forma de hacer el trazado, y no de la asignación de memoria. De hecho, al

<sup>2</sup> Los términos *big-endian* y *little-endian* proceden de la Primera Parte, Capítulo 4, de *Los claves de Guliver*, de Jonathan Swift. Se refieren a una guerra religiosa entre dos grupos, uno que rompía los huevos por el extremo más grueso (*big-endian*), y el otro que lo hacía por el extremo más estrecho (*little-endian*).

```

Struct {
 int a; //0x1112_1314
 int pad1; //
 double b; //0x2122_2324_2526_2728
 char* c; //0x3132_3334
 char d[7]; // 'A', 'B', 'C', 'D', 'E', 'F', 'G'
 short e; //0x5152
 int f; //0x6161_6364
} s;

```

| Asignación de direcciones "big-endian" |     |     |     |    |     |     |     | Asignación de direcciones "little-endian" |    |    |    |    |     |     |     |     |     |     |    |    |    |    |    |    |
|----------------------------------------|-----|-----|-----|----|-----|-----|-----|-------------------------------------------|----|----|----|----|-----|-----|-----|-----|-----|-----|----|----|----|----|----|----|
| Dirección de byte                      | 11  | 12  | 13  | 14 | 00  | 01  | 02  | 03                                        | 04 | 05 | 06 | 07 | 08  | 09  | 0A  | 0B  | 0C  | 0D  | 0E | 0F | 11 | 12 | 13 | 14 |
| 00                                     | 21  | 22  | 23  | 24 | 25  | 26  | 27  | 28                                        |    |    |    |    | 21  | 22  | 23  | 24  | 25  | 26  | 27 | 28 |    |    |    |    |
| 08                                     | 08  | 09  | 0A  | 0B | 0C  | 0D  | 0E  | 0F                                        |    |    |    |    | 0F  | 0E  | 0D  | 0C  | 0B  | 0A  | 09 | 08 |    |    |    |    |
| 10                                     | 31  | 32  | 33  | 34 | 'A' | 'B' | 'C' | 'D'                                       |    |    |    |    | 'D' | 'C' | 'B' | 'A' | 31  | 32  | 33 | 34 |    |    |    |    |
| 18                                     | 10  | 11  | 12  | 13 | 14  | 15  | 16  | 17                                        |    |    |    |    | 17  | 18  | 15  | 14  | 13  | 12  | 11 | 10 |    |    |    |    |
|                                        | 'E' | 'F' | 'G' |    | 51  | 52  |     |                                           |    |    |    |    | 51  | 52  |     | 'G' | 'F' | 'E' |    |    |    |    |    |    |
| 20                                     | 18  | 19  | 1A  | 1B | 1C  | 1D  | 1E  | 1F                                        |    |    |    |    | 1F  | 1E  | 1D  | 1C  | 1B  | 1A  | 19 | 18 |    |    |    |    |
|                                        | 61  | 62  | 63  | 64 |     |     |     |                                           |    |    |    |    |     |     |     |     |     |     |    |    | 61 | 62 | 63 | 64 |
|                                        | 20  | 21  | 22  | 23 |     |     |     |                                           |    |    |    |    |     |     |     |     |     |     |    |    | 23 | 22 | 21 | 20 |

Figura 9.18. Ejemplo de estructura de datos en C, y sus mapas «endians» (IBM94).

mirar los manuales de programación de diversas máquinas, encontramos una confusa colección de formatos, incluso dentro de un mismo manual.

Podemos hacer varias observaciones acerca de esta estructura de datos:

- Cada elemento de datos tiene la misma dirección en ambos esquemas. Por ejemplo, la dirección de la palabra doble con el valor hexadecimal 2122232425262728, es 08.
- Dentro de un valor escalar multi-byte dado, el orden de los bytes en la estructura «little-endian» es el inverso del de la estructura «big-endian».
- El tipo de «endian» no afecta al orden de los elementos de datos dentro de una estructura. Así, en la palabra c de cuatro caracteres los bytes se invierten, pero no en la matriz de bytes d, de siete caracteres. Por lo tanto, la dirección de cada elemento individual de d es la misma en ambas estructuras.

El efecto del tipo de «endian» se muestra quizás más claramente si consideramos la memoria como una matriz vertical de bytes, como se muestra en la Figura 9.19.

No existe un consenso general sobre qué tipo de «endian» es mejor<sup>3</sup>. Los siguientes puntos hacen preferible el estilo «big-endian»:

- Ordenación de cadenas de caracteres: Un procesador «big-endian» es más rápido en comparar cadenas de caracteres alineadas como enteros; la ALU de enteros puede comparar varios bytes en paralelo.
- Volcados decimal/ASCII: Todos los valores pueden imprimirse de izquierda a derecha sin causar confusión.

<sup>3</sup> El profeta venerado por ambos grupos en las «guerras de Endian» de los Viajes de Gulliver dijo: «Todos los creyentes convencidos deben romper los huevos por el extremo conveniente». Esto no nos es de gran ayuda.

|    |     |    |     |
|----|-----|----|-----|
| 00 | 11  | 00 | 14  |
|    | 12  |    | 13  |
|    | 13  |    | 12  |
|    | 14  |    | 11  |
| 04 |     | 04 |     |
|    |     |    | •   |
| 08 | 21  | 08 | 28  |
|    | 22  |    | 27  |
|    | 23  |    | 26  |
|    | 24  |    | 25  |
| 0C | 25  | 0C | 24  |
|    | 26  |    | 23  |
|    | 27  |    | 22  |
|    | 28  |    | 21  |
| 10 | 31  | 10 | 34  |
|    | 32  |    | 33  |
|    | 33  |    | 32  |
|    | 34  |    | 31  |
| 14 | 'A' | 14 | 'A' |
|    | 'B' |    | 'B' |
|    | 'C' |    | 'C' |
|    | 'D' |    | 'D' |
| 18 | 'E' | 18 | 'E' |
|    | 'F' |    | 'F' |
|    | 'G' |    | 'G' |
| 1C | 51  | 1C | 52  |
|    | 52  |    | 51  |
|    |     |    |     |
| 20 | 61  | 20 | 64  |
|    | 62  |    | 63  |
|    | 63  |    | 62  |
|    | 64  |    | 61  |

(a) Big-endian

(b) Little-endian

Figura 9.19. Otra visión de la Figura 9.18.

- **Orden coherente:** Los procesadores «big-endian» almacenan en el mismo orden los enteros y las cadenas de caracteres (el byte más significativo primero).

Los puntos siguientes favorecen al estilo «little endian»:

- Un procesador «big-endian» tiene que efectuar sumas cuando convierte una dirección entera de 32 bits a una de 16 bits, al objeto de utilizar los bytes menos significativos.
- Es más fácil realizar operaciones aritméticas de alta precisión con el estilo «little-endian»; no hay que buscar primero el byte menos significativo y luego retroceder.

Las diferencias son poco significativas, y la elección del estilo de «endian» está más motivada por la necesidad de compatibilizar con máquinas anteriores que por otras causas.

El PowerPC es un procesador «bi-endian» que admite los dos modos, «little-endian» y «big-endian». La arquitectura «bi-endian» permite a los que desarrollan software elegir uno u otro modo cuando migran aplicaciones o sistemas operativos de otras máquinas. El sistema operativo establece el modo de «endian» en el que se ejecutan los procesos. Una vez seleccionado un modo, todas las transferencias a/desde memoria subsiguientes emplean ese modo. Para materializar esta facilidad hardware, el sistema operativo mantiene dos bits del registro de estado de la máquina (MSR) como parte del estado del proceso. Un bit especifica el modo de «endian» en que ejecuta el núcleo; el otro especifica el modo de operación actual del procesador. Así pues, el modo puede cambiarse de proceso a proceso.

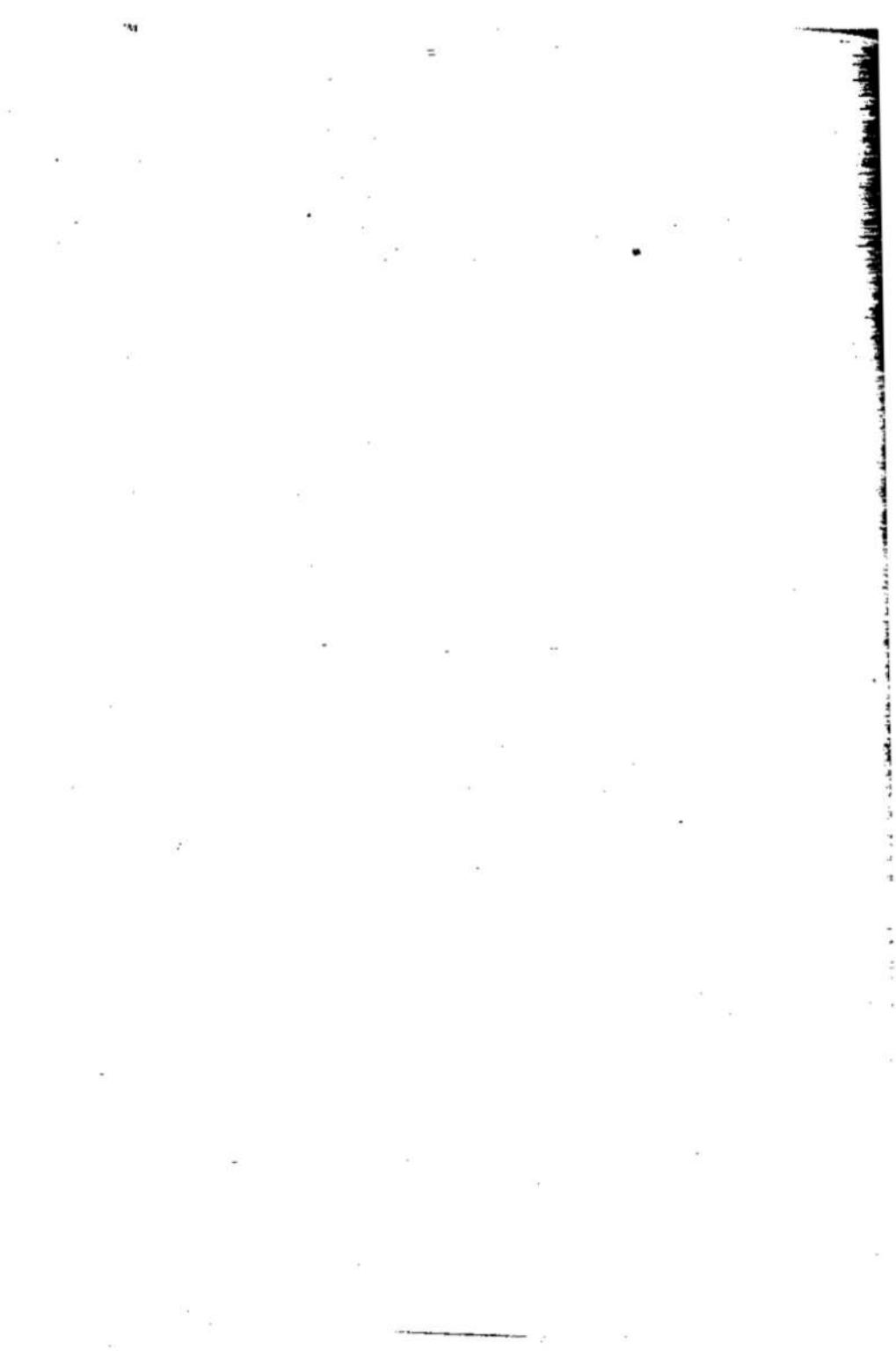
## ORDEN DE LOS BITS

Al considerar el orden de los bits dentro de un byte, inmediatamente nos planteamos dos cuestiones:

1. ¿Se cuenta el primer bit como bit número 0 o como número 1?
2. ¿El número de bit más bajo lo asignamos al bit menos significativo del byte (little-endian) o al más significativo (big-endian)?

Estas cuestiones no se responden por igual para las distintas máquinas. Realmente, en algunas máquinas, las respuestas son diferentes según las circunstancias. Además, la elección entre «big» o «little-endian» para la ordenación de bits dentro de un byte no es siempre coherente con la ordenación «big» o «little-endian» de los bytes en un escalar multi-byte. El programador necesita ser consciente de estos detalles cuando manipula bits individuales.

Otro aspecto a considerar es la transmisión de datos a través de una línea serie. Cuando se transmite un byte individual, ¿envía el sistema primero el bit más significativo, o el menos significativo? El diseñador debe asegurarse de que los bits recibidos son correctamente tratados. Para una discusión sobre este tema, véase [JAME90].



## CAPÍTULO 10

# **Repertorios de instrucciones: modos de direccionamiento y formatos**

### **10.1. Direccionamiento**

- Direccionamiento inmediato
- Direccionamiento directo
- Direccionamiento indirecto
- Direccionamiento de registros
- Direccionamiento indirecto con registro
- Direccionamiento con desplazamiento
- Direccionamiento de pila

### **10.2. Modos de direccionamiento en el Pentium y el PowerPC**

- Modos de direccionamiento del Pentium II
- Modos de direccionamiento del PowerPC

### **10.3. Formatos de instrucciones**

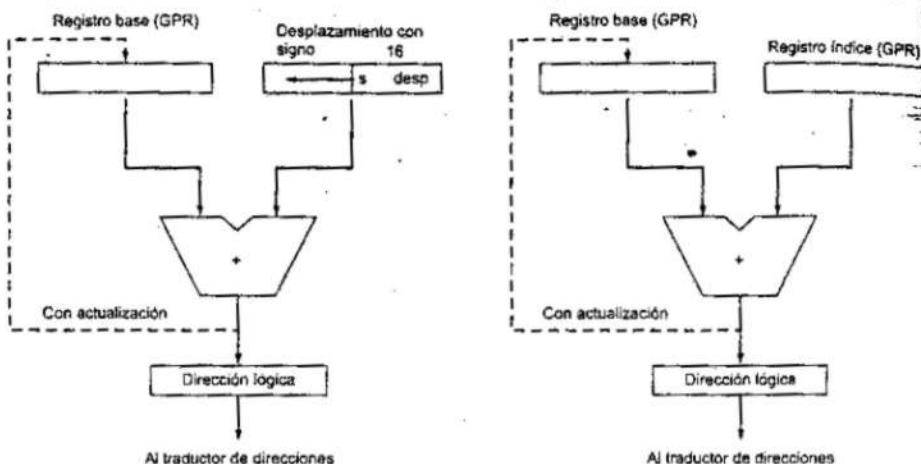
- Longitud de las instrucciones
- Asignación de los bits
- Instrucciones de longitud variable

### **10.4. Formatos de instrucciones del Pentium y del PowerPC**

- Formatos de instrucción del Pentium II
- Formatos de instrucción del PowerPC

### **10.5. Lecturas recomendadas**

### **10.6. Problemas**



- La referencia a un operando en una instrucción contiene, bien su valor (inmediato), bien una referencia a la dirección del operando. Los diversos repertorios de instrucciones utilizan una gran variedad de modos de direccionamiento. Estos modos incluyen: el direccionamiento directo (la dirección del operando está en el campo de direcciones), indirecto (el campo de direcciones apunta a la posición que contiene la dirección del operando), a registro, indirecto con registro, y diversos tipos de desplazamiento, en los que el valor de un registro se suma a un valor de dirección para producir la dirección del operando.
- El formato de instrucción define la forma de los distintos campos de la instrucción. El diseño del formato de instrucciones es una tarea difícil, que debe considerar la longitud de las instrucciones, fija o variable, los números de bits asignados al código de operación y a cada referencia a operando, y la forma en que se determina el modo de direccionamiento.

**E**n el Capítulo 9 nos hemos centrado en qué hace una instrucción. Concretamente, hemos examinado los tipos de operandos y de operaciones que pueden especificarse mediante instrucciones máquina. Este Capítulo 10 aborda la cuestión de cómo especificar los operandos y las operaciones de las instrucciones. Hay dos aspectos a tener en cuenta: el primero es cómo especificar la dirección de un operando, y el segundo cómo se organizan los bits de una instrucción para definir las direcciones de los operandos y la operación que realiza dicha instrucción.

### 10.1. DIRECCIONAMIENTO

El campo o campos de direcciones en un formato de instrucción usual está bastante limitado. Sería deseable poder referenciar un rango elevado de posiciones de memoria principal o, en algunos sistemas, de memoria virtual. Para conseguir este objetivo se han empleado diversas técnicas de direccionamiento. Todas ellas implican algún compromiso entre el rango de direcciones y/o flexibilidad de direccionamiento de una parte, y por otra, el número de referencias a memoria y/o la complejidad de cálculo de las direcciones. En esta sección analizamos los modos de direccionamiento más comunes:

- Inmediato
- Directo
- Indirecto
- Registro
- Indirecto con registro
- Con desplazamiento
- Pila

Estos modos se ilustran en la Figura 10.1. En esta sección utilizaremos la siguiente notación:

A = Contenido de un campo de dirección en la instrucción.

R = Contenido de un campo de dirección en la instrucción que referencia un registro.

EA = Dirección real (efectiva) de la posición que contiene el operando que se referencia.

(X) = Contenido de la posición X.

La Tabla 10.1 indica el cálculo de la dirección realizado para cada modo de direccionamiento.

Antes de comenzar esta discusión, debemos hacer dos comentarios. El primero es que prácticamente todas las arquitecturas de computadores ofrecen más de uno de estos modos de direccionamiento. La cuestión que surge es cómo determina la unidad de control qué modo de direccionamiento se está empleando en cada instrucción. Se adoptan diversos enfoques alternativos. A menudo, *codigos* diferentes emplearán modos de direccionamiento distintos. También, uno o más bits del formato de instrucción pueden utilizarse como *campo de modo*. El valor del campo de modo determina qué modo de direccionamiento va a utilizarse.

El segundo comentario se refiere a la *dirección efectiva* (EA, effective address). En un sistema sin memoria virtual, la dirección efectiva será o una dirección de memoria principal o un registro. En un sistema con memoria virtual, la dirección efectiva es una dirección virtual o un registro. La correspondencia real con una dirección física dependerá del mecanismo de paginación, y no está visible al programador.

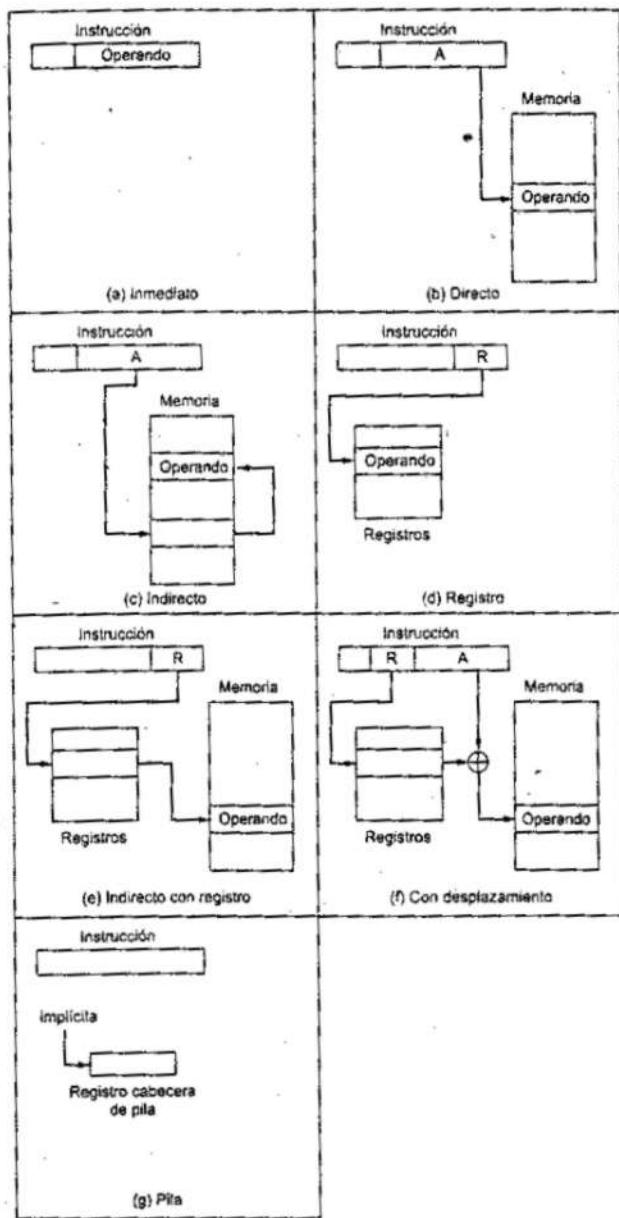


Figura 10.1. Modos de direccionamiento.

Tabla 10.1. Modos de direccionamiento básicos

| Modo                   | Algoritmo                | Principal ventaja             | Principal desventaja            |
|------------------------|--------------------------|-------------------------------|---------------------------------|
| Inmediato              | Operando = A             | No referencia a memoria       | Operando de magnitud limitada   |
| Directo                | EA = A                   | Es sencillo                   | Espacio de direcciones limitado |
| Indirecto              | EA = (A)                 | Espacio de direcciones grande | Referencias a memoria múltiples |
| Registro               | EA = R                   | No referencia a memoria       | Número limitado de registros    |
| Indirecto con registro | EA = (R)                 | Espacio de direcciones grande | Referencia extra a memoria      |
| Con desplazamiento     | EA = A + (R)             | Flexibilidad                  | Complejidad                     |
| Pila                   | EA = cabecera de la pila | No referencia a memoria       | Aplicabilidad limitada          |

## DIRECCIONAMIENTO INMEDIATO

La forma más sencilla de direccionamiento es el direccionamiento inmediato, en el que el operando está en realidad presente en la propia instrucción:

$$\text{OPERANDO} = \text{A}$$

Este modo puede utilizarse para definir y utilizar constantes, o para fijar valores iniciales de variables. Normalmente, el número se almacena en complemento a dos; el bit más a la izquierda del campo de operando se utiliza como bit de signo. Cuando el operando se carga en un registro de datos, el bit de signo se replica hacia la izquierda hasta la longitud total de la palabra de datos.

La ventaja del direccionamiento inmediato es que, una vez captada la instrucción, no se requiere una referencia a memoria para obtener el operando, ahorrándose pues un ciclo de memoria o de cache en el ciclo de instrucción. La desventaja es que el tamaño del número está restringido a la longitud del campo de direcciones que, en la mayoría de los repertorios de instrucciones, es pequeño comparado con la longitud de palabra.

## DIRECCIONAMIENTO DIRECTO

Una forma muy sencilla de direccionamiento es el direccionamiento directo, en el que el campo de direcciones contiene la dirección efectiva del operando.

$$\text{EA} = \text{A}$$

Esta técnica fue común en las primeras generaciones de computadores, y se encuentra aún en diversos sistemas. Sólo requiere una referencia a memoria, y no necesita ningún cálculo especial. La limitación obvia, mencionada con anterioridad, es que proporciona un espacio de direcciones restringido.

## DIRECCIONAMIENTO INDIRECTO

El problema del direccionamiento directo es que la longitud del campo de direcciones es, normalmente, menor que la longitud de palabra, limitando pues el rango de direcciones. Una solución es hacer que el campo de direcciones refiera la dirección de una palabra de memoria que contenga la dirección completa del operando. Esto es lo que se denomina *direcciónamiento indirecto*:

$$\text{EA} = (\text{A})$$

Como se ha definido anteriormente, el paréntesis se interpreta como «contenido de». La ventaja obvia de esta aproximación es que, para una longitud de palabra de  $N$  bits, se dispone ahora de un espacio de direcciones de  $2^N$ . La desventaja es que la ejecución de la instrucción requiere dos referencias a memoria para captar el operando: una para captar su dirección, otra para obtener su valor.

Notará que, aunque el número de palabras que pueden ahora direccionarse es  $2^N$ , el número de direcciones efectivas diferentes que pueden referenciarse en un instante dado está limitado a  $2^K$ , donde  $K$  es la longitud del campo de direcciones. Normalmente, esto no supone una restricción severa, y puede superarse. En un entorno de memoria virtual, todas las posiciones de direcciones efectivas pueden confinarse a la página 0 de cualquier proceso. Ya que el campo de direcciones de una instrucción es pequeño, sólo se podrán generar direcciones directas con valores bajos; éstas aparecerán en la página 0. (La única restricción es que el tamaño de página debe ser mayor o igual que  $2^K$ .) Cuando un proceso está activo, habrá repetidas referencias a la página 0, haciendo que se quede en memoria real. Por tanto, una referencia indirecta a memoria implicará, como mucho, una falta de página en lugar de dos.

Una variante de direccionamiento indirecto, raramente utilizada, es el direccionamiento multinivel o en cascada:

$$EA = (\dots(A)\dots)$$

En este caso, uno de los bits de una palabra de dirección es un «índicador de indirección» (I). Si el bit I es 0, la palabra contiene el valor de EA. Si el bit I es 1, entonces se invoca otro «nivel de indirección». No parece que esta aproximación ofrezca ninguna ventaja particular, y su desventaja es que podrían requerirse tres o más referencias para captar un operando.

## DIRECCIONAMIENTO DE REGISTROS

El direccionamiento de registros es similar al directo. La única diferencia es que el campo de direcciones referencia un registro, en lugar de una dirección de memoria principal:

$$EA = R$$

Normalmente, un campo de direcciones que referencia a registros consta de 3 o 4 bits, de manera que pueden referenciarse un total de 8 o 16 registros de uso general.

Las ventajas del direccionamiento de registros son que (1) sólo es necesario un campo pequeño de direcciones en la instrucción, y (2) no se requieren referencias a memoria. Como se comentó en el Capítulo 4, el tiempo de acceso a un registro interno a la CPU es mucho menor que para la memoria principal. La desventaja del direccionamiento a registros es que el espacio de direcciones está muy limitado.

Si se hace un uso masivo del direccionamiento a registros en un repertorio de instrucciones, los registros de la CPU se emplearán intensamente. Debido al número tan limitado de registros (en comparación con el número de posiciones de memoria principal), usarlos de esta manera tiene sentido sólo si se emplean eficientemente. Si todo operando se carga de memoria principal a un registro, se opera con él una vez, y se devuelve a memoria principal, resulta que se estaría añadiendo un paso intermedio improcedente. Si en lugar de esto, el operando del registro se mantiene en uso durante varias operaciones, se estaría consiguiendo un ahorro real. Un ejemplo son los resultados intermedios de un cálculo. En particular, suponga que se va a implementar en software el algoritmo de multiplicación en complemento a dos. La posición A del diagrama de flujo (Figura 8.12) se referencia muchas veces, y debiera estar implementada en un registro en lugar de en una posición de memoria principal.

Depende del programador decidir qué valores deben mantenerse en registros y cuáles deben almacenarse en memoria principal. La mayoría de las CPU modernas emplean múltiples registros de uso general, cargando al programador en lenguaje ensamblador (cuando desarrolla compiladores, por ejemplo) con la responsabilidad de conseguir una ejecución eficiente.

### DIRECCIONAMIENTO INDIRECTO CON REGISTRO

Igual que el direccionamiento de registros es análogo al directo, el direccionamiento indirecto con registro es análogo al direccionamiento indirecto. En ambos casos, la diferencia estriba en si el campo de direcciones hace referencia a una posición de memoria o a un registro. Así, para el direccionamiento indirecto con registro tenemos:

$$EA = (R)$$

Las ventajas y limitaciones del direccionamiento indirecto con registro son básicamente las mismas que se tienen para el direccionamiento indirecto. En ambos casos, la limitación del espacio (o rango de direcciones del campo de direcciones) se supera haciendo que dicho campo refiera una posición de una palabra completa (un registro completo, en este caso) que contenga la dirección. Además, el direccionamiento indirecto con registro emplea una referencia menos a memoria que el direccionamiento indirecto.

### DIRECCIONAMIENTO CON DESPLAZAMIENTO

Un modo muy potente de direccionamiento combina las posibilidades de los direccionamientos directo, e indirecto con registro. Se conoce con distintos nombres, dependiendo del contexto en que se emplee, pero el mecanismo básico es el mismo. Nosotros usaremos la denominación de *direcciónamiento con desplazamiento*:

$$EA = A + (R)$$

El direcciónamiento con desplazamiento requiere que las instrucciones tengan dos campos de direcciones, al menos uno de ellos explícito. El valor contenido en uno de los campos de direcciones (valor = A) se utiliza directamente; el otro campo de direcciones, o una referencia implícita definida por el código de operación, se refiere a un registro cuyo contenido se suma a A para generar la dirección efectiva.

Describiremos tres de los usos más comunes del direcciónamiento con desplazamiento:

- Desplazamiento relativo
- Direcciónamiento con registro base
- Indexado

#### Direcciónamiento relativo

Para el direcciónamiento relativo, el registro referenciado implícitamente es el contador de programa (PC). Es decir, la dirección de instrucción actual se suma al campo de direcciones para producir el valor EA. Normalmente, el campo de direcciones se trata como número en complemento a dos para esta operación. En consecuencia, la dirección efectiva es un desplazamiento relativo a la dirección de la instrucción.

El desplazamiento relativo aprovecha el concepto de localidad discutido en los Capítulos 4 y 7. Si la mayoría de las referencias a memoria están relativamente próximas a la instrucción en ejecución, el uso del direcciónamiento relativo ahorra bits de direcciones en la instrucción.

### Direccionamiento con registro-base

La interpretación del direccionamiento con registro-base es la siguiente: el registro referenciado contiene una dirección de memoria, y el campo de dirección contiene un desplazamiento (normalmente en representación entera sin signo) desde dicha dirección. La referencia a registro puede ser explícita o implícita.

El direccionamiento con registro base aprovecha también la localidad de las referencias a memoria. Es una forma conveniente de implementar la segmentación, que se estudió en el Capítulo 7. En algunas implementaciones se emplea un solo registro de base de segmento, y es utilizado implícitamente. En otras, el programador puede seleccionar un registro para guardar la dirección de base de un segmento, y la instrucción debe referenciarlo de manera explícita. En este último caso, si la longitud del campo de direcciones es  $K$  y el número de registros posibles es  $N$ , una instrucción puede referenciar una de entre  $N$  áreas de  $2^K$  palabras.

### Indexado

La interpretación usual del indexado es la siguiente: el campo de dirección referencia una dirección de memoria principal, y el registro referenciado contiene un desplazamiento positivo desde esa dirección. Obsérvese que este uso es justo el opuesto de la interpretación del direccionamiento con registro base. Por supuesto, hay algo más que una mera cuestión de interpretación de uso. Ya que en el indexado se considera que el campo de direcciones es una dirección de memoria, generalmente contiene más bits que un campo de direcciones de una instrucción comparable que emplee direccionamiento con registro base. También veremos que hay algunas mejoras del indexado que no serían tan útiles en el contexto de registros base. No obstante, el método para calcular EA es el mismo para ambos tipos de direccionamiento, y en ambos las referencias a los registros son a veces explícitas y a veces implícitas (para diferentes CPU).

Un uso importante del indexado es como mecanismo eficiente para ejecutar operaciones iterativas. Considere, por ejemplo, una lista de números almacenados a partir de la posición A. Suponga que se quiere sumar 1 a cada elemento de la lista. Necesitamos captar cada valor, sumarle 1, y memorizar el resultado. La secuencia de direcciones efectivas necesarias es A, A + 1, A + 2, ..., hasta la última posición de la lista. Con el indexado, esto es fácil de hacer. El valor A se almacena en el campo de dirección de la instrucción, y el registro elegido, llamado *registro índice*, se inicializa a 0. Tras cada operación, el registro índice se incrementa en 1.

Dado que los registros índice se usan normalmente para tales tareas iterativas, es normal incrementarlos o decrementarlos tras cada referencia. Ya que ésta es una operación común, algunos sistemas la efectúan automáticamente como parte del ciclo de instrucción. Esto se denomina *autoindexado*. Si un registro concreto está dedicado exclusivamente al indexado, el autoindexado puede ser invocado implícita y automáticamente. Si se emplean registros de uso general, la operación de auto-indexado puede requerir de un bit de la instrucción que lo indique. El auto-indexado con incremento puede describirse como sigue:

$$EA = A + (R)$$

$$(R) \leftarrow (R) + 1$$

Algunas máquinas disponen de direccionamiento tanto indirecto como indexado, y es posible emplear ambos en la misma instrucción. Hay dos posibilidades: el indexado se realiza bien antes o bien después de la indirección.

Si la indexación se realiza después de la indirección se denomina *post-indexado*:

$$EA = (A) + (R)$$

Primero, el contenido del campo de direcciones se emplea para acceder a la posición de memoria que contiene una dirección directa. Esta dirección se indexa entonces mediante el valor del registro. Esta técnica es útil para acceder a uno de entre un número de bloques de datos con un formato fijo. Por ejemplo, en el Capítulo 7 se describió que el sistema operativo necesita emplear un bloque de control para cada proceso. Las operaciones que se realizan son las mismas, independientemente del bloque manipulado. Por lo tanto, las direcciones indicadas en las instrucciones que referencian el bloque podrían apuntar a una posición (valor = A) que contenga un puntero variable hacia el inicio del bloque de control de proceso. El registro índice contiene el desplazamiento dentro del bloque.

Con *pre-indexado*, la indexación se realiza antes de la indirección:

$$EA = (A + (R))$$

La dirección se calcula como en el caso de indexado simple. No obstante, en este caso la dirección calculada no contiene al operando, sino la dirección del operando. Un ejemplo de uso de esta técnica es la construcción de tablas de bifurcación multi-rama. En un punto concreto de un programa puede haber una bifurcación a una de entre varias posiciones, en función de diversas condiciones. Puede establecerse una tabla de direcciones que comience en la posición A y, mediante indexado en esta tabla, encontrar la posición requerida.

Normalmente, un mismo repertorio de instrucciones no incluirá el pre-indexado y el post-indexado simultáneamente.

## DIRECCIONAMIENTO DE PILA

El último modo de direccionamiento que consideramos es el de pila. Como definimos en el Apéndice 9A, una pila es una matriz lineal de posiciones. A veces se le denomina *lista de apilamiento* o *cola último en entrar-primer en salir*. Una pila es un bloque de posiciones reservado. Los elementos se añaden en la cabecera de la pila de manera que, en cualquier instante, el bloque está parcialmente lleno. La pila tiene asociado un puntero, cuyo valor es la dirección de la cabecera o tope de la pila. Alternativamente, los dos elementos de la cabecera de la pila pueden residir en registros de la CPU, en cuyo caso el puntero de pila hace referencia al tercer elemento de la pila (Figura 9.14b). El puntero de pila se mantiene en un registro. Así, las referencias a posiciones de la pila en memoria son, de hecho, direcciones de acceso indirecto con registro.

El modo de direccionamiento de pila es una forma de direccionamiento implícito. Las instrucciones máquina no necesitan incluir una referencia a memoria, sino que operan implícitamente con la cabecera de la pila. Las pilas no habían sido demasiado empleadas antes, pero se han hecho bastante comunes en microprocesadores.

## 10.2 MODOS DE DIRECCIONAMIENTO EN EL PENTIUM Y EL PowerPC

### MODOS DE DIRECCIONAMIENTO DEL PENTIUM II

Recordemos, de la Figura 7.22, que el mecanismo de traducción de direcciones del Pentium II produce una dirección, denominada «dirección virtual o efectiva», que es un desplazamiento

dentro de un segmento. La suma de la dirección de comienzo del segmento y la dirección efectiva produce una dirección lineal. Si se está empleando paginación, esta dirección lineal debe pasar por un mecanismo de traducción de páginas para producir una dirección física. En lo que sigue ignoraremos este último paso, ya que es transparente para el repertorio de instrucciones y para el programador.

El Pentium II está equipado con diversos modos de direccionamiento, ideados para permitir la ejecución eficiente de lenguajes de alto nivel. La Figura 10.2 indica el hardware involucrado. El segmento objeto de la referencia se determina mediante el registro de segmento. Hay seis registros de segmento; cuál de ellos sea usado para una referencia dada depende del contexto de ejecución y de la instrucción. Cada registro de segmento retiene la dirección de comienzo del correspondiente segmento. Asociado con cada registro de segmento visible para el usuario, hay un registro descriptor de segmentos (no visible al programador), que registra los derechos de acceso para el segmento, así como la dirección de comienzo y el límite (longitud) del segmento. Además, hay dos registros que pueden emplearse para construir una dirección: el registro base y el registro índice.

La Tabla 10.2 lista los 12 modos de direccionamiento del Pentium II. Consideremos cada uno de ellos por separado.

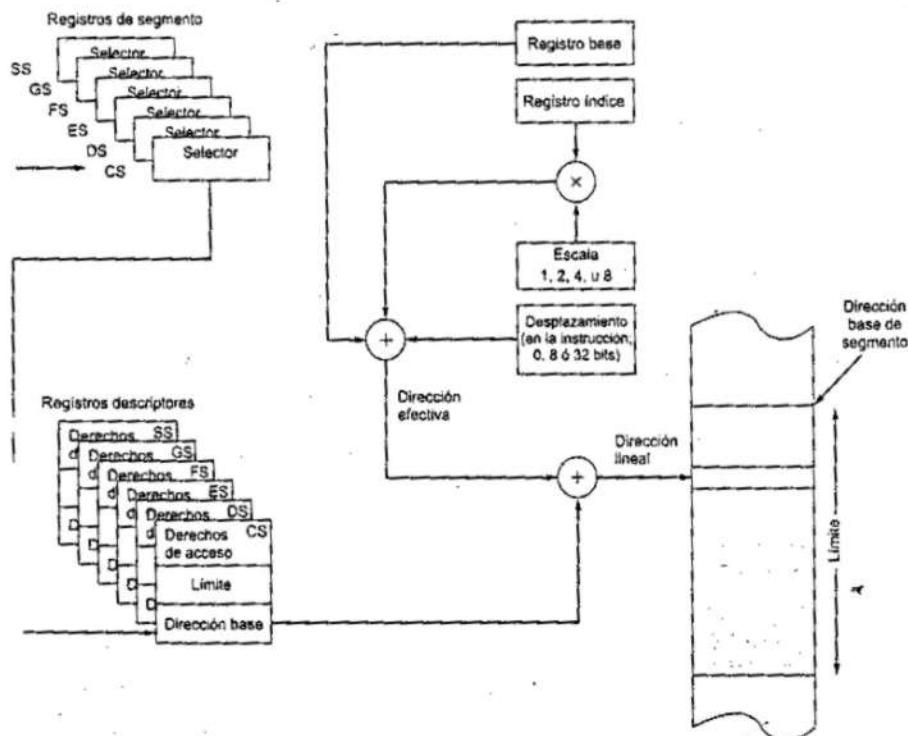


Figura 10.2. Cálculos en el modo de direccionamiento del Pentium II.

Tabla 10.2. Modos de direccionamiento del Pentium II

| Modo                                      | Algoritmo                     |
|-------------------------------------------|-------------------------------|
| Inmediato                                 | Operando = A                  |
| Registro                                  | LA = R                        |
| Con desplazamiento                        | LA = (SR) + A                 |
| Base                                      | LA = (SR) + (B)               |
| Base con desplazamiento                   | LA = (SR) + (B) + A           |
| Índice escalado con desplazamiento        | LA = (SR) + (I) × S + A       |
| Base con índice y desplazamiento          | LA = (SR) + (B) + (I) + A     |
| Base con índice escalado y desplazamiento | LA = (SR) + (I) × S + (B) + A |
| Relativo                                  | LA = (PC) + A                 |

LA = dirección lineal

(X) = contenido de X

SR = registro de segmento

PC = contador de programa

A = contenido de un campo de dirección de la instrucción

R = registro

B = registro base

I = registro índice

S = factor de escala

En el modo inmediato, el operando se incluye en la instrucción. El operando puede ser un byte, una palabra o una palabra doble de datos.

En el modo de operando en registro, el operando está situado en un registro. Para instrucciones de tipo general, tales como instrucciones de transferencias de datos, aritméticas, y lógicas, el operando puede ser uno de los registros generales de 32 bits (EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP), uno de los registros generales de 16 bits (AX, BX, CX, DX, SI, DI, SP, BP), o uno de los registros generales de 8 bits (AH, BH, CH, DH, AL, BL, CL, DL). Para operaciones en coma flotante, los operandos de 64 bits se forman utilizando dos registros de 32 bits como una pareja. Hay también algunas instrucciones que hacen referencia a los registros de segmento (CS, DS, ES, SS, FS, GS).

Los modos de direccionamiento restantes referencian posiciones de memoria. La posición de memoria debe especificarse en términos del segmento que la contiene y el desplazamiento desde el comienzo del segmento. En algunos casos, el segmento se especifica de manera explícita; en otros, queda especificado por las reglas de asignación implícita de segmentos.

En el modo de desplazamiento, el desplazamiento del operando (la dirección efectiva en la Figura 10.2) está incluido, formando parte de la instrucción, como desplazamiento de 8, 16 o 32 bits. Con segmentación, todas las direcciones dadas en instrucciones hacen referencia a desplazamientos dentro de segmentos. El modo de direccionamiento con desplazamiento se puede encontrar en pocas máquinas, ya que, como hemos mencionado antes, implica instrucciones largas. En el caso del Pentium II, el valor de desplazamiento puede ser tan largo como 32 bits, haciendo que la instrucción tenga 6 bytes. El direccionamiento con desplazamiento puede ser útil para referenciar variables globales.

Los modos de direccionamiento restantes son indirectos, en el sentido de que el campo de dirección de la instrucción indica al procesador dónde debe encontrar la dirección. El modo base especifica que uno de los registros de 8, 16 o 32 bits, contiene la dirección efectiva. Esto es equivalente a lo que hemos denominado «direcciónamiento indirecto con registro».

En el modo base con desplazamiento, la instrucción incluye un desplazamiento que ha que sumar a un registro base, que puede ser cualquiera de los registros de uso general. Ejemplos de uso de este modo son:

- Utilización por un compilador para apuntar, al comienzo de una zona de variables locales. Por ejemplo, el registro base podría apuntar al comienzo de un marco de pila que contiene las variables locales para el procedimiento correspondiente.
- Utilización para indexar en una matriz cuando el tamaño de elemento no es 2, 4 u 8 bytes, y no puede por tanto ser indexado utilizando un registro índice. En este caso, el desplazamiento apunta al comienzo de la matriz, y el registro base retiene los resultados del cálculo para determinar el desplazamiento de un elemento específico dentro de la matriz.
- Utilización para acceder a un campo de un registro. El registro base apunta al comienzo del registro, mientras que el desplazamiento indica la posición del campo.

En el modo desplazamiento con índice «escalado», la instrucción incluye un desplazamiento a sumar a un registro, llamado en este caso «registro índice». El registro índice puede ser cualquiera de los registros de uso general, excepto el ESP, que normalmente se emplea para procesamiento con la pila. Para calcular la dirección efectiva, el contenido del registro índice se multiplica por un factor de escala 1, 2, 4 u 8, y se suma después a un desplazamiento. Este modo es muy conveniente para indexar matrices. Un factor de escala 2 puede usarse para una matriz de enteros de 16 bits y un factor de escala 4 para enteros de 32 bits o para números en coma flotante; finalmente, un factor de escala 8 puede emplearse para una matriz de números en coma flotante de doble precisión.

El modo base con índice y desplazamiento suma los contenidos de los registros base e índice, y un desplazamiento, para formar la dirección efectiva. De nuevo, el registro base puede ser cualquier registro de uso general, y el registro índice puede ser cualquier registro de uso general excepto el ESP. Como ejemplo, este modo de direccionamiento podría emplearse para acceder a una matriz local en un marco de pila. Este modo puede usarse también para manejar una matriz bidimensional; en este caso, el desplazamiento apunta al inicio de la matriz, y cada registro gestiona una dimensión de la misma.

El modo base con índice escalado y desplazamiento suma el contenido de registro índice, multiplicado por un factor de escala, con el contenido del registro base y el desplazamiento. Esto es útil cuando una matriz está almacenada en un marco de pila; en este caso, los elementos de la matriz serían de 2, 4 u 8 bytes de longitud. Este modo permite también la indexación eficiente de una matriz bidimensional cuando los elementos de la misma tienen longitudes de 2, 4 u 8 bytes.

Finalmente, el direccionamiento relativo puede emplearse en instrucciones de transferencia del control (control de flujo). Se suma un desplazamiento al valor del contador de programa, que apunta a la instrucción siguiente. En este caso, el desplazamiento se trata como un byte, una palabra, o una palabra doble numérica con signo, cuyo valor bien incrementa, o bien decrementa la dirección contenida en el contador de programa.

## MODOS DE DIRECCIONAMIENTO DEL PowerPC

Como la mayoría de las máquinas RISC, y en contraste con el Pentium II y la mayoría de los CISC, el PowerPC emplea un conjunto de modos de direccionamiento sencillo y relativamente evidente. Como indica la Tabla 10.3, estos modos conviene clasificarlos con relación al tipo de instrucciones.

Tabla 10.3. Modos de direccionamiento del PowerPC

| Modo               | Algoritmo                                                 |
|--------------------|-----------------------------------------------------------|
| Indirecto          | Direccionamiento para carga/memorización<br>EA = (BR) + D |
| Indirecto indexado | EA = (BR) + (IR)                                          |
| Absoluto           | Direccionamiento de bifurcaciones                         |
| Relativo           | EA = I                                                    |
| Indirecto          | EA = (LCR)                                                |
| Registro           | Cálculos en coma fija                                     |
| Inmediato          | EA = GPR<br>Operando = I                                  |
| Registro           | Cálculos en coma flotante<br>EA = FPR                     |

EA = dirección efectiva  
 (X) = contenido de X  
 BR = registro base  
 IR = registro índice  
 LCR = registro de enlace o de cuenta  
 GPR = registro de uso general  
 FPR = registro de coma flotante  
 D = desplazamiento  
 I = valor inmediato  
 PC = contador de programa

### Direccionamiento para Carga/memorización

El PowerPC proporciona dos modos de direccionamiento alternativos para instrucciones de carga/memorización (Figura 10.3). En el direccionamiento indirecto, la instrucción incluye un desplazamiento de 16 bits, que se suma a un registro base, que puede ser alguno de los registros de uso general. Además, la instrucción puede especificar que la nueva dirección efectiva

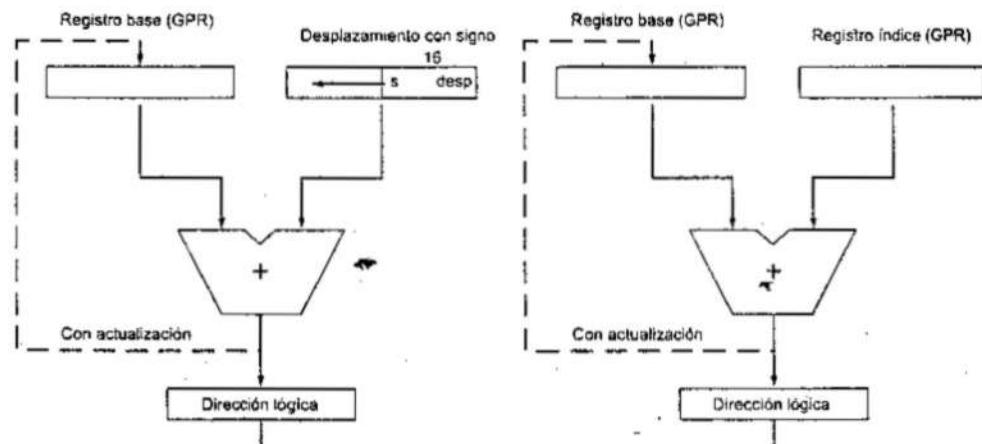


Figura 10.3. Modos de direccionamiento de operandos en memoria del PowerPC (DIEF94b).

calculada se devuelva al registro base, actualizando su contenido actual. La opción de actualización es útil para el indexado progresivo de matrices en bucles.

La otra técnica de direccionamiento para las instrucciones de carga/memorización es direcciónnamiento indexado indirecto. En este caso, la instrucción referencia un registro base, otro índice, pudiendo ambos ser cualesquiera de los registros de uso general. La dirección efectiva es la suma de los contenidos de estos dos registros. De nuevo, la opción de actualización hace que el registro base se actualice con la nueva dirección efectiva.

### Direccionamiento de bifurcaciones

Se dispone de tres modos de direccionamiento para bifurcaciones. Si se emplea direccionamiento absoluto con instrucciones de salto incondicional, la dirección efectiva de la siguiente instrucción se obtiene a partir de un valor inmediato de 24 bits contenido en la instrucción. El valor de 24 bits se extiende hasta 32 bits añadiendo dos ceros a su extremo menos significativo (esto es posible, ya que las instrucciones están cada 32 bits) y expandiendo (replicando) el signo. Para las instrucciones de salto condicional, la dirección efectiva de la siguiente instrucción se deduce de un valor inmediato de 16 bits incluido en la instrucción. Este valor se extiende a un valor de 32 bits, añadiendo dos ceros a su extremo menos significativo y expandiendo el signo.

Con direccionamiento relativo, el valor inmediato de 24 bits (instrucciones de salto incondicional) o de 14 bits (instrucciones de salto condicional) se extiende como en los casos anteriores. El valor resultante se suma entonces al contador de programa, a fin de determinar una posición relativa a la instrucción actual. El otro modo de direccionamiento de bifurcaciones condicionales es el direccionamiento indirecto. Este modo obtiene la dirección efectiva de la siguiente instrucción bien del registro de enlace, bien del registro de cuenta. En este caso, el registro de cuenta se usa para retener la dirección de la instrucción de bifurcación. Este registro puede también emplearse para mantener un contador para bucles, como explicamos con anterioridad.

### Instrucciones aritméticas

En operaciones aritméticas con enteros, un operando debe estar en un registro o bien formar parte de la instrucción. Con direccionamiento de registros, un operando fuente o destino se especifica como el contenido de uno de los registros de uso general. Con direccionamiento inmediato, un operando fuente aparece como cantidad de 16 bits, con signo, en la propia instrucción.

Para las operaciones aritméticas en coma flotante, todos los operandos se encuentran en registros de coma flotante; es decir, sólo se utiliza direccionamiento de registros.

## 0.3. FORMATOS DE INSTRUCCIONES

Un formato de instrucciones define la descripción en bits de una instrucción en términos de las distintas partes que la componen. Un formato de instrucciones debe incluir un código de operación (codop) e, implícita o explícitamente, ninguno o algunos operandos. Cada operando explícito se referencia utilizando uno de los modos de direccionamiento descritos en la Sección 10.1. El formato debe, implícita o explícitamente, indicar el modo de direccionamiento para cada operando. En la mayoría de los repertorios de instrucciones se emplea más de un formato de instrucción.

El diseño de un formato de instrucción es una labor compleja, habiéndose implementando una variedad muy amplia de diseños. En esta sección examinamos los aspectos clave de diseño, analizando brevemente algunos diseños que servirán de ilustración, y después examinaremos con detalle las soluciones adoptadas en el Pentium II y en el PowerPC.

## LONGITUD DE INSTRUCCIÓN

El aspecto de diseño más básico a considerar en el formato es la longitud de la instrucción. Esta decisión afecta, y se ve afectada, por el tamaño de la memoria, su organización, la estructura de buses, la complejidad de la CPU, y la velocidad de la CPU. Esta decisión define la riqueza y flexibilidad de la máquina desde el punto de vista del programador en lenguaje ensamblador.

El compromiso más obvio está entre el deseo de disponer de un repertorio de instrucciones máquina potente y la necesidad de ahorrar espacio. El programador desea más codops, más operandos, más modos de direccionamiento y mayor rango de direcciones. Más codops y más operandos facilitan el trabajo del programador, ya que puede redactar programas más cortos para resolver las mismas tareas. De manera similar, más modos de direccionamiento dan más flexibilidad al programador para implementar ciertas funciones, tales como la gestión de tablas y las bifurcaciones multi-rama. Y, por supuesto, con el aumento de tamaño de la memoria principal y el uso creciente de la memoria virtual, los programadores demandan poder direccionar rangos de memoria grandes. Todo esto (codops, operandos, modos de direccionamiento y rango de direcciones) requiere de bits, y empuja hacia longitudes de instrucción mayores. Pero una longitud de instrucción mayor puede ser imprudente. Una instrucción de 32 bits ocupa el doble de espacio que una de 16 bits pero, probablemente, no es el doble de útil.

A parte de este compromiso básico, hay otras consideraciones a tener en cuenta. Debiera cumplirse o que el tamaño de la instrucción fuera igual al tamaño de las transferencias a memoria (en un sistema basado en un bus, igual al tamaño del bus de datos), o que uno fuera un múltiplo del otro. En caso contrario, no conseguiremos un número entero de instrucciones durante un ciclo de captación. Una cuestión relacionada es la velocidad de transferencia de la memoria. Esta velocidad no ha seguido el mismo aumento que la velocidad de los procesadores. Por ello, la memoria puede convertirse en un cuello de botella si el procesador puede ejecutar las instrucciones más rápido que lo que tarda en capturarlas. Una solución a este problema es el uso de memoria cache (véase la Sección 4.3); otra es utilizar instrucciones más cortas. De nuevo, las instrucciones de 16 bits pueden captarse el doble de rápido que las de 32 bits, pero probablemente no pueden ejecutarse el doble de rápido.

Una característica aparentemente irrelevante, pero sin embargo importante, es que la longitud de la instrucción debiera ser un múltiplo de la longitud de un carácter, que normalmente es 8 bits, y de la longitud de los números en coma fija. Para verlo necesitamos hacer uso de un término desafortunadamente mal definido, la «palabra» [FRA183]. La longitud de palabra de memoria es, en cierto sentido, la unidad «natural» de organización. El tamaño de una palabra define normalmente el tamaño de los números en coma fija (usualmente ambos coinciden). El tamaño de palabra suele también coincidir (o, al menos, está directamente relacionado) con el tamaño de las transferencias a memoria. Ya que una forma común de datos es el carácter, sería deseable que una palabra almacenara un número entero de caracteres. Si no, se perderían bits en cada palabra cuando se almacenan múltiples caracteres, o habría algunos caracteres partidos entre dos palabras. La importancia de este punto es tal, que IBM, cuando introdujo el Sistema/360 y quiso emplear caracteres de 8 bits, tuvo que adoptar la decisión drástica de pasar de la arquitectura de 36 bits de las máquinas científicas de las series 700/7000 a una arquitectura de 32 bits.

## ASIGNACIÓN DE LOS BITS

Hemos visto algunos de los factores que influyen en la decisión de la longitud del formato de instrucción. Un aspecto igualmente difícil es cómo asignar los bits en dicho formato. Los compromisos a la hora de decidir son, en este caso, complejos.\*

Para una longitud de instrucción dada, existe claramente un compromiso entre el número de codops y la capacidad de direccionamiento. Un mayor número de codops, obviamente, implica más bits en el campo de codop. Esto reduce, para un formato de instrucción de una longitud dada, el número de bits disponibles para direccionamiento. Hay un refinamiento interesante al respecto, consistente en el uso de codops de longitud variable. En esta aproximación, existe una longitud mínima de codop, pero para algunos de ellos se pueden especificar operaciones adicionales utilizando más bits de la instrucción. En una instrucción de longitud fija, esto deja menos bits para direccionamiento. Así pues, esta característica se emplea en aquellas instrucciones que requieren menos operandos y/o menor capacidad de direccionamiento.

Los siguientes factores, relacionados entre sí, afectan a la definición del uso dado a los bits de direccionamiento.

- **Número de modos de direccionamiento:** Un modo de direccionamiento puede a veces indicarse de manera implícita. Por ejemplo, ciertos codops podrían siempre hacer referencia a indexación. En otros casos, los modos de direccionamiento deben ser explícitos, requiriéndose uno o más bits de modo.
- **Número de operandos:** Hemos visto que menos direcciones pueden hacer que los programas sean más largos y difíciles (véase como ejemplo la Figura 9.3). Las instrucciones típicas de las máquinas actuales permiten dos operandos. Cada dirección de operando podría requerir su propio indicador de modo dentro de la instrucción, o el uso del indicador de modo podría estar limitado a sólo uno de los campos de direcciones.
- **Registros frente a memoria:** Una máquina debe disponer de registros para traer los datos a la CPU a fin de procesarlos. En el caso de un solo registro visible para el usuario (normalmente denominado «acumulador»), la dirección del operando está implícita, y no consume bits de la instrucción. Sin embargo, la programación con un único registro es engorrosa, y requiere muchas instrucciones. Incluso con varios registros, sólo se necesitan unos pocos bits para especificar el registro. Diversos estudios indican que es aconsejable disponer de 8 a 32 registros visibles para el usuario [LUND77, HUCK83].
- **Número de conjuntos de registros:** Varias máquinas tienen un conjunto de registros de uso general, que contiene 8 o 16 registros. Estos registros pueden emplearse para guardar datos y para almacenar direcciones para direccionamiento con desplazamiento. La tendencia actual ha sido pasar de un solo banco de registros de uso general a un grupo de dos o más conjuntos especializados (por ejemplo, para datos y para desplazamientos). Esta tendencia se observa cada vez más a todos los niveles, desde microprocesadores en un solo chip hasta en supercomputadores. Una ventaja de este enfoque es que, para un número dado de registros, una partición funcional de éstos requiere menos bits de la instrucción. Por ejemplo, con dos conjuntos de ocho registros, sólo se necesitan 3 bits para identificar un registro; el codop determina de forma implícita qué conjunto de registros se está referenciando. Esta aproximación parece tener pocas desventajas [LUND77]. En sistemas tales como el S/370, que tiene un conjunto de registros de uso general, el programador normalmente establece como criterio asignar aproximadamente la mitad de los registros para datos y la mitad para desplazamiento, manteniendo una asignación fija [MALL79].

- **Rango de direcciones:** Para referencias a memoria, el rango de direcciones que puede utilizarse está relacionado con el número de bits de direccionamiento. Dado que esto impone una limitación severa, raramente se emplea direccionamiento directo. En direccionamiento con desplazamiento, el rango se amplía al definido por la longitud del registro de direcciones. Incluso así, es aún conveniente permitir desplazamientos bastante más largos que los del registro de direcciones, y esto requiere un número relativamente grande de bits de direcciones en la instrucción.
- **Granularidad de las direcciones:** Para direcciones que hacen referencia a memoria en lugar de registros, otro factor es la granularidad del direccionamiento. En un sistema con palabras de 16 o 32 bits, una dirección puede referenciar una palabra o un byte, según elija el diseñador. El direccionamiento por bytes es conveniente para manipular caracteres, pero requiere, para un tamaño de memoria dado, de más bits de direcciones.

Por lo tanto, el diseñador se enfrenta con una gran cantidad de factores a tener en cuenta y sopesar. No está claro lo críticas que son las distintas opciones. Como ejemplo, citamos un estudio [CRAG79] que compara varios enfoques al formato de instrucciones, incluyendo el uso de una pila, registros de uso general, un acumulador, y aproximaciones sólo memoria-a-registro. Haciendo uso de un conjunto coherente de suposiciones, no se observan diferencias significativas en espacio de código o en tiempo de ejecución.

Veamos brevemente cómo dos diseños concretos sopesan los distintos factores anteriores.

### PDP-8

Uno de los diseños de instrucciones más sencillos para un computador de uso general fue el del PDP-8 [BELL78b]. El PDP-8 utiliza instrucciones de 12 bits y opera con palabras de 12 bits. Hay un solo registro de uso general, el acumulador.

A pesar de lo limitado de este diseño, el direccionamiento es bastante flexible. Cada referencia a memoria consta de 7 bits más dos modificadores de 1 bit. La memoria se divide en páginas de longitud fija, cada una con  $2^7 = 128$  palabras. El cálculo de direcciones se basa en las referencias a la página 0 o a la página actual (página que contiene la instrucción), según el valor del bit de página. El segundo bit modificador indica si se va a usar direccionamiento directo o indirecto. Estos dos modificadores pueden utilizarse conjuntamente, de tal manera que una dirección indirecta será una dirección de 12 bits contenida en una palabra de la página 0 o de la página actual. Además, 8 palabras dedicadas de la página 0 son «registros» de auto-indexado. Cuando se hace una referencia indirecta a una de estas posiciones, tiene lugar un pre-indexado.

La Figura 10.4 muestra el formato de instrucción del PDP-8. Hay un codop de 3 bits y 3 tipos de instrucciones. Para los codops 0 a 5, el formato consiste en una instrucción con una sola referencia a memoria, incluyendo un bit de página y un bit de indirección. Así pues, hay sólo 6 operaciones básicas. Para ampliar el grupo de operaciones, el codop 7 define una referencia a registro o *microinstrucción*. En este formato, los bits restantes se emplean para codificar operaciones adicionales. En general, cada bit define una operación específica (por ejemplo, poner el acumulador a cero), y estos bits pueden combinarse en una sola instrucción. La estrategia de las microinstrucciones la empleó DEC ya con el PDP-1, y es, en cierto sentido, un precursor de las máquinas microprogramadas actuales, que se tratarán en la Parte IV del libro. El codop 6 es la operación de E/S: se emplean 6 bits para seleccionar uno de entre 64 dispositivos, y 3 bits especifican una orden particular de E/S.

El formato de instrucción del PDP-8 es bastante eficiente. Permite direccionamiento indirecto, direccionamiento con desplazamiento e indexado. Con el uso de la ampliación de

## Instrucciones con referencia a memoria

| Codop | D/I | Z/C |   | Desplazamiento |    |
|-------|-----|-----|---|----------------|----|
| 0     | 2   | 3   | 4 | 5              | 11 |

## Instrucciones de entrada/salida

|   |   |   |   |             |   |       |    |
|---|---|---|---|-------------|---|-------|----|
| 1 | 1 | 0 |   | Dispositivo |   | Codop |    |
| 0 |   | 2 | 3 |             | 8 | 9     | 11 |

## Instrucciones con referencia a registro

## Microinstrucciones del grupo 1

|   |   |   |   |     |     |     |     |     |     |     |     |
|---|---|---|---|-----|-----|-----|-----|-----|-----|-----|-----|
| 1 | 1 | 1 | 0 | CLA | CLL | CMA | CML | RAR | RAL | BSW | IAC |
| 0 | 1 | 2 | 3 | 4   | 5   | 6   | 7   | 8   | 9   | 10  | 11  |

## Microinstrucciones del grupo 2

|   |   |   |   |     |     |     |     |     |     |     |    |
|---|---|---|---|-----|-----|-----|-----|-----|-----|-----|----|
| 1 | 1 | 1 | 1 | CLA | SMA | SZA | SNL | RSS | OSR | HLT | 0  |
| 0 | 1 | 2 | 3 | 4   | 5   | 6   | 7   | 8   | 9   | 10  | 11 |

## Microinstrucciones del grupo 3

|   |   |   |   |     |     |   |     |   |   |    |    |
|---|---|---|---|-----|-----|---|-----|---|---|----|----|
| 1 | 1 | 1 | 1 | CLA | MQA | 0 | MQL | 0 | 0 | 0  | 1  |
| 0 | 1 | 2 | 3 | 4   | 5   | 6 | 7   | 8 | 9 | 10 | 11 |

## Nomenclátorios

CLA = Acumulador a cero

SMA = Salto si acumulador negativo

CLL = Enlace a cero

SZA = Salto si acumulador cero

CMA = Complementar acumulador

SNL = Salto si enlace distinto de cero

CML = Complementar enlace

RSS = Invertir sentido del salto

RAR = Rotar a derecha acumulador

OSR = OR con registro comutador

RAL = Rotar a izquierda acumulador

HLT = Parar

BSW = Intercambiar Byte

MQA = Multiplicador / cociente en el acumulador

IAC = Incrementar acumulador

MQL = Cargar multiplicador / cociente

Figura 10.4. Formatos de instrucciones del PDP-8.

codop, dispone de un total de 35 instrucciones. Debido a lo limitado de una longitud de instrucción de sólo 12 bits, los diseñadores difícilmente lo podrían haber hecho mejor.

## PDP-10

En fuerte contraste con el repertorio de instrucciones del PDP-8 está el del PDP-10. El PDP-10 se diseñó para ser un sistema de tiempo compartido de gran escala, haciendo hincapié en que fuera fácil de programar, incluso a costa de hardware adicional.

Algunos de los principios de diseño que se emplearon al definir el repertorio de instrucciones fueron [BELL78c]:

- **Ortogonalidad:** Es un principio que hace que dos variables sean independientes entre sí. En el contexto de los repertorios de instrucciones, este término indica que otros elementos de una instrucción son independientes de (no están determinados por) el codop. Los diseñadores del PDP-10 utilizan este término para describir el hecho de que una dirección se calcula siempre de la misma manera, independientemente del codop. En esto difiere de muchas máquinas, en las que el modo de direccionamiento a veces depende implícitamente del operador que se está usando.
- **Complitud:** Cada tipo de datos aritméticos (enteros, en coma fija, reales) debiera disponer de un conjunto completo e idéntico de operaciones.
- **Direccionamiento directo:** El direccionamiento mediante base más desplazamiento, que responsabiliza al programador de la organización de la memoria, fue evitado en favor del direccionamiento directo.

Cada uno de los principios anteriores es un avance hacia la meta global de una programación fácil.

El PDP-10 tiene una longitud de palabra de 36 bits, y una longitud de instrucción de 36 bits. El formato fijo de instrucción se muestra en la Figura 10.5. El codop ocupa 9 bits, permitiendo hasta 512 operaciones. De hecho, se define un total de 365 instrucciones diferentes. La mayoría de las instrucciones incluyen dos direcciones, una de las cuales especifica uno de los 16 registros de uso general. Esta referencia a operando ocupa pues 4 bits. La otra referencia a operando comienza con un campo de direcciones de 18 bits. Este puede utilizarse como operando inmediato o como una dirección de memoria. En el segundo caso, se permite direccionamiento, tanto indirecto como indexado. Los mismos registros de uso general se emplean también como registros índice.

Una longitud de instrucción de 36 bits es realmente un lujo. No hay que hacer nada especial para conseguir aumentar el número de codops; un campo de código de operación de 9 bits es más que suficiente. El direccionamiento es también obvio. Un campo de dirección de 18 bits hace deseable el direccionamiento directo. Para tamaños de memoria superiores a  $2^{18}$ , se dispone del direccionamiento indirecto. Para facilidad del programador, se dispone de indexado para manejar tablas y programas iterativos. También, con un campo de operando de 18 bits, resulta atractivo el direccionamiento inmediato.

El diseño del repertorio de instrucciones del PDP-10 cumple los objetivos antes enumerados [LUND77]. Este repertorio facilita el trabajo del programador, a costa de una utilización ineficiente del espacio. Esto fue una decisión consciente de los diseñadores, y no puede, pues, calificarse como un diseño poco elaborado.

## INSTRUCCIONES DE LONGITUD VARIABLE

Los ejemplos vistos hasta ahora hacen uso de una única longitud de instrucción fija e, implicitamente, hemos discutido decisiones adoptadas en ese contexto. No obstante, los

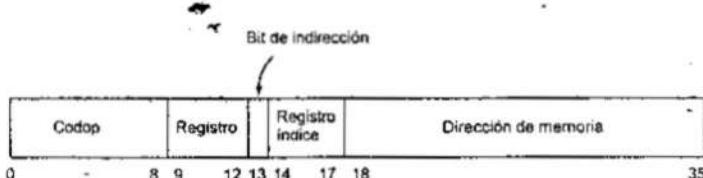


Figura 10.5. Formato de instrucción del PDP-10.

diseñadores pueden optar por utilizar varios formatos de instrucción de longitudes diferentes. Esta táctica hace fácil proporcionar un amplio repertorio de codops de longitud variable. El direccionamiento puede ser más flexible, con varias combinaciones de referencias a registro y a memoria, así como de modos de direccionamiento. Con instrucciones de longitud variable, estas múltiples variaciones pueden proporcionarse de manera eficiente y compacta.

El precio a pagar por las instrucciones de longitud variable es el aumento de complejidad de la CPU. La disminución del precio del hardware, el uso de microprogramación (discutida en la Parte IV), y un aumento general en el conocimiento de los principios de diseño de la CPU, contribuyen todos a hacer que el precio a pagar mencionado sea pequeño.

El uso de instrucciones de longitud variable no elimina el deseo de que todas las longitudes de instrucciones estén directamente relacionadas con la longitud de palabra. Ya que la CPU no conoce la longitud de la instrucción que va a captar, una estrategia usual es captar un número de bytes o de palabras igual, al menos, al tamaño de la instrucción más larga. Esto significa que a veces se captan varias instrucciones. Sin embargo, como veremos en el Capítulo 11, ésta es en general una buena estrategia a seguir.

### PDP-11

El PDP-11 se diseñó para proporcionar un repertorio de instrucciones flexible dentro de las limitaciones de un minicomputador de 16 bits [BELL70].

El PDP-11 emplea un conjunto de 8 registros de uso general de 16 bits. Dos de estos registros tienen una función adicional: uno se emplea como puntero de pila en operaciones específicas con la pila, y el otro se emplea como contador de programa, que contiene la dirección de la siguiente instrucción.

La Figura 10.6 muestra los formatos de instrucciones del PDP-11. Se usan 13 formatos, compaginando instrucciones de ninguna, una, y dos direcciones. La longitud del codop puede variar de 4 a 16 bits. Para las referencias a registros se emplean 6 bits. Tres bits identifican el registro, y los otros tres restantes indican el modo de direccionamiento. El PDP-11 está dotado de una rica variedad de modos de direccionamiento. Una ventaja de asociar el modo de direccionamiento con el operando en lugar de con el codop, es que todos los modos de direccionamiento pueden emplearse con cualquier codop. Como hemos mencionado con anterioridad, esta independencia se denomina *ortogonalidad*.

Las instrucciones del PDP-11 tienen normalmente la longitud de una palabra (16 bits). En algunas instrucciones se añade una o dos direcciones de memoria, así que hay instrucciones de 32 y de 48 bits formando parte del repertorio. Esto permite una flexibilidad de direccionamiento adicional.

El repertorio de instrucciones del PDP-11 y su capacidad de direccionamiento son complejos. Esto aumenta tanto el coste del hardware como la complejidad de programación. La ventaja es que pueden desarrollarse programas más eficientes o compactos.

### VAX

La mayoría de las arquitecturas proporcionan un número relativamente pequeño de formatos de instrucciones. Esto puede causar problemas al programador. En primer lugar, el modo de direccionamiento y el codop no son ortogonales. Por ejemplo, para una operación dada, un operando debe proceder de un registro y otro de memoria, o ambos de registros, etc. En segundo lugar, sólo puede contemplarse un número limitado de operandos: normalmente hasta dos o tres. Dado que algunas operaciones requieren de más operandos, deben emplearse estrategias que permitan conseguir el resultado deseado mediante dos o más instrucciones.

|    |       |        |         |    |       |         |        |       |       |                |
|----|-------|--------|---------|----|-------|---------|--------|-------|-------|----------------|
| 1  | Codop | Fuente | Destino | 2  | Codop | R       | Fuente | 3     | Codop | Desplazamiento |
| 4  | 6     | 6      | 6       | 7  | 3     | 6       | 8      | 8     | 8     | 8              |
| 4  | Codop | FP     | Destino | 5  | Codop | Destino | 6      | Codop | CC    |                |
| 8  | 2     | 6      | 6       | 10 | 6     | 6       | 12     | 4     |       |                |
| 7  | Codop | R      | 8       |    | Codop |         |        |       |       |                |
| 13 | 3     | 3      |         |    | 16    |         |        |       |       |                |
| 10 | Codop | R      | Fuente  |    |       |         |        |       |       |                |
| 7  | 3     | 6      |         |    |       |         |        |       |       |                |
| 11 | Codop | FP     | Fuente  |    |       |         |        |       |       |                |
| 8  | 2     | 6      |         |    |       |         |        |       |       |                |
| 12 | Codop |        | Destino |    |       |         |        |       |       |                |
| 10 |       | 6      |         |    |       |         |        |       |       |                |
| 13 | Codop | Fuente | Destino |    |       |         |        |       |       |                |
| 4  | 6     | 6      |         |    |       |         |        |       |       |                |

Fuente y destino contienen un campo de modo de direccionamiento de 3 bits y un número del registro de 3 bits.

FP es uno de los registros de punto flotante 0, 1, 2 o 3.

R es uno de los registros de uso general.

CC es el campo de códigos de condición.

Figura 10.6. Formatos de instrucciones utilizados en el PDP-11; los números indican longitudes de campos.

Para evitar estos problemas, se siguieron dos criterios al diseñar el formato de instrucción del VAX [STRE78]:

1. Todas las instrucciones debían tener el número «natural» de operandos.
2. Todos los operando debían tener la misma generalidad de especificación.

El resultado es un formato de instrucción muy variable. Una instrucción consta de un codop de 1 o dos bytes, seguido de una especificación de cero a seis operandos, que depende del codop en cuestión. La longitud mínima de instrucción es 1 byte, pudiéndose construir instrucciones de hasta 37 bytes. La Figura 10.7 muestra algunos ejemplos.

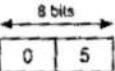
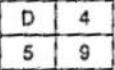
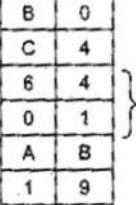
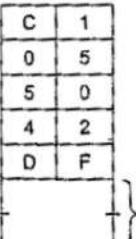
| Formato hexadecimal                                                                 | Explicación                                                                                                                                                                                                 | Notación en ensamblador y descripción                                                                                                                                      |
|-------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|    | Codop de RSB                                                                                                                                                                                                | RSB<br>Retorno de subrutina                                                                                                                                                |
|    | Codop de CLRL<br>Registro R9                                                                                                                                                                                | CLRL R9<br>Desocupar (clear) el registro R9                                                                                                                                |
|   | Codop de MOVW<br>Modo de desplazamiento de palabra, registro R4<br>356 en hexadecimal<br>Modo de desplazamiento de byte, registro R1 1<br>25 en hexadecimal                                                 | MOVW 356(R4), 25(R11)<br>Transfiere una palabra desde la dirección: 356 más el contenido de R4, a la dirección: 25 más el contenido de R11                                 |
|  | Codop de ADDI3<br>Literal corto 5<br>Modo registro R0<br>Prefijo de índice R2<br>Palabra indirecta relativa (desplazamiento desde el PC)<br>Cantidad de desplazamiento desde el PC relativo a la posición A | ADDI3 #5, R0, @ A[R2]<br>Suma 5 al entero de 32 bits contenido en R0 y almacena el resultado en la posición cuya dirección es la suma de A y de 4 veces el contenido de R2 |

Figura 10.7. Ejemplos de instrucciones del VAX.

Una instrucción del VAX comienza por un codop de 1 byte. Este es suficiente para representar la mayoría de las instrucciones. No obstante, como dispone de más de 300 instrucciones diferentes, 8 bits no son suficientes. Los códigos hexadecimales FD y FF indican un codop ampliado; el código real lo especifica un segundo byte.

El resto de la instrucción consta de hasta seis especificadores de operandos. Un especificador de operando ocupa como mínimo un byte, en el que los 4 bits de la izquierda especifican el modo de direccionamiento. La única excepción a esta regla es el modo literal, que es identificado por el patrón 00 en los dos bits más a la izquierda, quedando espacio para un literal de 6 bits. Debido a esta excepción, son un total de 12 los modos de direccionamiento que se pueden especificar.

Un especificador de operando consiste a menudo en un solo byte, en el que los cuatro bits de la derecha especifican uno de entre los 16 registros de uso general. La longitud del especificador de operando puede ampliarse en una de las dos formas siguientes: en primer lugar, un valor constante de uno o más bytes puede seguir inmediatamente al primer byte del especificador de operando. Un ejemplo es el modo de desplazamiento, en el que se emplea un desplazamiento de 8, 16 o 32 bits. En segundo lugar, puede utilizarse un modo indexado de direccionamiento. En este caso, el primer byte del especificador de operando está formado por el código de modo de direccionamiento de 4 bits 0100, y un identificador de registro índice de 4 bits. El resto del especificador de operando contiene la especificación de la dirección base, que puede a su vez ocupar uno o más bytes.

Puede que el lector se pregunte, como también lo hizo el autor, qué clase de instrucción puede requerir seis operandos. Sorprendentemente, el VAX dispone de varias de estas instrucciones. Consideremos:

ADDP6 OP1, OP2, OP3, OP4, OP5, OP6

Esta instrucción suma dos números decimales empacados. OP1 y OP2 especifican la longitud y la dirección de inicio de una de las cadenas decimales; OP3 y OP4 especifican la segunda cadena. Las dos cadenas se suman, y el resultado es memorizado como cadena decimal, cuya longitud y posición inicial vienen dadas por OP5 y OP6.

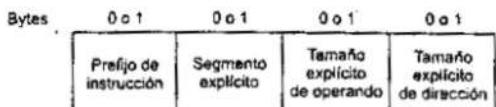
El repertorio de instrucciones del VAX ofrece una gran variedad de operaciones y de modos de direccionamiento. Esto proporciona al programador una herramienta muy potente y flexible para el desarrollo de programas. También, en teoría, debiera producir compilaciones eficientes en lenguaje máquina de programas en lenguajes de alto nivel y, en general, obtener un uso eficiente y efectivo de los recursos de la CPU. El precio a pagar por estos beneficios es un incremento en la complejidad de la CPU en comparación con un procesador que posea un repertorio de instrucciones más reducido y de formato más simple.

Volveremos sobre estas cuestiones en el Capítulo 12, donde examinaremos el caso de repertorios de instrucciones muy sencillos.

#### 10.4. FORMATOS DE INSTRUCCIONES DEL PENTIUM Y DEL PowerPC

##### FORMATOS DE INSTRUCCIÓN DEL PENTIUM II

El Pentium II está equipado con varios formatos de instrucciones. De los elementos descritos anteriormente, sólo el campo codop está siempre presente. La Figura 10.8 ilustra el formato de instrucción general. Las instrucciones se componen de entre cero y cuatro prefijos de instrucción opcionales, un codop de uno o dos bytes, una especificación de dirección opcional,



(a) Prefijo

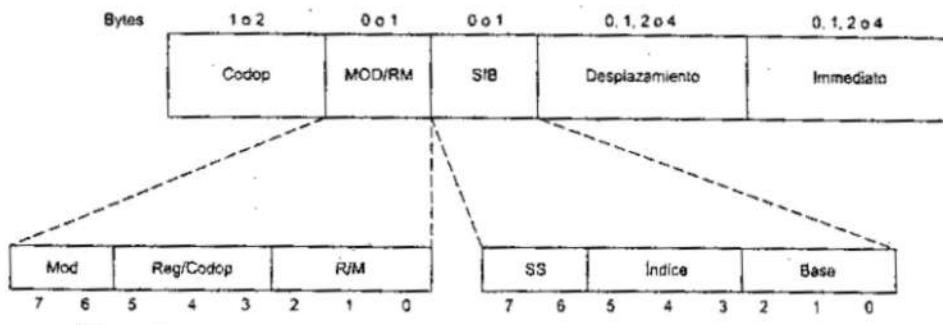


Figura 10.8. Formato de instrucción del Pentium II.

que consta del byte «Mod r/m» y del byte de índice de escala («Scale Index»), un desplazamiento opcional, y un campo inmediato opcional.

Consideremos primero los bytes de prefijo:

- Prefijos de instrucción:** El prefijo de instrucción, si está presente, contiene el prefijo LOCK o uno de los prefijos de repetición. El prefijo LOCK se emplea para asegurar el uso exclusivo de la memoria compartida en entornos multiprocesador. Los prefijos de repetición indican que se repita una operación en una cadena, lo que permite al Pentium II procesar cadenas mucho más rápido que con un bucle software habitual. Hay cinco prefijos de repetición diferentes: REP, REPE, REPZ, REPNE y REPNZ. Cuando está presente el prefijo absoluto REP, la operación que se especifica en la instrucción se ejecuta repetidas veces con los elementos sucesivos de la cadena; el número de repeticiones se especifica en el registro CX. Un prefijo REP condicional hace que se repita la instrucción hasta que la cuenta de CX llegue a cero o hasta que se cumpla la condición.
- Segmento explícito:** Especifica de forma explícita qué registro de segmento deberá utilizar una instrucción, prevaleciendo sobre la selección de registro de segmento implícito generada por el Pentium II para dicha instrucción.
- Tamaño de la dirección:** El procesador puede direccionar memoria utilizando direcciones de 16 o de 32 bits. El tamaño de la dirección define el tamaño del desplazamiento indicado en las instrucciones y el tamaño de los desplazamientos de direcciones generados durante el cálculo de la dirección efectiva. Uno de estos tamaños se considera valor implícito, y el prefijo de tamaño de la dirección commuta entre la generación de una dirección de 32 bits o una de 16 bits.
- Tamaño de operando:** De manera similar, una instrucción tiene, implícitamente, un ta-

maño de operando de 16 ó 32 bits, y el prefijo de operando comuta entre operandos de 32 o de 16 bits.

La propia instrucción incluye los siguientes campos:

- **Codop:** Codop de uno o dos bytes. El codop puede incluir también bits que indican si el operando es de 1 byte o de tamaño completo (16 ó 32 bits dependiendo del contexto), la dirección de la operación con los datos (a, o desde, memoria), y si el campo de dato inmediato debe o no extenderse con el signo.
- **Mod r/m:** Este byte y el siguiente aportan información de direccionamiento. El byte «Mod r/m» indica si un operando está en un registro o en memoria; si está en memoria, los campos del byte especifican el modo de direccionamiento a utilizar. El byte «Mod r/m» consta de tres campos. El campo Mod (2 bits) se combina con el campo r/m para formar 32 valores posibles: 8 a registro y 24 modos de indexado; el campo Reg/Codop (3 bits) especifica o bien un número de registro, o bien tres bits más de información del codop; el campo r/m (3 bits) puede especificar un registro como posición de un operando o puede formar parte de la codificación del modo de direccionamiento, en combinación con el campo Mod.
- **SIB:** Cierta codificación del byte «Mod r/m» indica la inclusión del byte SIB para especificar por completo el modo de direccionamiento. El byte SIB consta de tres campos. El campo SS (2 bits) especifica el factor de escala para el indexado escalado; el campo de índice (3 bits) especifica el registro índice; el campo de base (3 bits) especifica el registro base.
- **Desplazamiento:** Cuando el indicador de modo de direccionamiento especifica que se emplea desplazamiento, se suma un campo de desplazamiento entero con signo de 8, 16 o 32 bits.
- **Inmediato:** Proporciona el valor de un operando de 8, 16 o 32 bits.

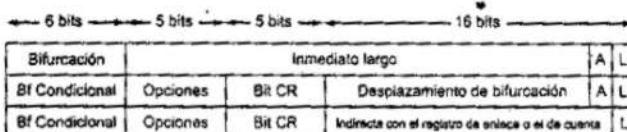
Algunas comparaciones pueden ser útiles aquí. En el formato del Pentium II, el modo de direccionamiento se da como parte de la secuencia de codop, en lugar de con cada operando. Ya que sólo un operando puede tener información de modo de dirección, en una instrucción sólo puede referenciarse un operando de memoria. En contraste, el VAX lleva la información de modo de dirección con cada operando, permitiendo operaciones de memoria a memoria. Las instrucciones del Pentium II son, por tanto, más compactas. Sin embargo, si se necesita una operación de memoria a memoria, el VAX puede efectuarla con una sola instrucción.

El formato del Pentium II permite el uso de desplazamientos para indexado, no sólo de 1 byte, sino de 2 y 4 bytes. Aunque el utilizar desplazamientos de índice mayores tiene como resultado instrucciones más largas, esta característica proporciona la flexibilidad necesaria. Por ejemplo, es útil para direccionar matrices largas o marcos de pila grandes. En contraste, el formato de instrucción del IBM S/370 permite desplazamientos no mayores de 4 Kbytes (12 bits de información para desplazamiento), y el desplazamiento debe ser positivo. Cuando una posición no se puede alcanzar con este desplazamiento, el compilador debe producir código extra para generar la dirección necesaria. Este problema se manifiesta especialmente al trabajar con marcos de pila que tengan más de 4 Kbytes de variables locales. Como se afirma en [DEWA90], «como consecuencia de la restricción, generar código para el 370 es tan penoso que ha habido incluso compiladores para el 370 que simplemente eligieron limitar el tamaño del marco de pila a 4 Kbytes».

Como puede verse, la codificación del conjunto de instrucciones del Pentium II es muy compleja. Esto se debe, en parte a la necesidad de ser compatible con el 8086, y en parte al deseo de algunos diseñadores de suministrar todas las ayudas posibles al diseñador del compilador para que produzca código eficiente. Es un tema de controversia si un repertorio de instrucciones tan complejo como éste es preferible al repertorio de un RISC.

**FORMATOS DE INSTRUCCIÓN DEL PowerPC**

Todas las instrucciones del PowerPC tienen una longitud de 32 bits, y siguen un formato regular. Los primeros 6 bits especifican la operación a efectuar. En algunos casos, hay una ampliación al codop en alguna parte de la instrucción, que especifica un caso particular de la



(a) Instrucciones de bifurcación

|    |             |            |            |                    |   |
|----|-------------|------------|------------|--------------------|---|
| CR | Bit destino | Bit fuente | Bit fuente | And, Or, Xor, etc. | / |
|----|-------------|------------|------------|--------------------|---|

(b) Instrucciones lógicas con registros de condición

|                   |              |               |                 |                           |  |
|-------------------|--------------|---------------|-----------------|---------------------------|--|
| Carga/mem. indir. | Reg. destino | Registro base | Desplazamiento  |                           |  |
| Carga/mem. indir. | Reg. destino | Registro base | Registro índice | Tamaño, signo, actualizar |  |
| Carga/mem. indir. | Reg. destino | Registro base | Desplazamiento  |                           |  |

(c) Instrucciones de carga/memorización

|                           |              |              |                           |                                  |                   |      |
|---------------------------|--------------|--------------|---------------------------|----------------------------------|-------------------|------|
| Aritmética                | Reg. destino | Reg. fuente  | Reg. fuente               | O                                | Suma, resta, etc. | R    |
| Suma, resta, etc.         | Reg. destino | Reg. fuente  | Valor inmediato con signo |                                  |                   |      |
| Lógica                    | Reg. fuente  | Reg. destino | Reg. fuente               | And, Or, Xor, etc.               | R                 |      |
| And, Or, etc.             | Reg. fuente  | Reg. destino | Valor inmediato sin signo |                                  |                   |      |
| Rotación                  | Reg. fuente  | Reg. destino | Num. despl.               | Inicio máscara                   | Fin máscara       | R    |
| Rotación o desplazamiento | Reg. fuente  | Reg. destino | Reg. fuente               | Tipo de desplazamiento o máscara |                   |      |
| Rotación                  | Reg. fuente  | Reg. destino | Num. despl.               | Máscara                          | XO                | S, R |
| Rotación                  | Reg. fuente  | Reg. destino | Reg. fuente               | Máscara                          | XO                | R    |
| Desplazamiento            | Reg. fuente  | Reg. destino | Num. despl.               | Tipo de desplazamiento o máscara |                   |      |
|                           |              |              |                           |                                  | S                 | R    |

(d) Instrucciones aritméticas con enteros, lógicas y de desplazamiento/rotación

|                |              |             |             |             |            |   |
|----------------|--------------|-------------|-------------|-------------|------------|---|
| Fiz simp/doble | Reg. destino | Reg. fuente | Reg. fuente | Reg. fuente | Suma, etc. | R |
|----------------|--------------|-------------|-------------|-------------|------------|---|

(e) Instrucciones en coma flotante

A = Absoluto o relativo al PC      \* = Sólo implementaciones de 64 bits

L = Enlace o subrutina

O = Guardar desbordamiento en XER

R = Guardar condiciones en CR1

XO = Ampliación de Codop

S = Ampliación del campo de número de desplazamientos

Figura 10.9. Formatos de instrucción del PowerPC.

operación. En la Figura 10.9, los bits de codop se representan mediante la parte sombreada de cada formato.

Observe la estructura regular de los formatos, que facilita el trabajo de las unidades de ejecución de instrucciones. Para todas las instrucciones de carga/memorización, aritméticas y lógicas, el codop viene seguido de 2 referencias a registros de 5 bits cada una, posibilitando el uso de 32 registros de uso general.

Las instrucciones de bifurcación incluyen un bit de enlace (L), que indica que la dirección efectiva de la instrucción siguiente a la de bifurcación va a ubicarse en el registro de enlace. Dos formas de la instrucción incluyen también un bit (A), que indica si el modo de direccionamiento es absoluto o relativo al PC. Para las instrucciones de salto condicional, el campo de bits CR especifica el bit del registro de condición a comprobar. El campo de opciones especifica las condiciones bajo las que va a producirse la bifurcación. Pueden especificarse las siguientes condiciones:

- Saltar siempre
- Saltar si cuenta  $\neq 0$  y la condición es falsa
- Saltar si cuenta  $\neq 0$  y la condición es verdadera
- Saltar si cuenta = 0 y la condición es falsa
- Saltar si cuenta = 0 y la condición es verdadera
- Saltar si cuenta  $\neq 0$
- Saltar si cuenta = 0
- Saltar si la condición es falsa
- Saltar si la condición es verdadera

La mayoría de las instrucciones que efectúan cálculos (aritméticas, aritméticas en coma flotante y lógicas) incluyen un bit que indica si el resultado de la operación debiera grabarse en el registro de condición. Como se mostrará, esta posibilidad es útil para procesar la predicción de saltos.

Las instrucciones en coma flotante tienen campos para tres registros fuente. En muchos casos sólo se emplean dos registros fuente. Unas cuantas instrucciones implican multiplicar dos registros fuente y después sumar o restar de otro registro fuente. Se han incluido estas instrucciones compuestas por ser de uso bastante frecuente. Por ejemplo, con instrucciones de multiplicación-suma puede implementarse el producto escalar que forma parte de muchas operaciones con matrices.

## 10.5. LECTURAS RECOMENDADAS

Las referencias citadas en el Capítulo 9 son igualmente aplicables para el material de este capítulo. En [BLAA97] se incluye una descripción detallada de formatos de instrucciones y modos de direccionamiento. Además, puede que al lector le interese consultar [FLYN85] para una discusión y análisis sobre aspectos de diseño de repertorios de instrucciones, particularmente aquellos relativos a los formatos.

BLAA97 Blaauw, G., y Brooks, F. *Computer Architecture: Concepts and Evolution*. Reading, MA: Addison-Wesley, 1997.

FLYN85 Flynn, M.; Johnson, J.; y Wakefield, S. «On Instruction Sets and Their Formats.» *IEEE Transactions on Computers*, March, 1985.

**10.6. PROBLEMAS**

- 10.1. Justifique la afirmación de que una instrucción de 32 bits es probablemente mucho menos del doble de útil que una de 16 bits.
- 10.2. Dados los valores de memoria siguientes, y suponiendo una máquina con instrucciones de una sola dirección con un acumulador, ¿qué valores cargan las siguientes instrucciones en el acumulador?
  - La palabra 20 contiene 40.
  - La palabra 30 contiene 50.
  - La palabra 40 contiene 60.
  - La palabra 50 contiene 70.
    - a) CARGA INMEDIATA 20.
    - b) CARGA DIRECTA 20.
    - c) CARGA INDIRECTA 20.
    - d) CARGA INMEDIATA 30.
    - e) CARGA DIRECTA 30.
    - f) CARGA INDIRECTA 30.
- 10.3. Suponga que la dirección almacenada en el contador de programa se designa con el símbolo  $X_1$ . La instrucción almacenada en  $X_1$  tiene una parte de dirección (referencia a operando)  $X_2$ . El operando que se necesita para ejecutar la instrucción se almacena en la palabra de memoria con dirección  $X_3$ . Un registro de índice contiene el valor  $X_4$ . ¿Qué relación existe entre estas cantidades si el modo de direccionamiento de la instrucción es (a) directo, (b) indirecto, (c) relativo al PC, (d) indexado?
- 10.4. Una instrucción de bifurcación con modo relativo al PC está almacenada en la posición de memoria  $620_{10}$ . El salto se efectúa a la posición  $530_{10}$ . El campo de dirección de la instrucción es de 10 bits. ¿Cuál es el valor binario de la instrucción?
- 10.5. ¿Cuántas veces necesita la CPU referenciar a memoria cuando capta y ejecuta una instrucción con modo de direccionamiento indirecto, si dicha instrucción es: (a) un cálculo que requiere de un solo operando; (b) un salto?
- 10.6. El IBM 370 no permite direccionamiento indirecto. Suponga que la dirección de un operando está en memoria principal. ¿Cómo podría accederse al operando?
- 10.7. ¿Por qué fue drástica la decisión de IBM de pasar de 36 a 32 bits por palabra? ¿Para quién lo fue?
- 10.8. En [COOK82], el autor propone que los modos relativos al PC se eliminan en favor de otros modos, tales como el uso de una pila. ¿Cuál es la desventaja de esta propuesta?
- 10.9. Suponga un repertorio de instrucciones que utiliza una longitud de instrucción de 16 bits. Los operandos se especifican con 6 bits. Hay  $K$  instrucciones de dos operandos y  $L$  instrucciones de cero operandos. ¿Cuál es el número máximo de instrucciones de un operando que pueden permitirse?

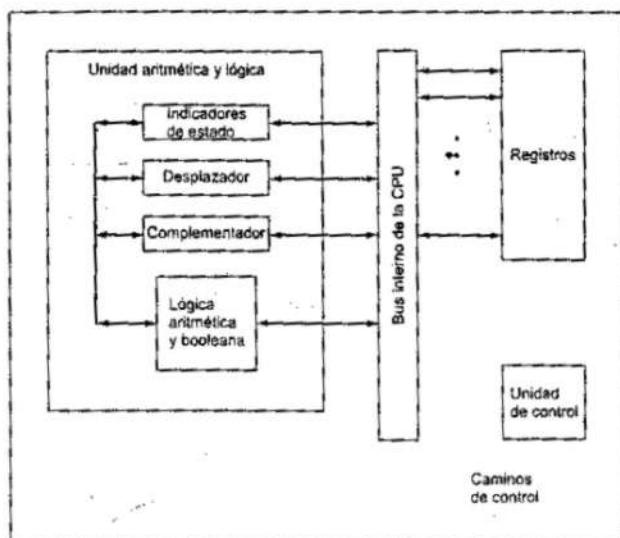
- 10.10. Diseñe un código de operación de longitud variable, para poder codificar todas las instrucciones siguientes con una longitud de instrucción de 36 bits:
- Instrucciones con dos direcciones de 15 bits y un número de registro de 3 bits.
  - Instrucciones con una dirección de 15 bits y un número de registro de 3 bits.
  - Instrucciones sin direcciones ni registros.
- 10.11. Considere los resultados del Problema 9.3. Suponga que M es una dirección de memoria de 16 bits y que X, Y, y Z son direcciones de 16 bits, o bien números de registros de 4 bits. La máquina de una dirección emplea un acumulador, y las máquinas de 2 y 3 direcciones tienen 16 registros e instrucciones, que operan con todas las combinaciones de posiciones de memoria y de registros. Suponiendo codops de 8 bits y longitudes de instrucción múltiplos de 4 bits, ¿cuántos bits necesita cada máquina para calcular X?
- 10.12. ¿Hay alguna justificación posible para una instrucción con dos codops?
- 10.13. El Pentium II incluye la siguiente instrucción:  
`IMUL op1, op2, immediate`  
Esta instrucción multiplica op2, que puede ser registro o memoria, por un valor de operando inmediato, y guarda el resultado en op1, que debe ser un registro. No hay otra instrucción con tres operandos de este tipo en el repertorio. ¿Qué posible uso tiene esta instrucción? (Nota: considere el indexado.)



## CAPÍTULO 11

# Estructura y función de la CPU

- 11.1. Organización del procesador**
- 11.2. Organización de los registros**
  - Registros visibles para el usuario
  - Registros de control y de estado
  - Ejemplos de organizaciones de registros de microprocesadores
- 11.3. Ciclo de instrucción**
  - El ciclo indirecto
  - Flujo de datos
- 11.4. Segmentación de instrucciones**
  - Estrategia de segmentación
  - Prestaciones de un cauce segmentado
  - Tratamiento de saltos
  - Segmentación del Intel 80486
- 11.5. El procesador Pentium**
  - Organización de los registros
  - Procesamiento de interrupciones
- 11.6. El procesador PowerPC**
  - Organización de los registros
  - Procesamiento de interrupciones
- 11.7. Lecturas recomendadas**
- 11.8. Problemas**



- Un procesador incluye registros visibles para el usuario y registros de control/estado. Los primeros pueden referenciarse, implícita o explícitamente, en las instrucciones máquina. Los registros visibles para el usuario pueden ser de uso general, o tener una utilidad especial, tal como almacenamiento de números en coma fija o coma flotante, direcciones, índices o punteros de segmento. Los registros de control y de estado se usan para controlar el funcionamiento de la CPU. Un ejemplo obvio es el contador de programa. Otro ejemplo importante es la palabra de estado del programa (PSW, program status word), que contiene diversos bits de estado y condición. Éstos incluyen bits para reflejar el resultado de la operación aritmética más reciente, bits de habilitación de interrupciones y un indicador de si la CPU funciona en modo supervisor o usuario.
- Los procesadores utilizan la segmentación de instrucciones para acelerar la ejecución. Fundamentalmente, la segmentación de cauce supone dividir el ciclo de instrucción en varias etapas separadas que operan secuencialmente, tales como captación de instrucción, decodificación de instrucción, y escritura del operando resultado. Las instrucciones se mueven a través de estas etapas como en una cadena de montaje, de modo que, en principio, cada etapa puede estar trabajando en una instrucción diferente al mismo tiempo. La existencia de saltos y dependencias entre instrucciones complica el diseño y el uso de los cauces segmentados.

Este capítulo trata aspectos del procesador no cubiertos en la Parte III, y establece el escenario para la discusión de los RISC y de la arquitectura superescalar en los Capítulos 12 y 13.

Comenzamos con un resumen de la organización del procesador. Después analizaremos los registros, que constituyen la memoria interna del procesador. Estaremos entonces en condiciones de volver a la discusión (comenzada en la Sección 3.2) sobre el ciclo de instrucción. Completa nuestro estudio una descripción del ciclo de instrucción y de una técnica usual conocida como segmentación («pipelining») de instrucciones. El capítulo concluye con un examen de algunos aspectos adicionales de las organizaciones del Pentium II y del PowerPC.

## 11.1. ORGANIZACIÓN DEL PROCESADOR

Para comprender la organización de la CPU, consideremos los requisitos fijados para la CPU, las cosas que debe hacer:

- **Captar instrucción:** La CPU lee una instrucción de la memoria.
- **Interpretar instrucción:** La instrucción se decodifica para determinar qué acción es necesaria.
- **Captar datos:** La ejecución de una instrucción puede exigir leer datos de la memoria o de un módulo de E/S.
- **Procesar datos:** La ejecución de una instrucción puede exigir llevar a cabo alguna operación aritmética o lógica con los datos.
- **Escribir datos:** Los resultados de una ejecución pueden exigir escribir datos en la memoria o en un módulo de E/S.

Para hacer estas cosas, es obvio que la CPU necesita almacenar algunos datos temporalmente. Debe recordar la posición de la última instrucción, de forma que pueda saber adónde ir a buscar la siguiente. Necesita almacenar instrucciones y datos temporalmente mientras una instrucción está ejecutándose. En otras palabras, la CPU necesita una pequeña memoria interna.

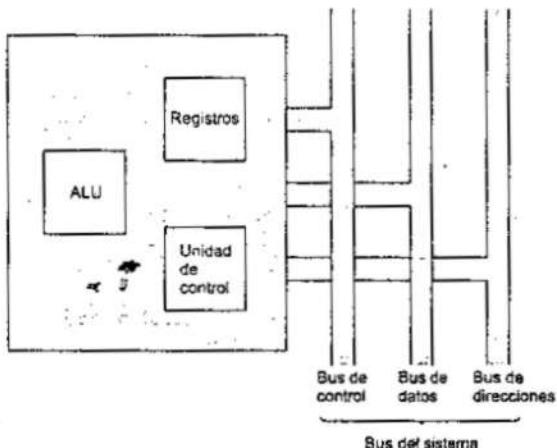


Figura 11.1. La CPU y el bus del sistema.

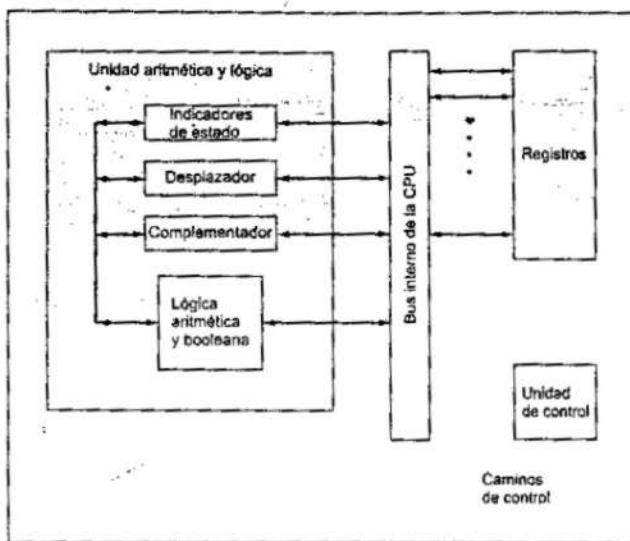


Figura 11.2. Estructura interna de la CPU.

La Figura 11.1 es una visión simplificada de la CPU, que indica su conexión con el resto del sistema a través del bus del sistema. Una interfaz similar será necesaria para cualquiera de las estructuras de interconexión descritas en el Capítulo 3. El lector recordará que los principales componentes de la CPU son una *unidad aritmético lógica* (ALU, arithmetic and logic unit) y una *unidad de control* (CU, control unit). La ALU lleva a cabo el verdadero cálculo o procesamiento de datos. La unidad de control controla las transferencias de datos hacia dentro y hacia fuera de la CPU, y el funcionamiento de la ALU. Además, la figura muestra una memoria interna mínima, que consta de un conjunto de posiciones de almacenamiento, llamadas *registros*.

La Figura 11.2 presenta una visión un poco más detallada de la CPU. Se indican los caminos de transferencia de datos y de la lógica de control, que incluyen un elemento con el rótulo *bus interno de la CPU*. Este elemento es necesario para transferir datos entre los diversos registros y la ALU, ya que la ALU, en realidad, sólo opera con datos de la memoria interna de la CPU. La figura muestra también los elementos básicos típicos de la ALU. Observe la similitud entre la estructura interna del computador en su totalidad y la estructura interna de la CPU. En ambos casos hay una pequeña colección de elementos principales (computador: CPU, E/S, memoria; CPU: unidad de control, ALU, registros) conectados por caminos de datos.

## 11.2. ORGANIZACIÓN DE LOS REGISTROS

Como discutimos en el Capítulo 4, un computador emplea una jerarquía de memoria. En los niveles más altos de la jerarquía, la memoria es más rápida, más pequeña y más cara (por bit). Dentro de la CPU hay un conjunto de registros que funciona como un nivel de memoria,

por encima de la memoria principal y de la cache en la jerarquía. Los registros de la CPU son de dos tipos:

- **Registros visibles para el usuario:** Permiten al programador de lenguaje máquina o ensamblador, minimizar las referencias a memoria principal cuando optimiza el uso de registros.
- **Registros de control y de estado:** Son utilizados por la unidad de control para controlar el funcionamiento de la CPU, y por programas privilegiados del sistema operativo para controlar la ejecución de programas.

No hay una separación bien definida de registros dentro de estas dos categorías. Por ejemplo, en algunas máquinas el contador de programa es visible para el usuario (por ejemplo, en el VAX), pero en muchas no lo es. Para el objetivo de la siguiente discusión, no obstante, usaremos estas categorías.

## REGISTROS VISIBLES PARA EL USUARIO

Un registro visible para el usuario es uno que puede ser referenciado por medio del lenguaje máquina que ejecuta la CPU. Podemos clasificarlos en las siguientes categorías:

- Uso general
- Datos
- Direcciones
- Códigos de condición

Los registros de uso general pueden ser asignados por el programador a diversas funciones. A veces, su uso dentro del repertorio de instrucciones es ortogonal a la operación. Es decir, cualquier registro de uso general puede contener el operando para cualquier código de operación. Esto proporciona una utilización de registros de auténtico uso general. Con frecuencia, sin embargo, existen restricciones. Por ejemplo, puede haber registros específicos para operaciones en coma flotante y operaciones de pila.

En algunos casos, los registros de uso general pueden ser utilizados para funciones de direccionamiento (por ejemplo, indirecto por medio de registro, con desplazamiento). En otros casos hay una separación parcial o total entre registros de datos y registros de direcciones. Los registros de datos pueden usarse únicamente para contener datos, y no se pueden emplear en el cálculo de una dirección de operando. Los registros de dirección pueden ser de uso más o menos general, o pueden estar dedicados a un modo de direccionamiento particular. Entre otros, se pueden citar los siguientes ejemplos:

- **Punteros de segmento:** En una máquina con direccionamiento segmentado (véase Sección 7.3) un registro de segmento contiene la dirección de la base del segmento. Puede haber múltiples registros: por ejemplo, uno para el sistema operativo y otro para el proceso actual.
- **Registros índice:** Se usan para direccionamiento indexado, y pueden ser autoindexados.
- **Puntero de pila:** Si existe direccionamiento a pila visible al usuario, la pila está normalmente en memoria, y hay un registro dedicado que apunta a la cabecera de ésta. Esto permite un direccionamiento implícito; es decir, apilar («push»), desapilar («pop»), y otras instrucciones de la pila que no necesitan contener un operando explícito referente a ella.

Hay aquí varias cuestiones de diseño a estudiar. Una importante, es si usar registros de uso completamente general o si especializar su uso. Nos referimos ya a este asunto en el capítulo anterior, dado que afecta al diseño del repertorio de instrucciones. Con el uso de registros

especializados, generalmente puede quedar implícito en el código de operación a qué tipo de registro se refiere un determinado campo de operando. El campo de operando sólo identifica uno de entre un conjunto de registros especializados, en lugar de uno de entre dos los registros, lo cual ahorra bits. Por otra parte, esta especialización limita la flexibilidad del programador. No hay una óptima y definitiva solución a este problema de diseño, pero como se mencionó, la tendencia parece ir hacia el uso de registros especializados.

Otro problema de diseño es el número de registros de uso general o de datos más direcciones que tienen que incluirse. De nuevo, esto afecta al diseño del repertorio de instrucciones, ya que más registros requieren más bits para el campo de operando. Como discutimos anteriormente, parece óptimo alrededor de entre 8 y 32 registros [LUND77]. Menos registros se traducen en más referencias a memoria; más registros no reducen notablemente las referencias a memoria (como ejemplo, véase [WILL90]). Sin embargo, una nueva aproximación, que saca partido al uso de cientos de registros, se manifiesta en algunos sistemas RISC y esto se estudia en el Capítulo 12.

Por último, está la cuestión de la longitud de los registros. Los registros que han de contener direcciones, han de ser lo suficientemente grandes como para albergar la dirección más grande. Los registros de datos deben ser capaces de contener valores de la mayoría de tipos de datos. Algunas máquinas permiten que dos registros contiguos sean usados como uno solo para contener valores de doble longitud.

Una última categoría de registros, que es al menos parcialmente visible al usuario, contiene códigos de condición (también llamados «indicadores» o «flags»). Los códigos de condición son bits fijados por el hardware de la CPU como resultado de alguna operación. Por ejemplo, una operación aritmética puede producir un resultado positivo, negativo, nulo, o de desbordamiento. Además de almacenarse el propio resultado en un registro o en la memoria se obtiene también un código de condición. El código puede ser examinado con posterioridad, como parte de una operación de bifurcación condicional.

Los bits de códigos de condición se reúnen en uno o más registros. Normalmente forman parte de un registro de control. Por lo general, las instrucciones máquina permiten que estos bits sean leídos por referencia implícita, pero no pueden ser alterados por el programador.

En algunas máquinas, una llamada a una subrutina dará lugar a la salvaguarda automática de todos los registros visibles por el usuario, que serán restablecidos en el retorno de la subrutina. La CPU lleva a cabo la salvaguarda y restablecimiento como parte de la ejecución de las instrucciones de llamada y retorno, respectivamente. Esto permite a cada subrutina usar independientemente los registros visibles por el usuario. En otras máquinas, es responsabilidad del programador guardar los contenidos de los registros visibles para el programador relevantes, antes de la llamada a la subrutina, teniendo que incluir en el programa instrucciones para este fin.

## REGISTROS DE CONTROL Y DE ESTADO

Hay diversos registros de la CPU que se emplean para controlar su funcionamiento. La mayoría de ellos, en la mayor parte de las máquinas, no son visibles para el usuario. Algunos de ellos pueden ser visibles a instrucciones máquina ejecutadas en un modo de control o de sistema operativo.

Naturalmente, máquinas diferentes tendrán diferentes organizaciones de registros y usarán distinta terminología. Se incluye aquí una lista razonablemente completa de tipos de registros, con una breve descripción.

Son esenciales cuatro registros para la ejecución de una instrucción:

- **Contador de programa (Program Counter, PC):** contiene la dirección de la instrucción a captar.
- **Registro de instrucción (Instruction Register, IR):** contiene la instrucción captada más recientemente.
- **Registro de dirección de memoria (Memory Address Register, MAR):** contiene la dirección de una posición de memoria.
- **Registro intermedio de memoria (Memory Buffer Register, MBR):** contiene la palabra de datos a escribir en memoria, o la palabra leída más recientemente.

Normalmente, la CPU actualiza el contador de programa después de cada captación de instrucción, de manera que siempre apunta a la siguiente instrucción a ejecutar. Una instrucción de bifurcación o salto también modificará el contenido de PC. La instrucción captada se carga en IR, donde son analizados el código de operación y los campos de operando. Se intercambian datos con la memoria por medio de MAR y de MBR. En un sistema con organización de bus, MAR se conecta directamente al bus de direcciones y MBR directamente al bus de datos. Los registros visibles por el usuario intercambian repetidamente datos con MBR.

Los cuatro registros que se acaban de mencionar se usan para la transferencia de datos entre la CPU y la memoria. Dentro de la CPU, los datos tienen que ofrecerse a la ALU para su procesamiento. La ALU puede tener acceso directo a MBR y a los registros visibles para el usuario. Como alternativa, puede haber registros intermedios adicionales en torno a la ALU; estos registros sirven como registros de entrada y salida de la ALU, e intercambian datos con MBR y los registros visibles para el usuario.

Todos los diseños de CPU incluyen un registro o un conjunto de registros, conocidos a menudo como *palabra de estado del programa* (program status word, PSW), que contiene información de estado. PSW contiene normalmente códigos de condición, además de otra información de estado. Entre los campos o indicadores comunes se incluyen los siguientes:

- **Signo:** Contiene el bit de signo del resultado de la última operación aritmética.
- **Cero:** Puesto a uno cuando el resultado es 0.
- **Acarreo:** Puesto a uno si una operación da lugar a un acarreo (suma) o adeudo (resta) del bit más significativo. Se usa en operaciones aritméticas multipalabra.
- **Igual:** Puesto a uno si el resultado de una comparación lógica es la igualdad.
- **Desbordamiento:** Usado para indicar un desbordamiento aritmético.
- **Interrupciones habilitadas/inhabilitadas:** Usado para permitir o inhabilitar interrupciones.
- **Supervisor:** Indica si la CPU funciona en modo supervisor o usuario. Únicamente en modo supervisor se pueden ejecutar ciertas instrucciones privilegiadas y se puede acceder a ciertas áreas de memoria.

En algún diseño concreto de CPU es posible encontrar otros registros relativos a estado y control. Además de PSW, puede existir un puntero a un bloque de memoria que contenga información de estado adicional (por ejemplo, bloques de control de procesos). En las máquinas que usan interrupciones vectorizadas puede existir un registro de vector de interrupción. Si se utiliza una pila para llevar a cabo ciertas funciones (por ejemplo, llamada a subrutina), se necesita un puntero de pila del sistema. En un sistema de memoria virtual se usa un puntero a la tabla de páginas. Por último, pueden emplearse registros para el control de operaciones de E/S.

En el diseño de la organización de los registros de control y estado entran en juego varios factores. Una cuestión importante es el soporte del sistema operativo. Algunos tipos de infor-

mación de control son de utilidad específica para el sistema operativo. Si el diseñador de CPU posee una comprensión funcional del sistema operativo que se va a utilizar, la organización de los registros puede adaptarse hasta cierto punto a ese sistema operativo.

Otra decisión clave en el diseño es la distribución de información de control entre registros y memoria. Es frecuente dedicar los primeros (más bajos) pocos cientos o miles de palabras de memoria para fines de control. El diseñador debe decidir cuánta información de control debiera estar en registros y cuánta en memoria. Surge el habitual compromiso entre coste y velocidad.

### EJEMPLOS DE ORGANIZACIONES DE REGISTROS DE MICROPROCESADORES

Resulta instructivo examinar y comparar las organizaciones de registros de sistemas análogos. En esta sección, examinamos dos microprocesadores de 16 bits que fueron diseñados aproximadamente al mismo tiempo: el Motorola MC68000 [STR79] y el Intel 8086 [MORS78]. Las Figuras 11.3a y 11.3b representan la organización de registros de cada uno de ellos; los registros puramente internos, tales como el registro de dirección de memoria, no se muestran.

El MC68000 distribuye sus registros de 32 bits en ocho de datos y nueve de dirección. Los ocho registros de datos se usan principalmente para manipulación de datos, y también se usan en direccionamiento como registros índice. El ancho de los registros permite operaciones con datos de 8, 16 y 32 bits, según determine el código de operación. Los registros de direcciones contienen direcciones de 32 bits (no hay segmentación); dos de estos registros se usan también como punteros de pila, uno para los usuarios y el otro para el sistema operativo, dependiendo del modo de ejecución en curso. Los dos registros se referencian como 7, dado que sólo uno de ellos se puede usar en un instante dado. El MC68000 también incluye un contador de programa de 32 bits y un registro de estado de 16 bits.

El equipo de Motorola quiso un repertorio de instrucciones muy regular, sin registros de uso especial. Su interés por la eficiencia del código los condujo a dividir los registros en dos componentes funcionales, ahorrando un bit en cada campo de especificación de registro. Esto parece un compromiso razonable entre generalidad total y compactad del código.

El Intel 8086 usa un enfoque diferente para la organización de los registros. Cada uno de los registros tiene un uso especial, aunque algunos registros se pueden emplear también para un uso general. El 8086 contiene cuatro registros de datos de 16 bits, que son direccionables como registros de 16 bits o como registros de bytes, y cuatro registros punteros e índices de 16 bits. Los registros de datos pueden utilizarse como registros de uso general en algunas instrucciones. En otras, los registros se usan implícitamente. Por ejemplo, una instrucción de multiplicación siempre usa el acumulador. Los cuatro registros punteros se usan también implícitamente en algunas operaciones; cada uno contiene un desplazamiento dentro de un segmento. Hay también cuatro registros de segmento de 16 bits. Tres de los cuatro registros de segmento se usan de una forma dedicada e implícita para apuntar al segmento de la instrucción en curso (útil para instrucciones de salto), a un segmento que contenga datos y a un segmento que contenga una pila, respectivamente. Estos usos dedicados e implícitos proporcionan una codificación compacta con el coste de una flexibilidad reducida. El 8086 incluye también un puntero de instrucción y un conjunto de indicadores de estado y control de 1 bit.

Debe quedar claro qué es lo significativo de esta comparación. No hay, por el momento, una filosofía universalmente aceptada sobre la mejor forma de organizar los registros de la CPU [TOON81]. Igual que ocurre en el diseño global del repertorio de instrucciones y en algunos otros aspectos del diseño de la CPU, se trata más bien de una cuestión de opinión y gustos.

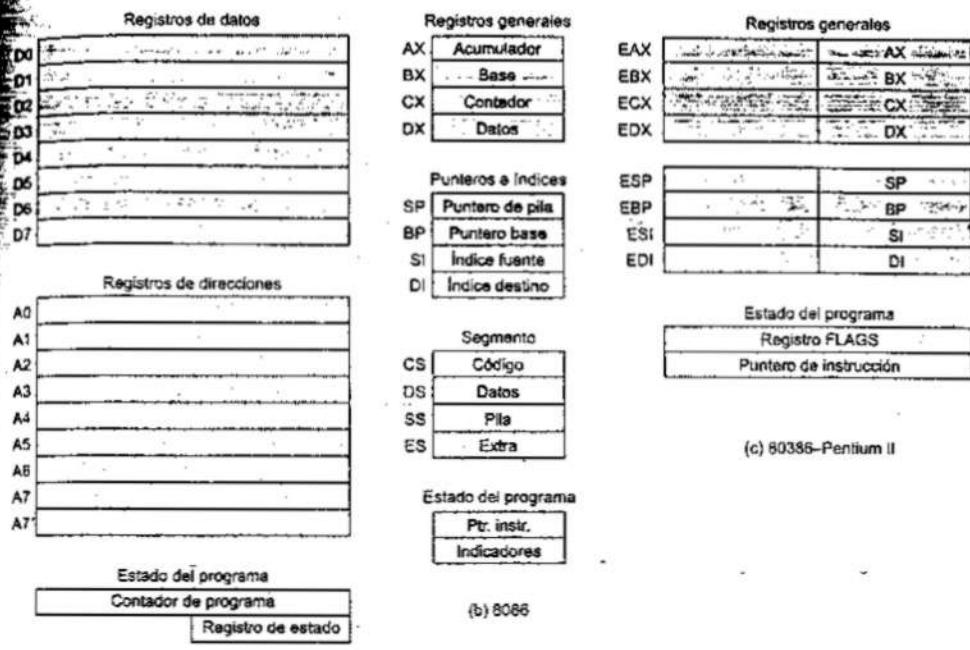


Figura 11.3. Ejemplos de organizaciones de registros de microprocesadores.

En la Figura 11.3c se ilustra un segundo aspecto instructivo acerca del diseño de la organización de los registros. Esta figura muestra la organización de los registros visibles por el usuario en el Intel 80386 [ELAY85], que es un microprocesador de 32 bits diseñado como una ampliación del 8086<sup>1</sup>. El 80386 usa registros de 32 bits. No obstante, para proporcionar compatibilidad ascendente para los programas escritos en la primera máquina, el 80386 conserva la organización de registros original integrada en la nueva organización. Dada esta restricción en el diseño, los arquitectos de los procesadores de 32 bits han limitado la flexibilidad al diseñar la organización de los registros.

### 11.3. EL CICLO DE INSTRUCCIÓN

En la Sección 3.2, describimos el ciclo de instrucción de la CPU. La Figura 11.4 repite una de las figuras usadas en esa descripción (Figura 3.9). Recordemos que un ciclo de instrucción incluye los siguientes subciclos:

- **Captación:** Llevar la siguiente instrucción de la memoria a la CPU.
- **Ejecución:** Interpretar el código de operación y llevar a cabo la operación indicada.

<sup>1</sup> Dado que el MC68000 ya usaba registros de 32 bits, el MC68020 [MACG84], que es una ampliación completa a 32 bits, usa la misma organización de registros.

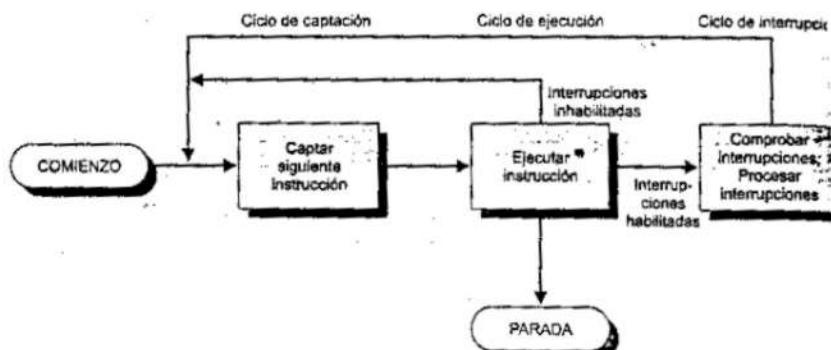


Figura 11.4. Ciclo de instrucción que incluye interrupciones.

- **Interrupción:** Si las interrupciones están habilitadas y ha ocurrido una interrupción, salvar el estado del proceso actual y atender la interrupción.

Estamos ahora en condiciones de dar alguna explicación más acerca del ciclo de instrucción. En primer lugar, debemos introducir un subciclo adicional, conocido como «ciclo indirecto».

### EL CICLO INDIRECTO

Hemos visto, en el Capítulo 10, que la ejecución de una instrucción puede involucrar a uno o más operandos en memoria, cada uno de los cuales requiere un acceso a memoria. Además, si se usa direccionamiento indirecto serán necesarios accesos a memoria adicionales.

Podemos considerar la captación de direcciones indirectas como un subciclo de instrucción más. El resultado se muestra en la Figura 11.5. La principal línea de actividad consiste

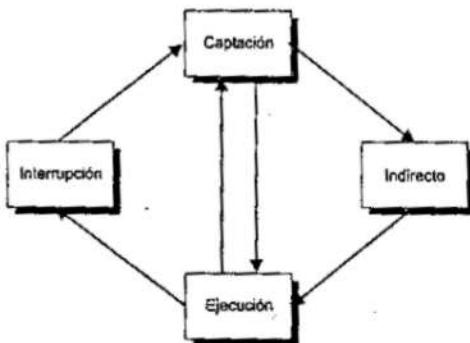


Figura 11.5. El ciclo de instrucción.



Figura 11.6. Diagrama de estados del ciclo de instrucción.

en alternar las actividades de captación y ejecución de instrucciones. Después de que una instrucción sea captada, ésta es examinada para determinar si implica algún direccionamiento indirecto. Si es así, los operandos requeridos se captan usando direccionamiento indirecto. Tras la ejecución se puede procesar una interrupción antes de la captación de la siguiente instrucción.

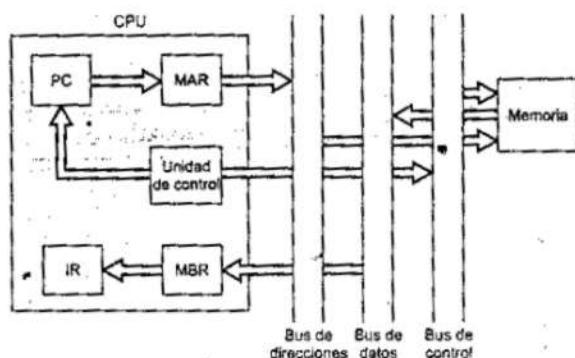
En la Figura 11.6, que es una versión revisada de la Figura 3.12, se muestra otra forma de ver este proceso. Ésta ilustra más correctamente la naturaleza del ciclo de instrucción. Una vez que una instrucción es captada, deben identificarse sus campos de operando. Se capta entonces de la memoria cada operando de entrada, pudiendo requerir este proceso direccionamiento indirecto. Los operandos ubicados en registros no necesitan ser captados. Una vez que se ejecuta la operación, puede ser necesario un proceso similar para almacenar el resultado en la memoria principal.

## FLUJO DE DATOS

La secuencia exacta de eventos que tienen lugar durante un ciclo de instrucción depende de diseño de la CPU. Podemos, no obstante, indicar qué debe ocurrir en términos generales. Asumamos una CPU que emplee un registro de dirección de memoria (MAR), un registro intermedio de memoria (MBR), un contador de programa (PC) y un registro de instrucción (IR).

Durante el *ciclo de captación* se lee una instrucción de la memoria. La Figura 11.7 muestra el flujo de datos en este ciclo. PC contiene la dirección de la siguiente instrucción que ha de captar. Esta dirección es llevada a MAR y puesta en el bus de direcciones. La unidad de control solicita una lectura de memoria, y el resultado se pone en el bus de datos, se copia al MBR y después se lleva a IR. Mientras tanto, PC se incrementa en 1 como preparación para la siguiente captación.

Una vez concluido el ciclo de captación, la unidad de control examina los contenidos de IR para determinar si contiene un campo de operando que use direccionamiento indirecto. Si



MBR = Registro intermedio de memoria  
 MAR = Registro de dirección de memoria  
 IR = Registro de instrucción  
 PC = Contador de programa

Figura 11.7. Flujo de datos, ciclo de captación.

es así, se lleva a cabo un *ciclo indirecto*. Tal como se muestra en la Figura 11.8, se trata de un ciclo simple. Los  $N$  bits más a la derecha de MBR, que contienen la dirección de referencia, se transfieren a MAR. Entonces, la unidad de control solicita una lectura de memoria para llevar la dirección del operando deseada a MBR.

Los ciclos de captación e indirecto son sencillos y predecibles. El *ciclo de ejecución* adopta muchas formas, ya que depende de cuál de las diversas instrucciones máquina esté en IR. Este ciclo puede implicar transferencias de datos entre registros, lectura o escritura de memoria o E/S, y/o la invocación de la ALU.

Del mismo modo que los ciclos de captación e indirecto, el *ciclo de interrupción* es simple y predecible (Figura 11.9). El contenido actual de PC tiene que ser salvado, de manera que la CPU pueda reanudar su actividad normal tras la interrupción. Así, el contenido de PC se

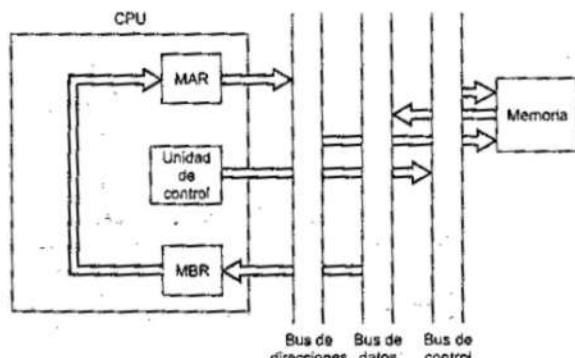


Figura 11.8. Flujo de datos, ciclo indirecto.

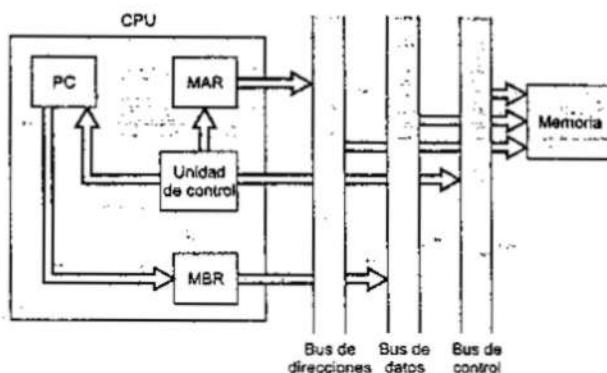


Figura 11.9. Flujo de datos, ciclo de interrupción.

transfiere a MBR para ser escrito en memoria. La posición de memoria especial reservada para este propósito se carga en MAR desde la unidad de control. Podría ser, por ejemplo, un puntero de pila. PC se carga con la dirección de la rutina de interrupción. Como resultado, el siguiente ciclo de instrucción comenzará captando la instrucción oportuna.

## SEGMENTACIÓN DE INSTRUCCIONES

A medida que los computadores evolucionan, se pueden conseguir mayores prestaciones aprovechando los progresos en la tecnología, tales como una circuitería más rápida. Los avances en la organización de la CPU también pueden mejorar las prestaciones. Hemos visto ya algunos ejemplos de esto, tales como el empleo de múltiples registros en lugar de un único acumulador, y el uso de una memoria cache. Otra aproximación referente a la organización, que es bastante común, es la segmentación de instrucciones.

### ESTRATEGIA DE SEGMENTACIÓN

La segmentación de instrucciones es similar al uso de una cadena de montaje en una fábrica de manufacturación. Una cadena de montaje saca partido del hecho de que el producto pasa a través de varias etapas de producción. Disponiendo el proceso de producción como una cadena de montaje, se puede trabajar sobre los productos en varias etapas simultáneamente. A este proceso se hace referencia como *segmentación de cauce* (pipelining), porque, como en una tubería o cauce (pipeline), en un extremo se aceptan nuevas entradas antes de que algunas entradas aceptadas con anterioridad aparezcan como salidas en el otro extremo.

Para aplicar este concepto a la ejecución de instrucciones, debemos darnos cuenta de que, de hecho, una instrucción tiene varias etapas. La Figura 11.6, por ejemplo, parte el ciclo de instrucción hasta en 10 tareas, que tienen lugar secuencialmente. Claramente, puede pensarse en la utilización de la segmentación.

Como una aproximación sencilla, consideraremos la subdivisión del procesamiento de una instrucción en dos etapas: captación de instrucción y ejecución de instrucción. Hay períodos en la ejecución de una instrucción en los que no se accede a memoria principal. Este tiempo

podría utilizarse en captar la siguiente instrucción en paralelo con la ejecución de la actual. La Figura 11.10a representa esta aproximación. El cauce tiene dos etapas independientes. La primera etapa capta una instrucción y la almacena en un buffer. Cuando la segunda etapa está libre, la primera le pasa la instrucción almacenada. Mientras que la segunda etapa ejecuta la instrucción, la primera etapa utiliza algún ciclo de memoria no usado para captar y almacenar la siguiente instrucción. A esto se llama *prebúsqueda o precaptación de instrucción* (instruction prefetch) o *solapamiento de la captación* (fetch overlap).

Debería estar claro que este proceso acelerará la ejecución de instrucciones. Si las etapas de captación y ejecución fueran de igual duración, el tiempo de ciclo de instrucción se reduciría a la mitad. Sin embargo, si miramos más atentamente a este cauce (Figura 11.10b), veremos que esta duplicación de la velocidad de ejecución es poco probable por dos razones:

1. El tiempo de ejecución será generalmente más largo que el tiempo de captación. La ejecución implicará la lectura y almacenamiento de operandos y la realización de alguna operación. Así, la etapa de captación puede tener que esperar algún tiempo antes de que pueda vaciar su buffer.
2. Una instrucción de bifurcación condicional hace que la dirección de la siguiente instrucción a captar sea desconocida. De este modo, la etapa de captación debe esperar hasta que reciba la dirección de la siguiente instrucción desde la etapa de ejecución. La etapa de ejecución puede entonces tener que esperar mientras se capta la siguiente instrucción.

La pérdida de tiempo debida a la segunda razón puede reducirse haciendo una estimación. Una regla simple es la siguiente: cuando una instrucción de bifurcación condicional pasa de la etapa de captación a la de ejecución, la etapa de captación capta la instrucción de memoria que sigue a la instrucción de salto; entonces, si el salto no se produce, no se pierde tiempo; si el salto se produce, debe desecharse la instrucción captada y captarse una nueva instrucción.

Aunque estos factores reduzcan la efectividad potencial del cauce de dos etapas, se produce alguna aceleración. Para conseguir una mayor aceleración, el cauce debe tener más etapas. Consideraremos la siguiente descomposición del procesamiento de una instrucción.

- **Captar instrucción (Fetch Instruction, FI):** Leer la supuesta siguiente instrucción en un buffer.

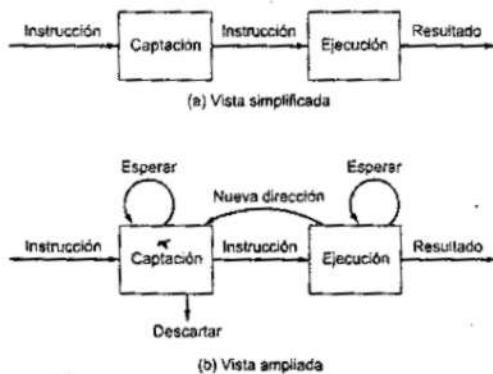


Figura 11.10. Cauce de instrucciones con dos etapas.

- **Decodificar instrucción (Decode Instruction, DI):** Determinar el código de operación y los campos de operando.
- **Calcular operandos (Calculate Operands, CO):** Calcular la dirección efectiva de cada operando fuente. Esto puede involucrar direccionamiento mediante un desplazamiento, indirecto a través de registro, indirecto, u otras formas de calcular la dirección.
- **Captar operandos (Fetch Operands, FO):** Captar cada operando de memoria. Los operandos en registros no tienen que ser captados.
- **Ejecutar instrucción (Execute Instruction, EI):** Realizar la operación indicada y almacenar el resultado, si lo hay, en la posición de operando destino especificada.
- **Escribir operando (Write Operand, WO):** Almacenar el resultado en memoria.

Con esta descomposición, las diversas etapas tendrán casi igual duración. Por motivos de claridad, asumimos igual duración. En ese caso, la Figura 11.11 muestra que un cauce de seis etapas puede reducir el tiempo de ejecución de 9 instrucciones, de 54 a 14 unidades de tiempo.

Conviene hacer algunos comentarios. El diagrama supone que cada instrucción recorre las seis etapas del cauce. No siempre se dará este caso. Por ejemplo, una instrucción de carga no necesita la etapa WO. Sin embargo, para simplificar los circuitos del cauce, la temporización se establece asumiendo que cada instrucción requiere las seis etapas. Además el diagrama supone que todas las etapas pueden funcionar en paralelo. En particular, se supone que no hay conflictos de memoria. Por ejemplo, las etapas FI, FO y WO requieren un acceso a memoria. El diagrama implica que todos estos accesos pueden tener lugar simultáneamente. La mayoría de los sistemas de memoria no permitirán esto. No obstante, el valor deseado puede estar en cache, o las etapas FO o WO pueden ser nulas. De este modo, casi siempre, los conflictos de memoria no reducirán la velocidad del cauce.

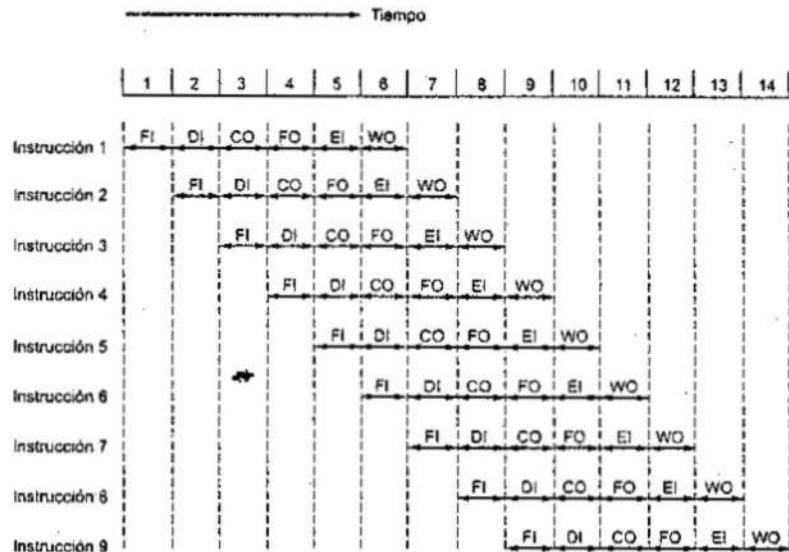


Figura 11.11. Diagrama de tiempos del funcionamiento del cauce de instrucciones.

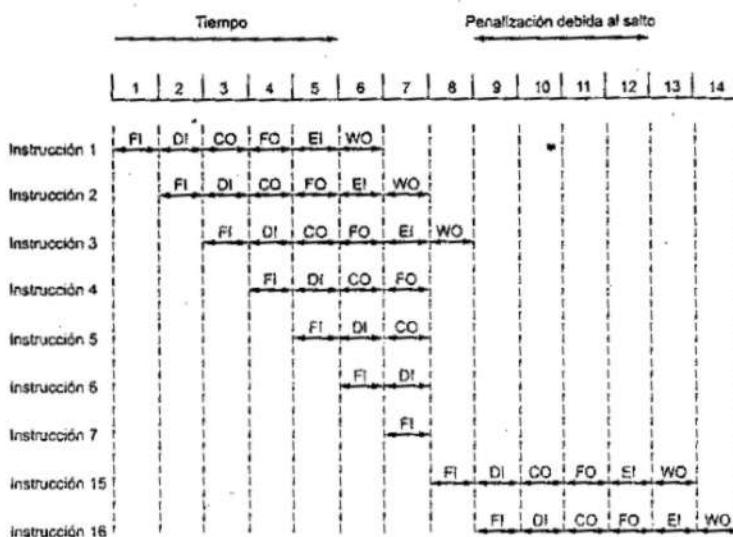


Figura 11.12. Efecto de una bifurcación condicional en el funcionamiento del cauce de instrucciones.

Algunos otros factores contribuyen a limitar la mejora de prestaciones. Si las seis etapas no son de igual duración, habrá cierta espera en algunas etapas del cauce, como se discutió antes para el cauce de dos etapas. Otra dificultad es la instrucción de bifurcación condicional, que puede invalidar varias captaciones de instrucción. Un evento impredecible similar, es la llegada de una interrupción. La Figura 11.12 ilustra los efectos de la bifurcación condicional, usando el mismo programa de la Figura 11.11. Se supone que la instrucción 3 es una bifurcación condicional a la instrucción 15. Hasta que no se ejecuta la instrucción no hay forma de saber qué instrucción vendrá a continuación. El cauce, en este ejemplo, simplemente carga la siguiente instrucción secuencialmente (instrucción 4) y continúa. En la Figura 11.11 el salto no se efectúa, y obtenemos el máximo provecho en cuanto a rendimiento de este diseño. En la Figura 11.12 se produce el salto. Éste no se determina hasta el final de la unidad de tiempo 7. En ese momento, el cauce debe limpiarse de instrucciones que no son útiles. Durante la unidad de tiempo 8, la instrucción 15 entra en el cauce. Ninguna instrucción termina durante las unidades de tiempo de la 9 a la 12; ésta es la penalización en las prestaciones sufrida por no haber podido prever el salto. La Figura 11.13 indica la lógica necesaria para considerar las bifurcaciones e interrupciones en la segmentación de cauce.

Se presentan además otros problemas que no aparecen en nuestra organización simple de dos etapas. La etapa CO puede depender del contenido de un registro que podría verse alterado por una instrucción previa que aún esté en el cauce. Podrían ocurrir otros conflictos con registros y con memoria. El sistema tiene que incluir lógica para tener en cuenta este tipo de conflictos.

Según la discusión precedente, puede parecer que cuanto mayor sea el número de etapas en el cauce, más rápida será la velocidad de ejecución. Algunos diseñadores del IBM S/360

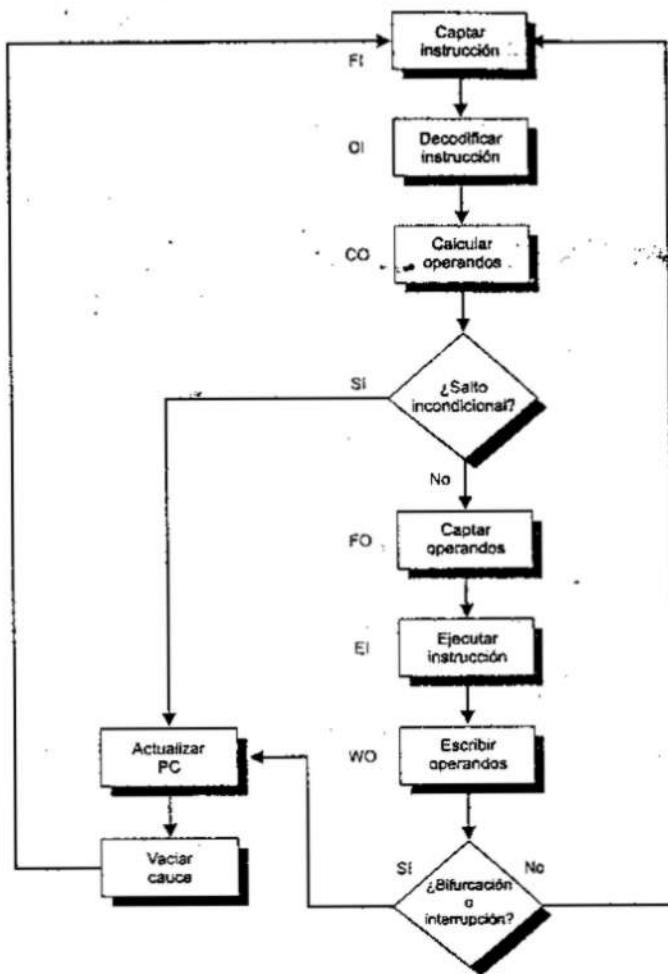


Figura 11.13. Cauce de instrucciones de una CPU, con seis etapas.

observaron dos factores que frustran este aparentemente sencillo patrón de diseño de altas prestaciones [ANDE67], y que siguen siendo ciertos hoy día:

1. En cada etapa del cauce, hay algún gasto extra debido a la transferencia de datos de buffer a buffer, y a la realización de varias funciones de preparación y distribución. Este gasto adicional puede prolongar sensiblemente el tiempo de ejecución total de una instrucción aislada. Esto es importante cuando las instrucciones secuenciales son lógicamente dependientes, bien a causa de un uso abundante de bifurcaciones, bien debido a dependencias de acceso a memoria.

2. La cantidad de lógica de control necesaria para manejar dependencias de memoria y registros, y para optimizar el uso del cauce aumenta enormemente con el número de etapas. Esto puede llevar a una situación donde la lógica para controlar el paso entre etapas sea más compleja que las etapas controladas.

La segmentación de instrucciones es una poderosa técnica para aumentar las prestaciones pero requiere un diseño cuidadoso si se quieren obtener resultados óptimos con una complejidad razonable.

## PRESTACIONES DE UN CAUCE SEGMENTADO

En esta subsección, desarrollamos algunas medidas sencillas de las prestaciones de un cauce segmentado y del incremento de velocidad relativa (basadas en la discusión de [HWAN93]). El tiempo de ciclo  $\tau$  de un cauce de instrucciones es el tiempo necesario para que un conjunto de instrucciones avance una etapa a través del cauce; cada columna de las Figuras 11.11 y 11.12 representa un tiempo de ciclo. El tiempo de ciclo puede determinarse como

$$\tau = \max[\tau_i] + d = \tau_m + d \quad i, 1 \leq i \leq k$$

donde:

$\tau_m$  = máximo retardo de etapa (retardo a través de la etapa que experimenta el mayor retardo).

$k$  = número de etapas del cauce de instrucciones.

$d$  = retardo de tiempo de un registro *latch*, necesario para que avancen las señales y datos de una etapa a la siguiente.

En general, el retardo de tiempo  $d$  es equivalente a un pulso de reloj y  $\tau_m \gg d$ . Suponga ahora que se procesan  $n$  instrucciones, sin saltos. El tiempo total  $T_k$  necesario para ejecutar las  $n$  instrucciones es:

$$T_k = [k + (n - 1)]\tau \quad (11.1)$$

Se requiere un total de  $k$  ciclos para completar la ejecución de la primera instrucción, y las  $n - 1$  instrucciones restantes requieren  $n - 1$  ciclos<sup>2</sup>. Esta ecuación se puede verificar fácilmente en la Figura 11.11. La novena instrucción se completa en el ciclo de tiempo 14:

$$14 = [6 + (9 - 1)]$$

El factor de aceleración para el cauce de instrucciones segmentado, comparado con la ejecución sin segmentar se define como

$$S_k = \frac{T_1}{T_k} = \frac{nkr}{[k + (n - 1)]\tau} = \frac{nk}{k + (n - 1)} \quad (11.2)$$

La Figura 11.14a representa el factor de aceleración en función del número de instrucciones que se ejecutan sin saltos. Como cabría esperar, en el límite ( $n \rightarrow \infty$ ) la velocidad se incrementa  $k$  veces. La Figura 11.14b muestra el factor de aceleración en función del número de etapas en el cauce de instrucciones<sup>3</sup>. En este caso, el factor de aceleración se aproxima al número de instrucciones que pueden introducirse en el cauce sin saltos. De este modo, cuanto

<sup>2</sup> Aquí no estamos siendo muy rigurosos. El tiempo de ciclo solamente igualará el valor de  $\tau$  cuando todas las etapas estén llenas. Al principio, el tiempo de ciclo será menor para el primer o los primeros ciclos.

<sup>3</sup> Observe que el eje  $x$  es logarítmico en la Figura 11.14a y lineal en la Figura 11.14b.

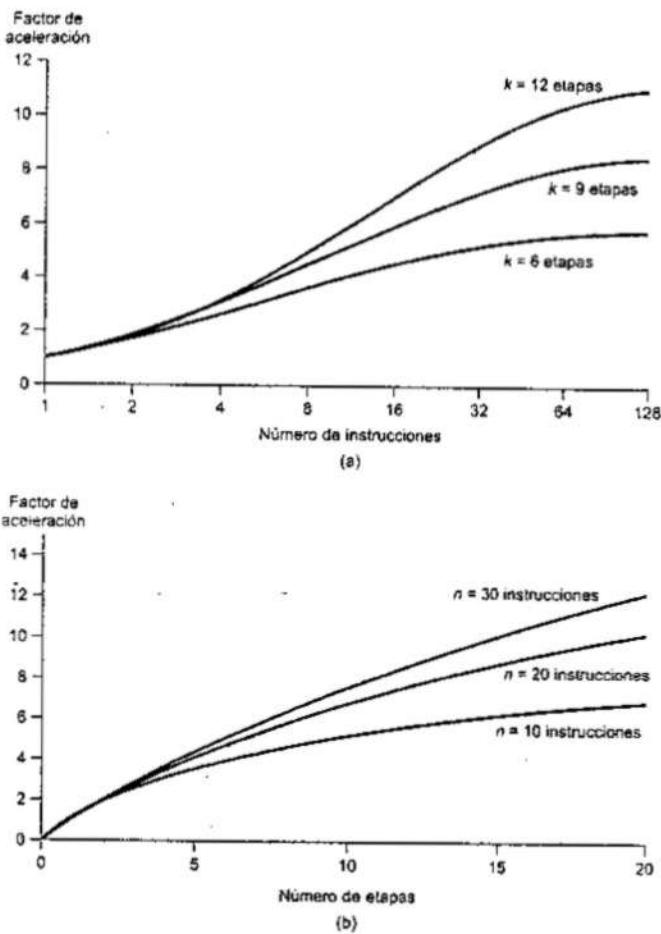


Figura 11.14. Factores de aceleración en segmentación de cauces de instrucciones.

mayor es el número de etapas del cauce, mayor es su potencial para conseguir aceleración. Sin embargo, una cuestión práctica es que los beneficios potenciales de etapas adicionales en el cauce se contrarrestan por los incrementos en coste, los retardos entre etapas, y el hecho de que se encontrarán saltos que requieran vaciar el cauce. Algo más allá del rango de 6 a 9 etapas, son contraproducentes incrementos adicionales en el número de etapas.

## TRATAMIENTO DE SALTOS

Uno de los mayores problemas del diseño de un cauce de instrucciones, es asegurar un flujo estable de instrucciones a las etapas iniciales del cauce. El principal obstáculo, como hemos

visto, es la instrucción de bifurcación condicional. Hasta que la instrucción no se ejecuta realmente, es imposible determinar si el salto se producirá o no.

Se han considerado varias aproximaciones en el tratamiento de bifurcaciones condicionales:

- Flujos múltiples
- Precaptar el destino del salto
- Buffer de bucles
- Predicción de saltos
- Salto retardado

### Flujos múltiples

Un cauce simple sufre penalización por las instrucciones de bifurcación, porque debe escoger una de las dos instrucciones a captar a continuación, y puede hacer la elección equivocada. Una solución burda es duplicar las partes iniciales del cauce y dejar que éste capte las dos instrucciones utilizando los dos caminos. Esta aproximación tiene dos problemas:

- Con cauces múltiples hay retardos debidos a la competencia por el acceso a los registros y a la memoria.
- Pueden entrar en el cauce (en cualquiera de los dos flujos) instrucciones de bifurcación adicionales antes de que se resuelva la decisión de la bifurcación original. Cada una de esas instrucciones exige un flujo adicional.

A pesar estos inconvenientes, esta estrategia puede aumentar las prestaciones. El IBM 370/168 y el IBM 3033 son ejemplos de máquinas con dos o más flujos en el cauce.

### Precaptar el destino del salto

Cuando se identifica una instrucción de bifurcación condicional, se precapta la instrucción destino del salto, además de la siguiente a la de bifurcación. Se guarda entonces esta instrucción hasta que se ejecute la instrucción de bifurcación. Si se produce el salto, el destino ya habrá sido precaptado.

El IBM 360/91 usa este método.

### Buffer de bucles

Un buffer de bucles es una memoria pequeña de gran velocidad, gestionada por la etapa de captación de instrucción del cauce, que contiene, secuencialmente, las  $n$  instrucciones captadas más recientemente. Si se va a producir un salto, el hardware comprueba en primer lugar si el destino del salto está en el buffer. En ese caso, la siguiente instrucción se capta del buffer. El buffer de bucles tiene tres utilidades:

1. Con el uso de precaptación, el buffer de bucles se anticipa, almacenando algunas instrucciones que secuencialmente están después de la dirección de donde se capta la instrucción actual. De este modo, las instrucciones que se captan secuencialmente estarán disponibles sin el tiempo de acceso a memoria habitual.
2. Si ocurre un salto a un destino a sólo unas pocas posiciones más allá de la dirección de la instrucción de bifurcación, el destino ya estará en el buffer. Esto es útil para el caso bastante común de las secuencias IF-THEN e IF-THEN-ELSE.

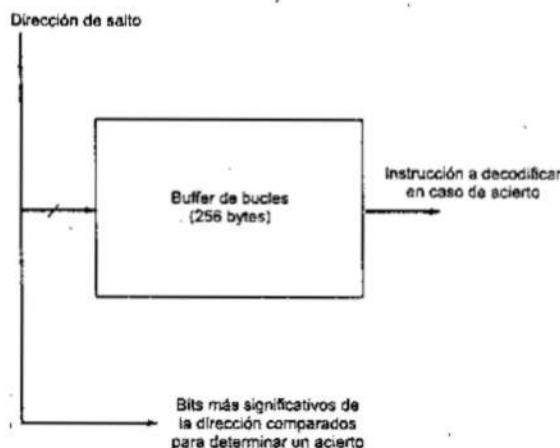


Figura 11.15. Buffer de bucles.

3. Esta estrategia se acomoda particularmente bien al tratamiento de bucles, o iteraciones, de ahí el nombre de *buffer de bucles*. Si el buffer de bucles es lo suficientemente grande como para contener todas las instrucciones de un bucle, entonces esas instrucciones sólo necesitan ser captadas de la memoria una vez, durante la primera iteración. En las siguientes iteraciones, todas las instrucciones necesarias se encuentran ya en el buffer.

El buffer de bucles es similar, en principio, a una cache de instrucciones. Las diferencias son que el buffer de bucles sólo guarda instrucciones consecutivas, y que es mucho más pequeño en tamaño y, por tanto, de menor coste.

La Figura 11.15 ofrece un ejemplo de un buffer de bucles. Si el buffer contiene 256 bytes, y se usa direccionamiento a bytes, los 8 bits menos significativos se usan para indexar el buffer. Los restantes bits más significativos se examinan para determinar si el destino del salto cae dentro del entorno capturado por el buffer.

Entre las máquinas que usan un buffer de bucles se encuentran algunas de las máquinas CDC (Star-100, 6600, 7600) y el CRAY-1. Una forma especializada de buffer de bucles está disponible en el 68010, para ejecutar un bucle de tres instrucciones involucrado en la instrucción DBcc (decrementar y saltar si se cumple la condición) (véase Problema 11.6). Se mantiene un buffer de tres palabras, y el procesador ejecuta repetidamente estas instrucciones hasta que se satisface la condición del bucle.

### Predicción de saltos

Se pueden usar varias técnicas para predecir si un salto se va a producir. Entre las más usuales se encuentran las siguientes:

- Predecir que nunca se salta.
- Predecir que siempre se salta.
- Predecir según el código de operación.

- Comutador saltar/no saltar.
- Tabla de historia de saltos.

Las tres primeras soluciones son estáticas: no dependen de la historia de la ejecución que haya tenido lugar hasta la instrucción de bifurcación condicional. Las dos últimas aproximaciones son dinámicas: dependen de la historia de la ejecución.

Las dos primeras soluciones son las más simples. Una asume que el salto no se producirá y continuará captando instrucciones secuencialmente, y la otra que el salto se producirá y siempre captará la instrucción destino del salto. El 68020 y el VAX 11/780 usan la aproximación de predecir que nunca se salta. El VAX 11/780 incluye además una forma de minimizar el efecto de una decisión errónea. Si la captación de la instrucción que viene después de la de salto causa un fallo de página o una violación de protección, el procesador detiene la prebúsqueda hasta que esté seguro de que la instrucción debe ser captada.

Los estudios que analizan el comportamiento de un programa han mostrado que los saltos condicionales se producen más del 50 % de las veces [LILJ88], de forma que, si el coste de la prebúsqueda en cada camino es el mismo, precaptar siempre desde la dirección destino del salto debería proporcionar mayores prestaciones que precaptar siempre desde el camino secuencial. Sin embargo, en una máquina paginada, es más probable que cause un fallo de página la prebúsqueda en el destino del salto que la prebúsqueda de la siguiente instrucción secuencial, de manera que esta penalización en las prestaciones debería ser tenida en cuenta. Puede emplearse algún mecanismo que evite dicha penalización.

La última aproximación estática toma la decisión basándose en el código de operación de la instrucción de bifurcación. El procesador asume que el salto se producirá para ciertos códigos de operación de bifurcación y no para otros. [LILJ88] informa sobre tasas de acierto mayores del 75 por ciento con esta estrategia.

Las estrategias de bifurcación dinámicas intentan mejorar la exactitud de la predicción, registrando la historia de las instrucciones de bifurcación condicional en un programa. Por ejemplo, a cada instrucción de bifurcación pueden asociarse uno o más bits que reflejen su historia reciente. Estos bits son referenciados como un comutador saltar/no saltar, que dirige al procesador a tomar una determinada decisión la próxima vez que encuentre la instrucción. Típicamente, estos bits de historia no están asociados con la instrucción en memoria principal. Al contrario, se guardan temporalmente en un almacenamiento de alta velocidad. Una posibilidad es asociarlos con cualquier instrucción de bifurcación condicional que esté en cache. Cuando la instrucción en cache es reemplazada, su historia se pierde. Otra posibilidad es mantener una pequeña tabla para las instrucciones de bifurcación ejecutadas recientemente, con uno o más bits en cada elemento de la tabla. El procesador podría acceder a esa tabla asociativamente, como a una cache, o usando los bits de orden inferior de la dirección de la instrucción de bifurcación.

Con un único bit, todo lo que se puede registrar es si la última ejecución de una instrucción dio lugar a un salto o no. Una deficiencia de usar un solo bit se pone de manifiesto en el caso de instrucciones de bifurcación condicional, que casi siempre dan lugar a un salto, tales como la instrucción «loop». Con un único bit de historia, ocurrirá un error en la predicción dos veces en cada uso del bucle: una vez cuando se entra al bucle y otra cuando se sale de él.

Si se usan dos bits, se pueden emplear para registrar el resultado de las dos últimas veces que se ejecutó la instrucción asociada, o para registrar el estado de alguna otra forma. La Figura 11.16 muestra una aproximación típica (véase Problema 11.5 para otras posibilidades). El proceso de decisión puede representarse por medio de una máquina de estados finitos con cuatro estados. Si los dos últimos saltos de una instrucción dada han tomado el mismo camino, la predicción es tomar de nuevo el mismo camino. Si la predicción es errónea, per-

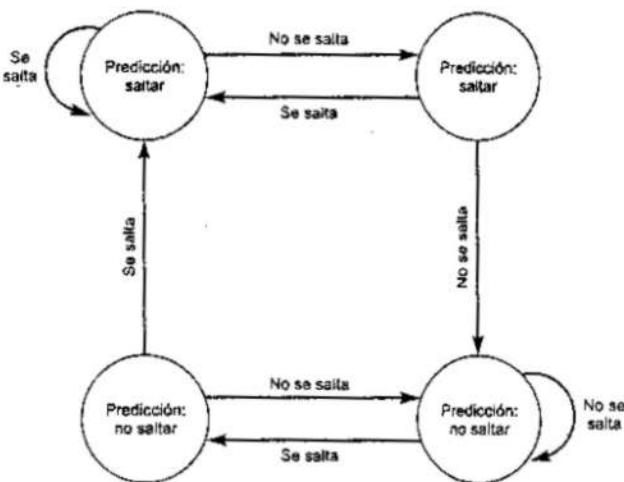


Figura 11.16. Diagrama de estados de la predicción de saltos.

manece igual la siguiente vez que se encuentre la instrucción. Sin embargo, si la predicción es errónea de nuevo, la siguiente predicción será seleccionar el camino opuesto. Por consiguiente, el algoritmo requiere dos predicciones erróneas consecutivas para cambiar la predicción. Si un salto toma un camino inusual una vez, tal como sucede con «loop», la predicción será errónea sólo una vez.

Un ejemplo de un sistema que usa la aproximación de un conmutador saltar/no saltar es el IBM 3090/400.

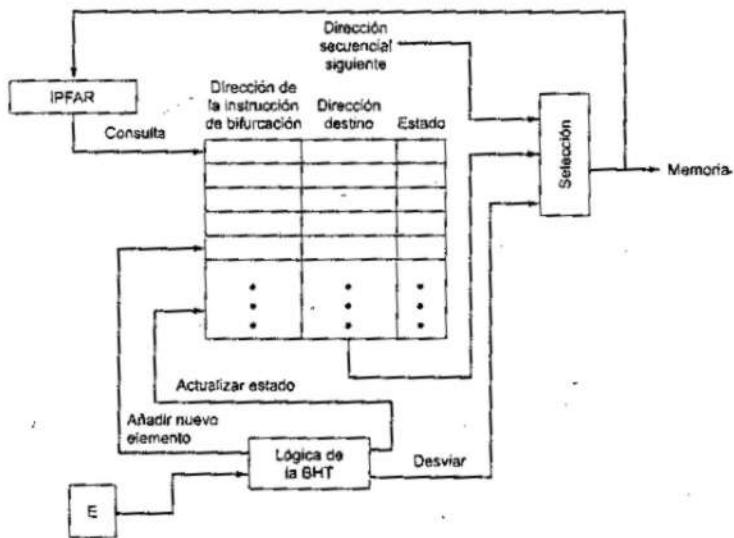
La utilización de bits de historia tiene, como se ha descrito, un inconveniente: si la decisión que se toma es efectuar el salto, la instrucción destino no puede captarse hasta que su dirección, que es un operando de la instrucción de bifurcación condicional, sea decodificada. Se podría lograr una mayor eficiencia si la instrucción captada pudiera iniciarse tan pronto como se tome la decisión de salto. Para ello se debe guardar más información en lo que se conoce como un «buffer de destino de saltos», o «tabla de historia de saltos» (branch history table, BHT).

La tabla de historia de saltos es una pequeña memoria cache asociada con la etapa de captación de instrucción del cauce. Cada elemento de la tabla consta de tres campos: la dirección de una instrucción de bifurcación, un determinado número de bits de historia que guardan el estado de uso de esa instrucción, e información sobre la instrucción destino. En la mayoría de los proyectos y realizaciones, este tercer campo contiene la dirección de la instrucción destino. Otra posibilidad es que el tercer campo contenga la instrucción destino. El compromiso es evidente: almacenar la dirección destino conduce a una tabla más pequeña, pero a un mayor tiempo de captación de instrucción que el almacenamiento de la instrucción destino [RECH98].

La Figura 11.17 contrasta este esquema frente a una estrategia del tipo «predecir que nunca se salta». En la primera estrategia (Figura 11.17a), la etapa de captación de instrucción siempre capta la siguiente dirección secuencial. Si se produce el salto, alguna lógica del procesador lo detecta, y ordena que la siguiente instrucción se capte de la dirección destinc-



(a) Estrategia de predecir que nunca se salta



(b) Estrategia de tabla de historia de saltos

Figura 11.17. Tratamiento de saltos.

(además de vaciar el cauce). La tabla de historia de saltos es tratada como una cache. Cada precaptación dispara una búsqueda en la tabla. Si no se encuentra ninguna coincidencia, se usa la dirección siguiente para la captación. Si hay coincidencia, se hace una predicción basada en el estado de la instrucción: a la lógica de selección se suministra, bien la dirección secuencial siguiente, bien la dirección destino del salto.

Cuando se ejecuta la instrucción de bifurcación, la etapa de ejecución comunica el resultado a la lógica de la tabla de historia de saltos. El estado de la instrucción se actualiza para

reflejar una predicción correcta o incorrecta. Si la predicción es incorrecta, la lógica de selección se desvía hacia la dirección correcta para la siguiente captación. Cuando se encuentra una instrucción de bifurcación condicional que no está en la tabla, se añade a la tabla, y uno de los elementos existentes se desecha, usando uno de los algoritmos de reemplazo de cache discutidos en el Capítulo 4.

Un ejemplo de realización de una tabla de historia de saltos se encuentra en el microprocesador Advanced Micro Device AMD29000.

### Salto retardado

Se pueden mejorar las prestaciones de un cauce reordenando automáticamente las instrucciones de un programa, de forma que las instrucciones de salto tengan lugar después de lo realmente deseado. Esta interesante aproximación se examina en el Capítulo 12.

## SEGMENTACIÓN DEL INTEL 80486

El 80486 tiene un cauce de cinco etapas:

- **Captación (Fetch):** Las instrucciones se captan de la cache o de la memoria externa, y se colocan en uno de los dos buffers de prebúsqueda de 16 bytes. El objetivo de la etapa de captación es llenar los buffers de prebúsqueda con nuevos datos, tan pronto como los viejos sean tratados por el decodificador de instrucciones. Dado que las instrucciones son de longitud variable (de 1 a 11 bytes sin contar prefijos), el estado del prebuscador relativo a las otras etapas del cauce varía de instrucción en instrucción. Como promedio, se captan alrededor de cinco instrucciones con cada carga de 16 bytes [CRAW90]. La etapa de captación opera independientemente de las otras etapas para mantener llenos los buffers de prebúsqueda.
- **Etapa de decodificación 1:** El código de operación y la información referente al modo de direccionamiento se decodifican en la etapa D1. La información necesaria, así como la información sobre la longitud de la instrucción, está contenida, a lo sumo, en los tres primeros bytes de la instrucción. De ahí que se pasen tres bytes desde los buffers de prebúsqueda a la etapa D1. El decodificador D1 puede entonces indicar a la etapa D2 que capture el resto de la instrucción (desplazamiento y dato inmediato) que no está involucrado en la decodificación de D1.
- **Etapa de decodificación 2:** La etapa D2 expande cada código de operación en señales de control para la ALU. También controla el cálculo de los modos de direccionamiento más complejos.
- **Ejecución (Execute, EX):** Esta etapa incluye operaciones de la ALU, acceso a cache y actualización de registros.
- **Escritura (Write Back, WB):** Esta etapa, cuando es necesaria, actualiza los registros e indicadores de estado modificados durante la etapa de ejecución precedente. Si la instrucción en curso actualiza la memoria, el valor calculado se envía al mismo tiempo a la cache y a los buffers de escritura de la interfaz del bus.

Con el uso de dos etapas de decodificación, el cauce puede mantener una productividad cercana a una instrucción por ciclo de reloj. Las instrucciones complejas y las bifurcaciones condicionales pueden reducir esta velocidad.

La Figura 11.18 presenta ejemplos de funcionamiento del cauce. La parte (a) muestra que no se introduce ningún retardo en el cauce cuando se necesita un acceso a memoria. No

|       |    |    |    |    |                |
|-------|----|----|----|----|----------------|
| Fetch | D1 | D2 | EX | WB | MOV Reg1, Mem1 |
| Fetch | D1 | D2 | EX | WB | MOV Reg1, Reg2 |
| Fetch | D1 | D2 | EX | WB | MOV Mem2, Reg1 |

(a) No hay retardo en el cauce debido a la carga de un dato

|       |    |    |    |    |                  |
|-------|----|----|----|----|------------------|
| Fetch | D1 | D2 | EX | WB | MOV Reg1, Mem1   |
| Fetch | D1 |    | D2 | EX | MOV Reg1, (Reg2) |

(b) Un retardo debido a una carga que utiliza un puntero

|       |    |    |       |    |               |
|-------|----|----|-------|----|---------------|
| Fetch | D1 | D2 | EX    | WB | CMP Reg1, imm |
| Fetch | D1 | D2 | EX    |    | Jcc Destino   |
|       |    |    | Fetch | D1 | Destino       |

(c) Temporización de una instrucción de bifurcación

Figura 11.18. Ejemplos del cauce de instrucciones del 80486.

obstante, como muestra la parte (b), puede haber un retardo debido a valores usados para calcular direcciones de memoria. Es decir, si un valor se carga desde memoria a un registro, y ese registro se usa como registro base en la siguiente instrucción, el procesador se parará durante un ciclo. En este ejemplo, el procesador accede a la cache en la etapa EX de la primera instrucción y almacena en el registro el valor recuperado durante la etapa WB. Sin embargo, la siguiente instrucción necesita este registro en su etapa D2. Cuando la etapa D2 se alinea con la etapa WB de la instrucción previa, hay caminos que desvian las señales, y permiten que la etapa D2 tenga acceso a los mismos datos que está usando la etapa WB para escritura, ahorrando una etapa del cauce.

La Figura 11.18c ilustra la temporización de una instrucción de bifurcación, suponiendo que se produce el salto. La instrucción de comparación actualiza los códigos de condición en la etapa WB, y unos caminos de atajo hacen que la etapa EX de la instrucción de salto disponga de ellos en el mismo momento. En paralelo, el procesador ejecuta un ciclo especulativo de captación del destino del salto durante la etapa EX de la instrucción de salto. Si el procesador determina que la condición de salto es falsa, desecha esta precaptación y continua la ejecución con la siguiente instrucción secuencial (ya captaida y decodificada).

## 11.5. EL PROCESADOR PENTIUM

En la Figura 4.23 se representó una visión de conjunto de la organización del procesador Pentium II. En esta sección examinaremos algunos detalles.

### ORGANIZACIÓN DE LOS REGISTROS

La organización de los registros incluye los siguientes tipos de registros (Tabla 11.1):

- **Generales:** Hay ocho registros de uso general de 32 bits (véase Figura 11.3c). Pueden ser usados por cualquier tipo de instrucción del Pentium II; también pueden contener operandos para cálculos de direcciones. Además, algunos de estos registros pueden servir para usos especiales. Por ejemplo, las instrucciones de cadenas usan los contenidos de los registros ECX, ESI y EDI como operandos, sin tener que referenciar explícitamente estos registros en la instrucción. Por consiguiente, varias instrucciones pueden codificarse de modo más compacto.
- **De segmento:** Los seis registros de segmento de 16 bits contienen selectores de segmento, que indexan tablas de segmentos, como vimos en el Capítulo 7. El registro de segmento de código (Code Segment, CS) referencia el segmento que contiene la instrucción que se está ejecutando. El registro de segmento de pila (Stack Segment, SS) referencia el segmento que contiene una pila visible para el usuario. Los demás registros de segmento (DS, ES, FS y GS) permiten al usuario referenciar, al mismo tiempo, hasta cuatro segmentos de datos distintos.
- **Indicadores:** El registro EFLAGS contiene los códigos de condición y diversos bits de modo.
- **Puntero de instrucción:** Contiene la dirección de la instrucción en curso.

Tabla 11.1. Registros del procesador Pentium II

| (a) Unidad de enteros  |        |                 |                                     |
|------------------------|--------|-----------------|-------------------------------------|
| Tipo                   | Número | Longitud (bits) | Propósito                           |
| Generales              | 8      | 32              | Registros de usuario de uso general |
| De segmento            | 6      | 16              | Contienen selectores de segmento    |
| Indicadores            | 1      | 32              | Bits de estado y control            |
| Puntero de instrucción | 1      | 32              | Puntero de instrucción              |

| (b) Unidad de coma flotante |        |                 |                                                        |
|-----------------------------|--------|-----------------|--------------------------------------------------------|
| Tipo                        | Número | Longitud (bits) | Propósito                                              |
| Numérico                    | 8      | 80              | Contienen números en coma flotante                     |
| Control                     | 1      | 16              | Bits de control                                        |
| Estado                      | 1      | 16              | Bits de estado                                         |
| Palabra de etiquetas        | 1      | 16              | Especifica los contenidos de los registros numéricos   |
| Puntero de instrucción      | 1      | 48              | Apunta a la instrucción interrumpida por una excepción |
| Puntero de dato             | 1      | 48              | Apunta al operando interrumpido por una excepción      |

Hay también registros dedicados específicamente a la unidad de coma flotante:

- **Numéricos:** Cada registro contiene un número en coma flotante de 80 bits de precisión ampliada. Hay ocho registros que funcionan como una pila, con operaciones «push» y «pop» disponibles en el repertorio de instrucciones.
- **De control:** El registro de control de 16 bits contiene bits que controlan el funcionamiento de la unidad de coma flotante (incluyendo el control del tipo de redondeo), precisión simple, doble o ampliada, y bits para habilitar e inhabilitar diversas condiciones de excepción.
- **De estado:** El registro de estado de 16 bits contiene bits que reflejan el estado presente de la unidad de coma flotante (incluyendo un puntero de 3 bits al tope de la pila), códigos de condición que informan sobre el resultado de la última operación, e indicadores de excepción.
- **Palabras de etiquetas:** Este registro de 16 bits contiene una etiqueta de 2 bits para cada registro numérico de coma flotante, que indica la naturaleza de los contenidos del registro correspondiente. Los cuatro valores posibles son válido, cero, especial (NaN, infinito, denormalizado), y vacío. Estas etiquetas permiten a los programas comprobar los contenidos de un registro numérico sin tener que realizar una compleja decodificación de los datos que hay en el registro. Por ejemplo, cuando se hace un cambio de contexto, el procesador no tiene que salvar ninguno de los registros de coma flotante que están vacíos.

El uso de la mayor parte de los registros anteriores se comprende fácilmente. Detallaremos brevemente algunos de los registros.

### Registro EFLAGS

El registro EFLAGS (Figura 11.19) indica el estado del procesador y ayuda a controlar su funcionamiento. Incluye los seis códigos de condición definidos en la Tabla 9.8 (acarreo, paridad, acarreo auxiliar, cero, signo y desbordamiento), que informan sobre el resultado de una operación entera. Además, en el registro hay bits que podemos denominar de control:

- **Indicador de trampa (Trap flag, TF):** Cuando está a uno provoca una interrupción tras la ejecución de cada instrucción. Se usa para depuración.

|  | 21     | 16     | 15     |        | 0        |
|--|--------|--------|--------|--------|----------|
|  | V<br>F | V<br>I | A<br>C | V<br>M | R<br>F   |
|  | D<br>P | F<br>F |        | N<br>T | IO<br>PL |

|                                     |                                                  |
|-------------------------------------|--------------------------------------------------|
| = Indicador de identificación       | DF = Indicador de dirección                      |
| = Interrupción virtual pendiente    | IF = Indicador de habilitación de interrupciones |
| = Indicador de interrupción virtual | TF = Indicador de trampa                         |
| = Verificación de alineación        | SF = Indicador de signo                          |
| = Modo virtual 8086                 | ZF = Indicador de cero                           |
| = Indicador de reanudación          | AF = Indicador de acarreo auxiliar               |
| = Indicador de tarea anidada        | PF = Indicador de paridad                        |
| = Nivel de privilegio de E/S        | CF = Indicador de acarreo                        |
| = Indicador de desbordamiento       |                                                  |

Figura 11.19. Registro EFLAGS del Pentium II.

- **Indicador de habilitación de interrupciones (Interrupt enable flag, IF):** Cuando está a uno, el procesador aceptará las interrupciones externas.
- **Indicador de dirección (Direction flag, DF):** Determina si las instrucciones de procesamiento de cadenas incrementan o decrementan los semiregistros de 16 bits SI y DI (para operaciones de 16 bits) o los registros de 32 bits ESI y EDI (para operaciones de 32 bits).
- **Nivel de privilegio de E/S (I/O privilege level, IOPL):** Hace que el procesador genere una excepción en los accesos no permitidos a dispositivos de E/S cuando funciona en modo protegido.
- **Indicador de reanudación (Resume flag, RF):** Permite al programador inhabilitar las excepciones de depuración, de manera que la instrucción pueda empezar de nuevo después de una excepción de depuración sin que cause inmediatamente otra excepción de depuración.
- **Verificación de alineación (Alignment check, AC):** Controla si una palabra o doble palabra puede referenciarse en una dirección que no sea múltiplo de dos o de cuatro, respectivamente.
- **Indicador de identificación (Identification flag, ID):** Si este bit se puede poner a uno y borrar, indica que el procesador soporta la instrucción CPUID. Esta instrucción proporciona información sobre el fabricante, la familia y el modelo.

Además, hay cuatro bits relacionados con el modo de funcionamiento. El indicador de tarea anidada (nested task, NT) indica que la tarea actual está anidada con otra tarea cuando el microprocesador funciona en modo protegido. El bit de modo virtual (virtual mode, VM) permite al programador habilitar o inhabilitar el modo virtual 8086, que determina si el procesador funciona o no como una máquina 8086. Los indicadores de interrupción virtual (virtual interrupt flag, VIF) e interrupción virtual pendiente (virtual interrupt pending, VIP) se usan en entornos multitarea.

### Registros de control

El Pentium II emplea cuatro registros de control de 32 bits (el registro CR1 no se usa) para controlar varios aspectos del funcionamiento del procesador (Figura 11.20). El registro CR0 contiene indicadores de control del sistema, que controlan modos o indican estados que se aplican generalmente al procesador, en lugar de a la ejecución de una tarea individual. Los indicadores son los siguientes:

- **Habilitación de protección (Protection enable, PE):** Habilita/inhabilita el modo de funcionamiento protegido.
- **Coprocésador presente (Monitor coprocessor, MP):** Sólo tiene interés cuando se ejecutan programas de máquinas anteriores al Pentium II; indica la presencia de un coprocesador aritmético.
- **Emulación (Emulation, EM):** Puesto a uno cuando el procesador no tiene una unidad de coma flotante, causa una interrupción cuando se intentan ejecutar instrucciones de coma flotante.
- **Tarea comutada (Task switched, TS):** Indica que el procesador tiene tareas comutadas.
- **Tipo de coprocésador (Extension type, ET):** No se usa en el Pentium II; se usaba para indicar el soporte de instrucciones del coprocésador en máquinas anteriores.

|     |        |        |        |  |  |  |  |  |        |        |  |  |  |  |  |  |        |        |        |        |        |   |  |  |
|-----|--------|--------|--------|--|--|--|--|--|--------|--------|--|--|--|--|--|--|--------|--------|--------|--------|--------|---|--|--|
|     |        |        |        |  |  |  |  |  |        |        |  |  |  |  |  |  |        |        |        |        |        |   |  |  |
| CR4 |        |        |        |  |  |  |  |  |        |        |  |  |  |  |  |  |        |        |        |        |        |   |  |  |
| CR3 |        |        |        |  |  |  |  |  |        |        |  |  |  |  |  |  |        |        |        |        |        |   |  |  |
| CR2 |        |        |        |  |  |  |  |  |        |        |  |  |  |  |  |  |        |        |        |        |        |   |  |  |
| CR1 |        |        |        |  |  |  |  |  |        |        |  |  |  |  |  |  |        |        |        |        |        |   |  |  |
| CR0 | P<br>G | C<br>D | N<br>W |  |  |  |  |  | A<br>M | W<br>P |  |  |  |  |  |  | N<br>E | E<br>T | T<br>S | E<br>M | E<br>P |   |  |  |
|     | 31     | 30     | 29     |  |  |  |  |  | 18     | 16     |  |  |  |  |  |  | 5      | 4      | 3      | 2      | 1      | 0 |  |  |

PCE = Habilitación del contador de prestaciones  
 PGE = Habilitación de páginas globales  
 MCE = Habilitación de verificación de la máquina  
 PAE = Ampliación de la dirección física  
 PSE = Ampliaciones del tamaño de página  
 DE = Ampliaciones de depuración  
 TSD = Inhabilitación de marca de tiempo  
 PVI = Interrupciones virtuales en el modo protegido  
 VME = Ampliación del modo virtual 8086  
 PCD = Inhabilitación de cache en el nivel de página  
 PWT = Escritura transparente en el nivel de página

PG = Paginación  
 CD = Inhabilitación de cache  
 NW = No escritura inmediata  
 AM = Máscara de alineación  
 WP = Protección de escritura  
 NE = Error numérico  
 ET = Tipo de coprocesador  
 TS = Tarea comunitaria  
 EM = Emulación  
 MP = Coprocesador presente  
 PE = Habilitación de protección

Figura 11.20. Registros de control del Pentium II.

- **Error numérico (Numeric error, NE):** Habilita el mecanismo estándar para informar de errores de coma flotante en las líneas de bus externas.
- **Protección de escritura (Write protect, WP):** Cuando este bit está a cero, un proceso supervisor puede escribir en páginas de sólo lectura de nivel de usuario. Esta característica es útil para dar soporte a la creación de procesos en algunos sistemas operativos.
- **Máscara de alineación (Alignment mask, AM):** Habilita/inhabilita la verificación de alineación.
- **No escritura inmediata (Not write through, NW):** Selecciona el modo de funcionamiento de la cache de datos. Cuando este bit está a uno, se impide a la cache de datos realizar operaciones de escritura inmediata.
- **Inhabilitación de cache (Cache disable, CD):** Habilita/inhabilita el mecanismo de llenado de la cache interna.
- **Paginación (Paging, PG):** Habilita/inhabilita la paginación.

Cuando la paginación está habilitada, los registros CR2 y CR3 son válidos. El registro CR2 contiene la dirección lineal de 32 bits de la última página accedida antes de una interrupción de fallo de página. Los 20 bits más a la izquierda de CR3 contienen los 20 bits más significativos de la dirección base del directorio de páginas; el resto de la dirección contiene ceros. Dos bits de CR3 se usan para activar patillas de interconexión que controlan el funcionamiento de una cache externa. El bit de inhabilitación de cache en el nivel de página («page-level cache disable», PCD) habilita o inhabilita la cache externa, y el bit de escritura transpa-

rente en el nivel de página («page-level writes transparent», PWT) controla la escritura inmediata en la cache externa.

En CR4 se definen nueve bits de control adicionales:

- **Ampliación del modo virtual 8086 (Virtual-8086 mode extension, VME):** Habilita el soporte del indicador de interrupción virtual en el modo virtual 8086.
- **Interrupciones virtuales en el modo protegido (Protected-mode virtual interrupts, PVI):** Habilita el soporte del indicador de interrupción virtual en el modo protegido.
- **Inhabilitación de marca de tiempo (Time stamp disable, TSD):** Inhabilita la instrucción de lectura del contador de marca de tiempo («read from time stamp counter», RDTSC), que se usa con objetivos de depuración.
- **Ampliaciones de depuración (Debugging extensions, DE):** Habilita puntos de interrupción de E/S; esto permite al procesador interrumpir lecturas y escrituras de E/S.
- **Ampliaciones del tamaño de página (Page size extensions, PSE):** Habilita el uso de páginas de 4 Mbytes cuando está a uno en el Pentium, o de 2 Mbytes cuando está a uno en el Pentium Pro o Pentium II.
- **Ampliación de la dirección física (Physical address extension, PAE):** Habilita las líneas de dirección, A35 a A32, cuando un nuevo modo de direccionamiento especial, controlado por PSE, esté habilitado en el Pentium Pro o Pentium II.
- **Habilitación de verificación de la máquina (Machine check enable, MCE):** Habilita la interrupción de verificación de la máquina, que tiene lugar cuando ocurre un error de paridad de datos durante un ciclo del bus de lectura o cuando un ciclo del bus no termina con éxito.
- **Habilitación de páginas globales (Page global enable, PGE):** Habilita el uso de páginas globales. Cuando PGE = 1 y se produce una conmutación de tarea, todas las entradas de la TLB se vacían, a excepción de aquellas señaladas como globales.
- **Habilitación del contador de prestaciones (Performance counter enable, PCE):** Habilita la ejecución de la instrucción RDPMC («read performance counter» o lectura de contador de prestaciones) en cualquier nivel de privilegio. Se usan dos contadores de prestaciones, para medir la duración de un tipo de evento específico y el número de ocurrencias de un tipo de evento específico.

## Registros MMX

En la Sección 9.4 vimos que la capacidad MMX del Pentium II utiliza varios tipos de datos de 64 bits. Las instrucciones MMX utilizan campos de direccionamiento de registros de 3 bits, de modo que son posibles ocho registros MMX. En realidad, el procesador no incluye registros MMX específicos. En vez de eso, usa una técnica de renombramiento (Figura 11.21). Los registros de coma flotante existentes se usan para almacenar los valores MMX. Concretamente, se emplean los 64 bits más bajos (mantisa) de cada registro de coma flotante para formar los ocho registros MMX. De este modo, la arquitectura Pentium II existente se amplía con facilidad para admitir la capacidad MMX. El uso MMX de estos registros tiene las siguientes características fundamentales:

- Recordemos que los registros de coma flotante se tratan como una pila en las operaciones de coma flotante. En las operaciones MMX, los mismo registros se acceden directamente.

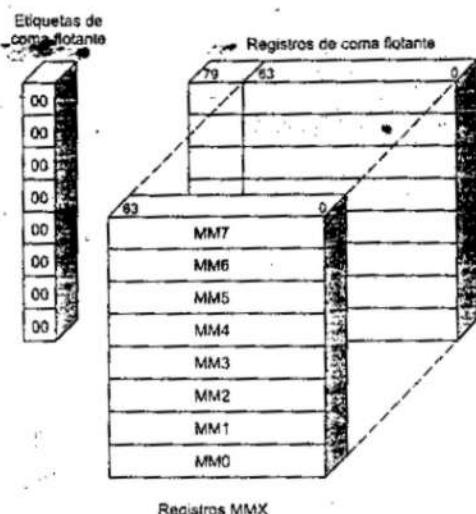


Figura 11.21. Correspondencia de registros MMX con registros de coma flotante.

- La primera vez que una instrucción MMX se ejecuta después de cualquier operación de coma flotante, la palabra de etiquetas FP (Floating Point: «coma flotante») se marca como válida. Esto refleja el cambio de funcionamiento como pila a direccionamiento directo a registro.
- La instrucción EMMS (Empty MMX State: «abandonar estado MMX») fija los bits de la palabra de etiquetas FP de modo que indique que todos los registros se encuentran vacíos. Es importante que el programador inserte esta instrucción al final de un bloque de código MMX para que las operaciones de coma flotante posteriores funcionen debidamente.
- Cuando se escribe un valor en un registro MMX, los bits [79:64] del correspondiente registro FP (bits de signo y exponente) se ponen todos a uno. Esto fija el valor del registro FP a NaN (Not a Number: «indeterminación») o infinito cuando se ve como un valor de coma flotante. Ello asegura que un dato MMX no parezca un valor de coma flotante válido.

## PROCESAMIENTO DE INTERRUPCIONES

El procesamiento de interrupciones dentro del procesador es un servicio que se proporciona para apoyar al sistema operativo. Permite que un programa de aplicación sea suspendido para que una gran variedad de causas de interrupción puedan atenderse, y reanudado más tarde.

### Interrupciones y excepciones

Hay dos clases de eventos que provocan que el Pentium II suspenda la ejecución del flujo de instrucciones en curso y responda al evento: las interrupciones y las excepciones. En los dos

casos, el procesador guarda el contexto del proceso actual y pasa a una rutina predefinida para atender la condición. Una **interrupción** se genera por una señal del hardware, y puede ocurrir en momentos aleatorios durante la ejecución de un programa. Una **excepción** se genera desde el software, y es provocada por la ejecución de una instrucción. Hay dos fuentes de interrupciones, y dos fuentes de excepciones:

1. Interrupciones
  - **Interrupciones enmascarables:** Las recibe el procesador por la patilla INTR. El procesador no reconoce una interrupción enmascarable, a no ser que el indicador de habilitación de interrupciones (IF) esté a uno.
  - **Interrupciones no enmascarables:** Las recibe el procesador por la patilla NMI. No se puede evitar el reconocimiento de tales interrupciones.
2. Excepciones
  - **Excepciones detectadas por el procesador:** Se producen cuando el procesador encuentra un error mientras intenta ejecutar una instrucción.
  - **Excepciones programadas:** Hay instrucciones que generan una excepción (INTO, INT 3, INT, y BOUND).

### Tabla de vectores de interrupción

El procesamiento de las interrupciones en el Pentium II usa la tabla de vectores de interrupción. Cada tipo de interrupción tiene asignado un número, que se usa para indexar en la tabla de vectores de interrupción. La tabla contiene 256 vectores de interrupción de 32 bits, cada uno de los cuales es la dirección (segmento y desplazamiento) de la rutina de servicio de interrupción del número de interrupción correspondiente.

La Tabla 11.2 muestra la asignación de números en la tabla de vectores de interrupción; las entradas sombreadas representan interrupciones, mientras que las no sombreadas son excepciones. La interrupción hardware NMI corresponde al vector 2. Las interrupciones hardware INTR tienen asignados números en el rango 32 a 255; cuando se genera una interrupción INTR, debe venir acompañada en el bus con el número de vector de interrupción para esa interrupción. Los números de vectores restantes se usan para excepciones.

Si hay pendientes más de una excepción o interrupción, el procesador las atiende en un orden previsible. La posición de los números de vector dentro de la tabla no refleja ninguna prioridad. En lugar de eso, la prioridad entre las excepciones e interrupciones se organiza en cinco clases. En orden decreciente de prioridad, son las siguientes:

- Clase 1: Intercepción en la instrucción previa (vector número 1).
- Clase 2: Interrupciones externas (2, 32-255).
- Clase 3: Fallos captando la siguiente instrucción (3, 14).
- Clase 4: Fallos decodificando la siguiente instrucción (6, 7).
- Clase 5: Fallos ejecutando una instrucción (0, 4, 5, 8, 10-14, 16, 17).

### Gestión de interrupciones

Igual que las transferencias de ejecución que usan la instrucción CALL, una transferencia a una rutina de gestión de interrupción usa la pila del sistema para almacenar el estado del

Tabla 11.2. Tabla de vectores de excepción e interrupción del Pentium II

| Número de vector | Descripción                                                                                                                                                                                 |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0                | Error al dividir; desbordamiento de división o división por cero.                                                                                                                           |
| 1                | Excepción de depuración; incluye varios fallos e interceptaciones relacionados con la depuración.                                                                                           |
| 2                | Interrupción de la patilla NMI; señal en la patilla NMI.                                                                                                                                    |
| 3                | Punto de parada; causado por la instrucción INT 3; una instrucción de un byte usado para depuración.                                                                                        |
| 4                | Desbordamiento detectado por INTO; ocurre cuando el procesador ejecuta INT3 con el indicador OF a uno.                                                                                      |
| 5                | Rango de BOUND excedido; la instrucción BOUND compara un registro con uno límites almacenados en memoria, y genera una interrupción si el contenido del registro está fuera de los límites. |
| 6                | Código de operación no definido.                                                                                                                                                            |
| 7                | Dispositivo no disponible; un intento de usar las instrucciones ESC o WAIT da error debido a la falta de un dispositivo externo.                                                            |
| 8                | Doble fallo: dos interrupciones ocurren durante la misma instrucción, y no pueden ser atendidas en serie.                                                                                   |
| 9                | Reservado.                                                                                                                                                                                  |
| 10               | Segmento de estado de tarea no válido; el segmento que describe la tarea solicitada no está inicializado o no es válido.                                                                    |
| 11               | Segmento no presente; el segmento solicitado no está presente.                                                                                                                              |
| 12               | Fallo en la pila; se ha excedido el límite del segmento de pila o el segmento de pila no está presente.                                                                                     |
| 13               | Protección general; violación de protección que no causa otra excepción (por ejemplo, ascribir en un segmento de sólo lectura).                                                             |
| 14               | Fallo de página.                                                                                                                                                                            |
| 15               | Reservado.                                                                                                                                                                                  |
| 16               | Error de coma flotante; generado por una instrucción de aritmética en coma flotante.                                                                                                        |
| 17               | Verificación de alineación; acceso a una palabra almacenada en una dirección de byte impar, o a una doble palabra almacenada en una dirección que no sea múltiplo de 4.                     |
| 18               | Verificación de la máquina; específica para cada modelo.                                                                                                                                    |
| 19-31            | Reservado.                                                                                                                                                                                  |
| 32-255           | Vectores de interrupción del usuario; suministrados cuando se activa la señal INTR.                                                                                                         |

No sombreadas: excepciones

Sombreadas: interrupciones

procesador. Cuando una interrupción tiene lugar y es reconocida por el procesador, tiene lugar la siguiente secuencia de eventos:

- Si la transferencia supone un cambio del nivel de privilegio, los contenidos actuales del registro de segmento de pila y del registro puntero de pila ampliado (ESP) se introducen en la pila.
- El valor actual del registro EFLAGS se introduce en la pila.
- Los indicadores de interrupciones (IF) y de trampa (TF) se ponen a cero. Ello inhabilita las interrupciones INTR y la interceptación o modo paso a paso.
- Los contenidos actuales del puntero de segmento de código (CS) y del puntero de instrucción (IP o EIP) se introducen en la pila.
- Si la interrupción viene acompañada por un código de error, éste se introduce en la pila.

6. Los contenidos del vector de interrupción se captan y se cargan en los registros CS e IP o EIP. La ejecución continúa por la rutina de servicio de interrupción.

Para retornar de una interrupción, la rutina de servicio de interrupción ejecuta una instrucción IRET. Esto provoca que todos los valores almacenados en la pila sean restablecidos; la ejecución se reanuda a partir del punto de interrupción.

### 11.6. El procesador PowerPC

En la Figura 4.25 se representó una visión de conjunto de la organización del procesador PowerPC. En esta sección examinamos algunos de los detalles de la implementación de 64 bits.

## ORGANIZACIÓN DE LOS REGISTROS

La Figura 11.22 representa los registros visibles para el usuario en el PowerPC. La unidad de coma flia incluye los siguientes:

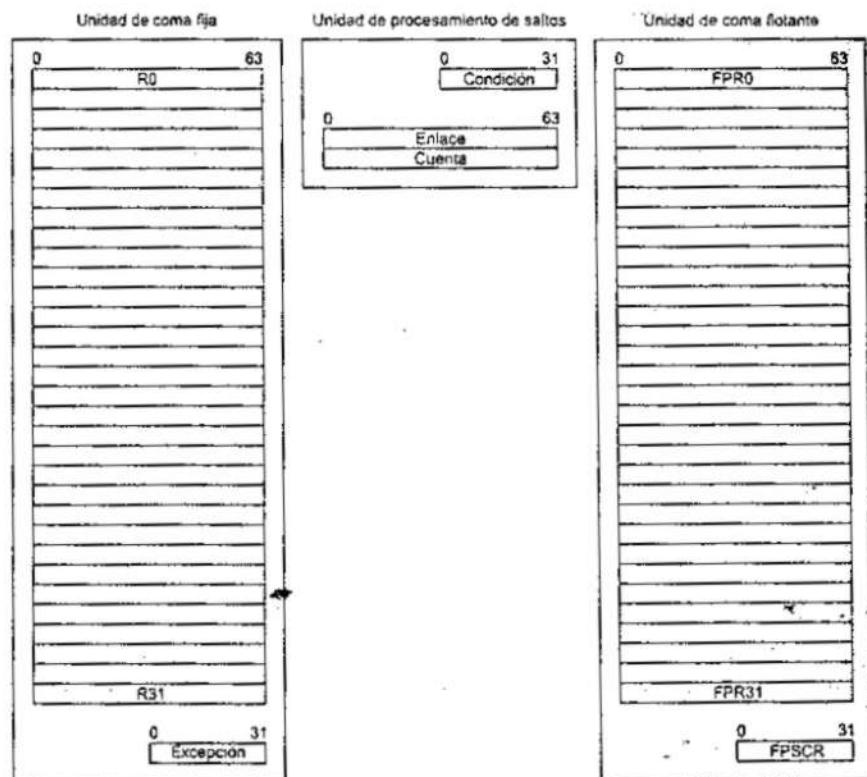


Figura 11-22. Registros del PowerPC visibles por el usuario.

|   |   |   |   |  |  |  |  |  |  |    |                 |
|---|---|---|---|--|--|--|--|--|--|----|-----------------|
| 0 | S | O | C |  |  |  |  |  |  | 25 | 31              |
| D | O | V | A |  |  |  |  |  |  |    | Cuenta de bytes |

- SO = Sumario de desbordamientos (Summary Overflow): puesto a 1 para indicar que hubo desbordamiento durante la ejecución de una instrucción; permanece a 1 hasta que se borre por software.
- OV = Desbordamiento (Overflow): puesto a 1 para indicar que hubo desbordamiento durante la ejecución de una instrucción; puesto a 0 por la siguiente instrucción si no hay desbordamiento.
- CA = Acarreo (Carry): puesto a 1 para indicar acarreo en el bit 0 durante la ejecución de una instrucción.
- Cuenta de bytes = Especifica el número de bytes a transferir por una instrucción indexada de carga/almacenamiento de cadena.

(a) Registro de excepción de coma fija (XER)

|     |       |       |         |         |         |         |         |    |
|-----|-------|-------|---------|---------|---------|---------|---------|----|
| 0   | 3 / 4 | 7 / 8 | 11 / 12 | 15 / 16 | 19 / 20 | 23 / 24 | 27 / 28 | 31 |
| CR0 | CR1   | CR2   | CR3     | CR4     | CR5     | CR6     | CR7     |    |

Instrucciones con enteros  
Instrucciones de coma flotante

Instrucciones de comparación

(b) Registro de condición

Figura 11.23. Formatos de registros del PowerPC.

- **Generales:** Hay treinta y dos registros de 64 bits de uso general. Se pueden utilizar para cargar, almacenar y manipular operandos de datos, y también se pueden utilizar para direccionamiento indirecto a través de registro. El registro 0 se trata de una forma algo diferente. Para operaciones de carga y almacenamiento, y algunas de las instrucciones de suma, el registro 0 se trata como si tuviera un valor constante cero sin hacer caso de su contenido real.
- **Registro de excepción (Exception register, XER):** Incluye tres bits, que informan sobre excepciones en operaciones aritméticas con enteros. También incluye un campo contador de bytes, que se usa como operando en algunas instrucciones con cadenas (Figura 11.23a).

La unidad de coma flotante contiene otros registros visibles para el usuario:

- **Generales:** Hay treinta y dos registros de uso general de 64 bits, usados para todas las operaciones de coma flotante.
- **Registro de estado y control de coma flotante (Floating-point status and control register, FPSCR):** Este registro de 32 bits contiene bits que controlan el funcionamiento de la unidad de coma flotante, y bits que guardan el estado resultante de operaciones de coma flotante (Tabla 11.3).

La unidad de procesamiento de saltos contiene estos registros visibles para el usuario:

- **Registro de condición:** Consta de ocho campos de código de condición de 4 bits (Figura 11.22b).

**Tabla 11.3.** Registro de estado y control de coma flotante del PowerPC .

| Bit   | Definición                                                                                                                                                                                          |
|-------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0     | Sumario de excepciones. A uno, si ocurre alguna excepción; permanece a uno hasta que se reinicialice por software.                                                                                  |
| 1     | Sumario de excepciones habilitadas. A uno, si ha ocurrido alguna excepción habilitada.                                                                                                              |
| 2     | Sumario de excepciones de operación no válida. A uno, si ha ocurrido una excepción de operación no válida.                                                                                          |
| 3     | Excepción de desbordamiento (overflow). La magnitud del resultado excede la que se puede representar.                                                                                               |
| 4     | Excepción de desbordamiento hacia cero (underflow). El resultado es demasiado pequeño para ser normalizado.                                                                                         |
| 5     | Excepción de división por cero. El divisor es cero y el dividendo es un número finito distinto de cero.                                                                                             |
| 6     | Excepción de inexactitud. El resultado redondeado es distinto del resultado intermedio o se produce desbordamiento con la excepción de desbordamiento inhabilitada.                                 |
| 7:12  | Excepción de operación no válida. 7: indica NaN; 8: ( $\infty - \infty$ ); 9: ( $\infty \div \infty$ ); 10: (0 - 0); 11: ( $\infty \times 0$ ); 12: comparación con NaN.                            |
| 13    | Fracción redondeada. El redondeo del resultado incrementó la fracción.                                                                                                                              |
| 14    | Fracción inexacta. El resultado redondeado cambia la fracción, u ocurre un desbordamiento con la excepción de desbordamiento inhabilitada.                                                          |
| 15:19 | Indicadores de resultado. Un código de cinco bits que especifica: menor que, mayor que, igual, fuera de categoría, NaN silencioso, $\pm \infty$ , $\pm$ normalizado, $\pm$ denormalizado, $\pm 0$ . |
| 20    | Reservado.                                                                                                                                                                                          |
| 21:23 | Excepción de operación no válida. 21: petición software; 22: raíz cuadrada de un número negativo; 23: conversión entera que involucra un número grande, un infinito, o un NaN.                      |
| 24    | Habilitación de excepción de operación no válida.                                                                                                                                                   |
| 25    | Habilitación de excepción de desbordamiento (overflow).                                                                                                                                             |
| 26    | Habilitación de excepción de desbordamiento hacia cero (underflow).                                                                                                                                 |
| 27    | Habilitación de excepción de división por cero.                                                                                                                                                     |
| 28    | Habilitación de excepción de inexactitud.                                                                                                                                                           |
| 29    | Modo no IEEE.                                                                                                                                                                                       |
| 30:31 | Control de redondeo. Un código de dos bits especifica al más cercano, a 0, a $+\infty$ , o a $-\infty$ .                                                                                            |

No sombreados: bits de estado

Sombreados: bits de control

- **Registro de enlace:** El registro de enlace puede usarse en una instrucción de bifurcación condicional para direccionamiento indirecto de la dirección destino. También se usa para el funcionamiento de «call/return». Si el bit LK en una instrucción de bifurcación condicional está a uno, la dirección siguiente a la instrucción de bifurcación se coloca en el registro de enlace, y se puede usar para un retorno posterior.
- **Cuenta:** El registro de cuenta puede usarse para controlar iteraciones de bucles, como se explicó en el Capítulo 9; este registro se decremente cada vez que es examinado por una instrucción de bifurcación condicional. Otro uso de este registro es para direccionamiento indirecto de la dirección destino en una instrucción de bifurcación.

Los campos del registro de condición tienen varios usos. Los primeros 4 bits (CR0) se actualizan en todas las instrucciones aritméticas con enteros para las cuales el bit Rc esté a uno. Como muestra la Tabla 11.4, el campo indica si el resultado de la operación es positivo, negativo o cero. El cuarto bit es una copia del bit sumario de desbordamientos del XER. El

Tabla 11.4. Interpretación de los bits del registro de condición

| Posición de bit | CR0<br>(instrucción entera con $Rc = 1$ ) | CR1<br>(instrucción de coma flotante con $Rc = 1$ ) | CRi<br>(instrucción de comparación en coma fija) | CRi<br>(instrucción de comparación en coma flotante) |
|-----------------|-------------------------------------------|-----------------------------------------------------|--------------------------------------------------|------------------------------------------------------|
| i               | resultado < 0                             | Sumario de excepciones                              | $op1 < op2$                                      | $op1 < op2$                                          |
| i + 1           | resultado > 0                             | Sumario de excepciones habilitadas                  | $op1 > op2$                                      | $op1 > op2$                                          |
| i + 2           | resultado = 0                             | Sumario de excepciones de operación no válida       | $op1 = op2$                                      | $op1 = op2$                                          |
| i + 3           | Sumario de desbordamientos                | Excepción de desbordamiento                         | Sumario de desbordamientos                       | Fuera de categoría (un operando es un NaN)           |

siguiente campo (CR1) se actualiza en todas las instrucciones aritméticas de coma flotante para las cuales el bit  $Rc$  esté a uno. En este caso, los 4 bits son iguales a los 4 primeros bits del FPSCR (Tabla 11.3). Por último, los ocho campos de condición (CR0 a CR7) se pueden usar con una instrucción de comparación: en cada caso, la identidad del campo se especifica en la propia instrucción. Para las dos instrucciones de comparación en coma fija y en coma flotante, los primeros 3 bits del campo de condición designado guardan si el primer operando es menor que, mayor que o igual que el segundo operando. El cuarto bit es el bit sumario de desbordamientos para una comparación de coma fija, y un indicador de fuera de categoría para la comparación en coma flotante.

## PROCESAMIENTO DE INTERRUPCIONES

Como en cualquier procesador, el PowerPC incluye un servicio que permite al procesador interrumpir el programa que se ejecuta para ocuparse de una condición de excepción.

### Tipos de interrupciones

Las interrupciones en un PowerPC se clasifican en: las que son causadas por alguna condición o evento del sistema, y las causadas por la ejecución de una instrucción. La Tabla 11.5 lista las interrupciones que reconoce el PowerPC.

La mayoría de las interrupciones listadas en la tabla se comprenden fácilmente. Unas pocas justifican el ser comentadas. La interrupción de reinicio del sistema ocurre en el encendido y cuando se pulsa el botón «reset» en la unidad del sistema, y hace que se reinicie el sistema. La interrupción de chequeo de la máquina se ocupa de ciertas anomalías, tales como un error de paridad de caché o una referencia a una posición de memoria inexistente, y puede hacer que el sistema entre en lo que se conoce como «estado de parada de chequeo»; este estado suspende la ejecución del procesador, y congela el contenido de los registros hasta un reinicio. La ayuda a coma flotante permite al procesador invocar rutinas software para completar operaciones que no puede manejar directamente la unidad de coma flotante, como las que implican números denormalizados o códigos de operación en coma flotante no implementados.

Tabla 11.5. Tabla de interrupciones del PowerPC

| Punto de entrada    | Tipo de interrupción                       | Descripción                                                                                                                                                                                     |
|---------------------|--------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 00000h              | Reservada                                  |                                                                                                                                                                                                 |
| 00100h              | Reinicio del sistema                       | Activación de las señales de entrada de reinicio físico o lógico del procesador por la lógica externa.                                                                                          |
| 00200h              | Chequeo de la máquina                      | Activación de TEA# en el procesador si está habilitado el reconocimiento de chequeos de la máquina.                                                                                             |
| 00300h              | Almacenamiento de datos                    | Ejemplos: fallo de página de datos, violación de los derechos de acceso en carga/almacenamiento.                                                                                                |
| 00400h              | Almacenamiento de instrucciones            | Fallo de página de código; intento de captación de instrucción de un segmento de E/S; violación de los derechos de acceso.                                                                      |
| 00500h              | Externa                                    | Activación de la señal de entrada al procesador de interrupción externa por la lógica externa, si está habilitado el reconocimiento de interrupciones externas.                                 |
| 00600h              | Alineación                                 | Intento fallido de acceder a memoria debido a un operando mal alineado.                                                                                                                         |
| 00700h              | Programa                                   | Interrupción de coma flotante; el usuario intenta ejecutar una instrucción privilegiada; se ejecutó una instrucción de intercepción con un código de condición específico; instrucción ilícita. |
| 00800h              | Coma flotante no disponible                | Intento de ejecutar una instrucción de coma flotante si la unidad de coma flotante se encuentra inhabilitada.                                                                                   |
| 00900h              | Decrementador                              | Agotamiento del registro decrementador si está habilitado el reconocimiento de interrupciones externas.                                                                                         |
| 00A00h              | Reservada                                  |                                                                                                                                                                                                 |
| 00B00h              | Reservada                                  |                                                                                                                                                                                                 |
| 00C00h              | Llamada al sistema                         | Ejecución de una instrucción de llamada al sistema.                                                                                                                                             |
| 00D00h              | Traza                                      | Interrupción paso a paso o de seguimiento de saltos.                                                                                                                                            |
| 00E00h              | Ayuda a coma flotante                      | Intento de ejecutar operaciones de coma flotante relativamente poco frecuentes o complejas (por ejemplo, operaciones con números denormalizados).                                               |
| 00E10h hasta 00FFFh | Reservada                                  |                                                                                                                                                                                                 |
| 01000h hasta 02FFFh | Reservada (depende de cada implementación) |                                                                                                                                                                                                 |

No sombreadas: interrupciones causadas por la ejecución de una instrucción

Sombreadas: interrupciones no causadas por la ejecución de una instrucción

### Registro de estado de la máquina (Machine State Register, MSR)

Para poder interrumpir un programa es fundamental que se tenga la capacidad de recuperar el estado que tenía el procesador en el momento de la interrupción. Esto incluye no sólo los contenidos de los diversos registros, sino también varias condiciones de control relativas a la ejecución. Estas condiciones se resumen convenientemente en el MSR (Tabla 11.6). De nuevo, algunos bits en este registro justifican comentarios adicionales.

Cuando el bit de modo de privilegio (bit 49) está a uno, el procesador opera en un nivel de privilegio de usuario. Sólo está disponible un subconjunto del repertorio de instrucciones. Cuando el bit se pone a cero, el procesador opera en el nivel de privilegio de supervisor. Ello permite todas las instrucciones, y proporciona acceso a ciertos registros del sistema (como el MSR) no accesibles desde el nivel de privilegio de usuario.

Tabla 11.6. Registro de estado de la máquina en el PowerPC

| Bit   | Definición                                                                                 |
|-------|--------------------------------------------------------------------------------------------|
| 0     | El procesador está en modo de 32 bits/64 bits.                                             |
| 1:44  | Reservados.                                                                                |
| 45    | Gestión de energía habilitada/inhabilitada.                                                |
| 46    | Depende de la implementación.                                                              |
| 47    | Define si los gestores de interrupción se ejecutan en modo «big endian» o «little endian». |
| 48    | Interrupción externa habilitada/inhabilitada.                                              |
| 49    | Estado privilegiado/no privilegiado                                                        |
| 50    | Unidad de coma flotante disponible/no disponible.                                          |
| 51    | Interrupciones de chequeo de la máquina habilitadas/inhabilitadas.                         |
| 52    | Modo 0 de las excepciones de coma flotante.                                                |
| 53    | Ejecución paso a paso habilitada/inhabilitada.                                             |
| 54    | Seguimiento de saltos habilitado/inhabilitado.                                             |
| 55    | Modo 1 de las excepciones de coma flotante.                                                |
| 56    | Reservado.                                                                                 |
| 57    | La parte más significativa de la dirección de excepción es 000h/FFFh.                      |
| 58    | Traducción de direcciones de instrucción activa/inactiva.                                  |
| 59    | Traducción de direcciones de datos activa/inactiva.                                        |
| 60:61 | Reservados.                                                                                |
| 62    | Interrupción recuperable/irrecuperable.                                                    |
| 63    | El procesador está en modo «big endian»/«little endian».                                   |

No sombreados: copiados en SRR1

Sombreados: no copiados en SRR1

Los valores de los dos bits de excepción de coma flotante (bits 52 y 55) definen los tipos de interrupciones que puede generar la unidad de coma flotante. La interpretación es la siguiente:

| FE0 | FE1 | Interrupciones que se pueden reconocer |
|-----|-----|----------------------------------------|
| 0   | 0   | Ninguna                                |
| 0   | 1   | Imprecisas irrecuperables              |
| 1   | 0   | Imprecisas recuperables                |
| 1   | 1   | Precisas                               |

Cuando el bit de paso a paso (bit 53) está a uno, el procesador salta al gestor de interrupción de traza después de la finalización con éxito de cada instrucción. Cuando el bit de seguimiento de saltos (bit 54) está a uno, el procesador salta al gestor de interrupción de traza de saltos después de la terminación con éxito de cada instrucción de salto, se produzca el salto o no.

La traducción de direcciones de instrucción (bit 58) y la traducción de direcciones de datos (bit 59) determinan si se usa direccionamiento real o si la unidad de gestión de memoria lleva a cabo traducción de direcciones.

### Manipulación de interrupciones

Cuando ocurre una interrupción, y es reconocida por el procesador, tiene lugar la siguiente secuencia de eventos:

- El procesador coloca la dirección de la siguiente instrucción a ejecutar en el registro salvar/restaurar 0 (Save/Restore Register 0, SRRO). Ésta es la dirección de la instrucción que se ejecuta en ese momento si la interrupción tuvo lugar por un intento fallido de ejecutar la instrucción; en cualquier otro caso, se ejecuta la dirección de la instrucción siguiente a ejecutar después de la actual.
- El procesador copia información sobre el estado de la máquina desde el MSR al registro salvar/restaurar 1 (Save/Restore Register 1, SRR1). Se copian los bits representados sin sombra en la Tabla 11.6. El resto de los bits de SRR1 se cargan con información específica para cada tipo de interrupción.
- El MSR se fija a un valor, definido por hardware, específico para cada tipo de interrupción. Para todos los tipos de interrupción, la traducción de direcciones se desactiva y se inhabilitan las interrupciones externas.
- El procesador transfiere el control al gestor de interrupción apropiado. Las direcciones de los gestores de interrupción están almacenadas en la tabla de interrupciones (Tabla 11.5). La dirección base de la tabla viene determinada por el bit 57 del MSR.

Para retornar de una interrupción, la rutina de servicio de interrupción ejecuta una instrucción *reti* (return from interrupt, «retorno de interrupción»). Ésta hace que los valores de los bits almacenados en SRR1 se recuperen en el MSR. La ejecución se reanuda en la posición almacenada en SRRO.

#### 11.7 LECTURAS RECOMENDADAS

[HENN91] y [HWAN93] contienen discusiones detalladas sobre la segmentación de cauce. [SOHI90] ofrece una excelente y detallada discusión sobre los aspectos del diseño hardware implicados en un cauce de instrucciones. [CRAG92] es un estudio detallado de la predicción de saltos en cauces de instrucciones. [DUBE91] y [LILJ88] examinan varias estrategias de predicción de saltos que se pueden usar para aumentar las prestaciones de la segmentación de instrucciones. [KAEL91] examina la dificultad que introducen en la predicción de saltos las instrucciones cuya dirección de destino es variable. El cauce de instrucciones del Intel 80486 se describe en [TABA91].

[BREY97] cubre ampliamente el procesamiento de interrupciones en el Pentium, lo mismo que [SHAN95a] para el PowerPC.

BREY97 Brey, B. *The Intel Microprocessors: 8086/8088, 80186/80188, 80286, 80386, 80486, Pentium, and Pentium Processor*. Upper Saddle River, NJ: Prentice Hall, 1997.

CRAG92 Cragon, H. *Branch Strategy Taxonomy and Performance Models*. Los Alamitos, CA: IEEE Computer Society Press, 1992.

DUBE91 Dubey, P., y Flynn, M. «Branch Strategies: Modeling and Optimization.» *IEEE Transactions on Computers*, October, 1991.

HENN91 Hennessy, J., y Jouppi, N. «Computer Technology and Architecture: An Evolving Interaction.» *Computer*, September, 1991.

HWAN93 Hwang, K. *Advanced Computer Architecture*. New York: McGraw-Hill, 1993.

KAEL91 Kaeli, D., y Emma, P., «Branch History Table Prediction of Moving Target Branches Due to Subroutine Returns.» *Proceeding, 18th Annual International Symposium on Computer Architecture*, May, 1991.

LILJ88 Lilja, D. «Reducing the Branch Penalty in Pipelined Processors.» *Computer*, July, 1988.

- SHAN95a Shanley, T. *PowerPC System Architecture*. Reading, MA: Addison-Wesley,
- SOHI90 Sohi, G. «Instruction Issue Logic for High-Performance Interruptible, Multi-Functional Unit, Pipelined Computers.» *IEEE Transactions on Computers*, March, 1990.
- TABA91 Tabak, D. *Advanced Microprocessors*. New York: McGraw-Hill, 1991.

### 11.8. PROBLEMAS

- 11.1. a) Si la última operación ejecutada en un computador con palabras de 8 bits fue suma en la que los dos operandos eran 2 y 3, ¿cuál sería el valor de los siguientes indicadores?
- Acarreo
  - Cero
  - Desbordamiento
  - Signo
  - Paridad par
  - Acarreo intermedio
- b) ¿Y si los operandos fueran  $-1$  (en complemento a dos) y  $+1$ ?
- 11.2. Considere el diagrama de tiempos de la Figura 11.11. Asuma que hay un cauce de sólo dos etapas (captar, ejecutar). Redibuje el diagrama para mostrar cuántas unidades de tiempo se necesitan ahora para cuatro instrucciones.
- 11.3. Considere una secuencia de instrucciones de longitud  $n$  que atraviesa un cauce de instrucciones. Sea  $p$  la probabilidad de encontrar una instrucción de bifurcación condicional o incondicional, y sea  $q$  la probabilidad de que la ejecución de una instrucción de bifurcación I provoque un salto a una dirección no consecutiva. Suponga que cada salto de este tipo requiere vaciar el cauce, destruyendo todo el procesamiento de instrucciones en marcha, cuando I sale de la última etapa. Modifique las Ecuaciones (11.1) y (11.2) de modo que tengan en cuenta estas probabilidades.
- 11.4. Una limitación de la aproximación de flujos múltiples para tratar los saltos en un cauce es que se encuentren saltos adicionales antes de que se resuelva el primero. Sugiera otras dos limitaciones o desventajas.
- 11.5. Considere los diagramas de estado de la Figura 11.24.
- Describa el funcionamiento de cada uno.
  - Compárelos con el diagrama de estados de la predicción de saltos de la Sección 11.4. Discuta las ventajas relativas de cada una de las tres aproximaciones para la predicción de saltos.
- 11.6. Las máquinas Motorola 680×0 incluyen la instrucción «decrementar y saltar según la condición», que tiene la siguiente forma:

DBcc Dn, desplazamiento

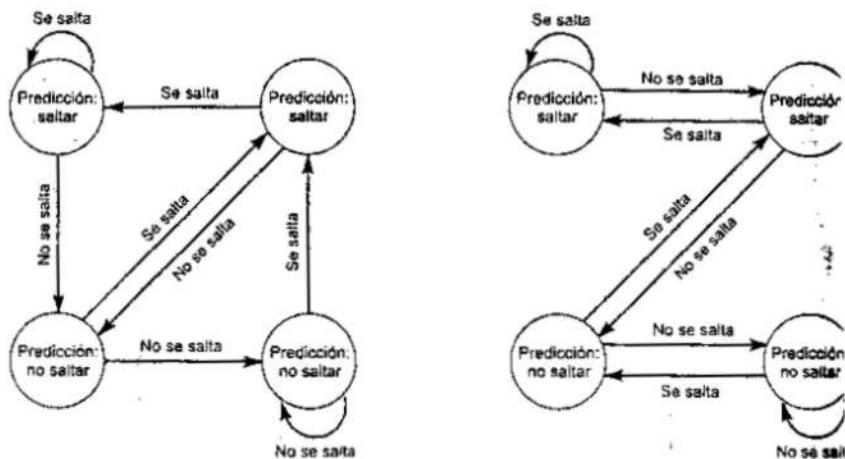


Figura 11.24. Diagramas de estado del Problema 11.5.

donde cc es una de las condiciones que se pueden examinar, Dn es un registro de uso general, y el desplazamiento especifica la dirección de destino relativa a la dirección actual. La instrucción se puede definir como sigue:

```

if (cc = False)
then begin
 Dn := (Dn) - 1;
 if Dn ≠ -1 then PC := (PC) + desplazamiento end
else PC := (PC) + 2;

```

Cuando se ejecuta la instrucción, primero se examina la condición, para determinar si se satisface la condición de terminación del bucle. Si es así, no se realiza ninguna operación, y la ejecución continúa secuencialmente por la siguiente instrucción. Si la condición es falsa, se decremente el registro de datos especificado y se comprueba si es menor que cero. Si lo es, el bucle termina, y la ejecución continúa secuencialmente por la siguiente instrucción. En caso contrario, el programa salta a la posición especificada. Considere ahora el siguiente fragmento de programa en lenguaje ensamblador:

```

OTRA_VEZ CMPM.L (A0)1, (A1)1
 DBNE D1, OTRA_VEZ
 NOP ←

```

Dos cadenas direccionadas por A0 y A1 se comparan para ver si son iguales; los punteros a las cadenas se incrementan en cada referencia. D1 contiene inicialmente el número de palabras largas (4 bytes) a comparar, menos uno.

- Los contenidos iniciales de los registros son A0 = \$00004000, A1 = \$00005000, D1 = \$000000FF (S indica notación hexadecimal). La memoria entre \$4000 y \$6000 se carga con palabras SAAAA. Si se ejecuta el programa anterior, especí

que el número de veces que se ejecuta el bucle DBNE y los contenidos de los tres registros cuando se llega a la instrucción NOP.

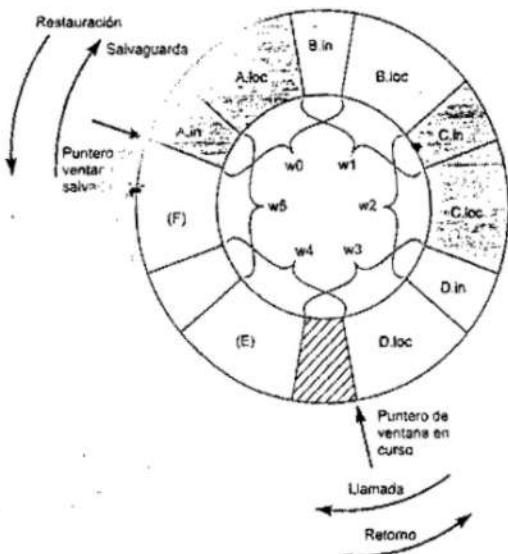
- b) Repita el Apartado (a), pero suponiendo ahora que la memoria entre \$4000 y \$4FEE se carga con \$0000, y las posiciones entre \$5000 y \$6000 contienen \$AAA.

11.7. Redibuje la Figura 11.18c, suponiendo que el salto condicional no se produce.

## CAPÍTULO 12

# Computadores de repertorio reducido de instrucciones

- 12.1. Características de la ejecución de instrucciones
  - Operaciones
  - Operandos
  - Llamadas a procedimientos
  - Implicaciones
- 12.2. Utilización de un amplio banco de registros
  - Ventanas de registros
  - VARIABLES globales
  - Un amplio banco de registros frente a una cache
- 12.3. Optimización de registros basada en el compilador
- 12.4. Arquitectura de repertorio reducido de instrucciones
  - ¿Por qué CISC?
  - Características de las arquitecturas de repertorio reducido de instrucciones
  - Características CISC frente a RISC
- 12.5. Segmentación de cauce en RISC
  - Segmentación con instrucciones regulares
  - Optimización de la segmentación
- 12.6. MIPS R4000
  - Repertorio de instrucciones
  - Segmentación de instrucciones
- 12.7. SPARC
  - Conjunto de registros del SPARC
  - Repertorio de Instrucciones
  - Formato de las instrucciones
- 12.8. La controversia entre RISC y CISC
- 12.9. Lecturas recomendadas
- 12.10. Problemas



- Los estudios del comportamiento de la ejecución de los programas escritos en lenguajes de alto nivel, proporcionaron la orientación para diseñar un nuevo tipo de arquitectura del procesador: el computador de repertorio reducido de instrucciones (*«reduced instruction set computer»*, RISC). Las sentencias de asignación predominan, lo que sugiere que la sencilla transferencia de datos debería optimizarse. Hay también muchas instrucciones IF y LOOP, lo que sugiere que el mecanismo de control de secuencia subyacente ha de ser optimizado para permitir una segmentación eficiente. Los estudios de los patrones de referencia a operandos sugieren que debería de ser posible mejorar las prestaciones, teniendo guardados un número moderado de operandos en registros.
- Estos estudios han motivado las características fundamentales de las máquinas RISC: (1) un repertorio de instrucciones limitado y con un formato fijo, (2) un número grande de registros o la utilización de un compilador que optimice el uso de éstos, y (3) un énfasis en la optimización del cauce de instrucciones.
- El repertorio de instrucciones sencillo de un RISC se presta a una segmentación eficiente, porque hay menos operaciones llevadas a cabo por instrucción, y éstas son más previsibles. Una arquitectura de repertorio de instrucciones RISC también se presta a la técnica de salto retardado, en la cual las instrucciones de salto se reubican entre otras instrucciones para mejorar la eficiencia del cauce.

**D**esde el desarrollo del computador de programa almacenado alrededor del año 1950, ha habido extraordinariamente pocas innovaciones auténticas en las áreas de organización y arquitectura de computadores. Los siguientes son algunos de los principales avances desde el nacimiento del computador.

- **El concepto de familia:** Introducido por IBM en su System/360 en 1964, y seguido poco después por DEC, en su PDP-8. El concepto de familia separa la arquitectura de una máquina de su implementación. Se ofrece un conjunto de computadores, con distintas características en cuanto a precio y prestaciones, que presentan al usuario la misma arquitectura. Las diferencias en precio y prestaciones se deben a las distintas implementaciones de la misma arquitectura.
- **Unidad de control microprogramada:** Propuesta por Wilkes en 1951, e introducida por IBM en la línea S/360 en 1964. La microprogramación facilita la tarea de diseñar e implementar la unidad de control y da soporte al concepto de familia.
- **Memoria cache:** Introducida comercialmente por primera vez en el IBM S/360 Modelo 85 en 1968. La introducción de este elemento en la jerarquía de memoria mejoró las prestaciones de manera espectacular.
- **Segmentación de cauce:** Una manera de introducir paralelismo en la naturaleza esencialmente secuencial de un programa constituido por instrucciones máquina. Dos ejemplos son la segmentación de instrucciones y el procesamiento vectorial.
- **Múltiples procesadores:** Esta categoría cubre diferentes organizaciones y objetivos.

A esta lista hay que añadir una de las más interesantes y, potencialmente, de las más importantes innovaciones: la arquitectura de computador de repertorio reducido de instrucciones (*«reduced instruction set computer»*, RISC). La arquitectura RISC se aparta de manera drástica de la tendencia histórica en la arquitectura del procesador. Un análisis de la arquitectura RISC involucra a la mayoría de los asuntos importantes de organización y arquitectura de computadores.

Aunque los sistemas RISC se han definido y diseñado de diversas formas por parte de diferentes grupos, los elementos clave compartidos por la mayoría (no todos) de los diseños son:

- Un gran número de registros de uso general, o el uso de tecnología de compiladores para optimizar el uso de los registros.
- Un repertorio de instrucciones limitado y sencillo.
- Un énfasis en la optimización de la segmentación de instrucciones.

La Tabla 12.1 compara varios sistemas RISC y no RISC.

Comenzamos este capítulo con una revisión breve de algunos resultados referentes a los repertorios de instrucciones, y después examinamos cada uno de los tres asuntos recién mencionados. Sigue a esto una descripción de dos de los diseños RISC mejor documentados.

## 12.1. CARACTERÍSTICAS DE LA EJECUCIÓN DE INSTRUCCIONES

Uno de los aspectos relacionados con los computadores que ha evolucionado de manera más visible es el de los lenguajes de programación. Conforme el coste del hardware ha disminuido, el coste relativo del software ha ido creciendo. Junto con esto, la escasez permanente de programadores ha hecho subir los costes del software en términos absolutos. De ahí que el coste principal del ciclo de vida de un sistema sea el software, no el hardware. Otro elemento

Tabla 12.1. Características de algunos procesadores CISC, RISC y superescalares

|                                         | Computador de repertorio complejo de instrucciones (CISC) |            |             | Computador de repertorio reducido de instrucciones (RISC) |            | Superescalar |             |             |
|-----------------------------------------|-----------------------------------------------------------|------------|-------------|-----------------------------------------------------------|------------|--------------|-------------|-------------|
|                                         | IBM 370/168                                               | VAX 11/780 | Intel 80486 | SPARC                                                     | MIPS R4000 | PowerPC      | Ultra SPARC | MIPS R10000 |
| Año de desarrollo                       | 1973                                                      | 1978       | 1989        | 1987                                                      | 1991       | 1993         | 1996        | 1996        |
| Número de instrucciones                 | 208                                                       | 303        | 235         | 69                                                        | 94         | 225          |             |             |
| Tamaño de instrucción (bytes)           | 2-6                                                       | 2-57       | 1-11        | 4                                                         | 4          | 4            | 4           | 4           |
| Modos de direccionamiento               | 4                                                         | 22         | 11          | 1                                                         | 1          | 2            | 1           | 1           |
| Número de registros de uso general      | 16                                                        | 16         | 8           | 40-520                                                    | 32         | 32           | 40-520      | 32          |
| Tamaño de la memoria de control (kbits) | 420                                                       | 480        | 246         | —                                                         | —          | —            | —           | —           |
| Tamaño de cache (kbytes)                | 64                                                        | 64         | 8           | 32                                                        | 128        | 16-32        | -32         | 64          |

que se añade al coste y a otros inconvenientes, es la falta de fiabilidad: es frecuente que los programas, tanto de sistema como de aplicación, continúen mostrando nuevos errores después de años de funcionamiento.

La respuesta de los investigadores y de la industria fue desarrollar lenguajes de programación de alto nivel más potentes y complejos. Estos lenguajes de alto nivel («high-level languages», HLL) permiten al programador expresar los algoritmos de manera más concisa, se encargan de gran parte de los pormenores y, a menudo, apoyan de manera natural la programación estructurada o el diseño orientado a objetos.

Desgraciadamente, la solución ocasionó otro problema, conocido como el *salto semántico*: la diferencia entre las operaciones que proporcionan los HLL y las que proporciona la arquitectura del computador. Los supuestos síntomas de este salto o hueco incluyen: ineficiencia de la ejecución, tamaño excesivo del programa en lenguaje máquina y complejidad de los compiladores. Los diseñadores respondieron con arquitecturas que se proponían cerrar ese hueco. Sus características principales incluyen repertorios de instrucciones grandes, docenas de modos de direccionamiento y varias sentencias HLL implementadas en hardware. Un ejemplo de esto último es la instrucción máquina CASE del VAX. Tales repertorios complejos de instrucciones están pensados para:

- Facilitar el trabajo del escritor de compiladores.
- Mejorar la eficiencia de la ejecución, ya que las secuencias complejas de operaciones pueden implementarse en microcódigo.
- Dar soporte a HLL aun más complejos y sofisticados.

Mientras tanto, se hicieron algunos estudios durante años para determinar las características y patrones de ejecución de las instrucciones máquina generadas por programas escritos en HLL. Los resultados de estos estudios motivaron a algunos investigadores a buscar una aproximación diferente; a saber, realizar una arquitectura que diera soporte a los HLL más sencillos, en lugar de a los más complejos.

Para comprender la línea de razonamiento de los partidarios de los RISC, comenzamos con una breve revisión de las características de la ejecución de instrucciones. Los aspectos cuyo cálculo tiene interés son los siguientes:

- **Operaciones realizadas:** Determinan las funciones que lleva a cabo el procesador, y su interacción con la memoria.
- **Operandos usados:** Los tipos de operandos y su frecuencia de uso determinan la organización de memoria para almacenarlos y los modos de direccionamiento para accederlos.
- **Secuenciamiento de la ejecución:** Determina la organización del control y del cauce.

En el resto de esta sección, resumimos los resultados de varios estudios sobre programas escritos en lenguaje de alto nivel. Todos los resultados se basan en medidas dinámicas. Es decir, las medidas están tomadas ejecutando el programa y contando el número de veces que alguna característica ha aparecido o una propiedad concreta ha sido cierta. Por contraste, las medidas estáticas sólo realizan estas cuentas sobre el texto fuente del programa. Éstas no nos dan una información útil sobre las prestaciones, ya que no están ponderadas por el número de veces que cada sentencia se ejecuta.

## OPERACIONES

Se han hecho varios estudios para analizar el comportamiento de los programas escritos en HLL. La Tabla 4.9, que se discutió en el Capítulo 4, incluye los principales resultados de varios de estos estudios. Existe una gran concordancia en los resultados de esta mezcla de lenguajes y aplicaciones. Las sentencias de asignación predominan, lo cual indica que el sencillo movimiento de datos tiene mucha importancia. También predominan las sentencias condicionales (IF, LOOP). Estas sentencias se implementan en el lenguaje máquina con alguna instrucción del tipo «comparar y saltar». Esto indica que el mecanismo incluido en el repertorio de instrucciones para el control del secuenciamiento es importante.

Estos resultados son instructivos para el diseñador del repertorio de instrucciones máquina, porque le dicen qué tipos de sentencias tienen lugar más a menudo y, por consiguiente, deben ser implementadas de una forma «óptima». No obstante, estos resultados no revelan qué sentencias consumen más tiempo en la ejecución de un programa típico. Es decir, dado un programa en lenguaje máquina compilado, ¿qué sentencias del lenguaje fuente provocan la ejecución de la mayoría de las instrucciones en lenguaje máquina?

Para averiguar este fenómeno subyacente, los programas de Patterson [PATT82a], descritos en el apéndice 4A, se compilaron en VAX, PDP-11 y Motorola 68000 para determinar el número medio de instrucciones máquina y referencias a memoria por cada tipo de sentencia. La segunda y tercera columnas de la Tabla 12.2 muestran la frecuencia relativa de aparición de varias instrucciones de HLL en varios programas; los datos se obtuvieron observando las apariciones en programas en ejecución, en lugar de examinar sólo el número de veces que aparecen esas sentencias en el código fuente. Por tanto, se trata de estadísticas de frecuencia dinámica. Para obtener los datos de las columnas cuatro y cinco (instrucciones máquina ponderadas), cada valor de la segunda y tercera columnas se multiplicó por el número de instrucciones máquina generadas por el compilador. Estos resultados están además normali-

Tabla 12.2. Frecuencia dinámica relativa ponderada de operaciones en HLL [PATT82a]

|        | Aparición dinámica |    | Instrucciones máquina ponderadas |    | Referencias a memoria ponderadas |    |
|--------|--------------------|----|----------------------------------|----|----------------------------------|----|
|        | Pascal             | C  | Pascal                           | C  | Pascal                           | C  |
| ASSIGN | 45                 | 38 | 13                               | 13 | 14                               | 15 |
| LOOP   | 5                  | 3  | 42                               | 32 | 33                               | 26 |
| CALL   | 15                 | 12 | 31                               | 33 | 44                               | 45 |
| IF     | 29                 | 43 | 11                               | 21 | 7                                | 13 |
| GOTO   | —                  | 3  | —                                | —  | —                                | —  |
| OTRAS  | 6                  | 1  | 3                                | 1  | 2                                | —  |

zados, para que las columnas cuatro y cinco muestren la frecuencia relativa de aparición, ponderada por el número de instrucciones máquina por sentencia de HLL. De un modo parecido, la sexta y séptima columnas se obtienen multiplicando la frecuencia de aparición de cada tipo de sentencia por el número relativo de referencias a memoria originado por esa sentencia. Los datos en las columnas de la cuatro a la siete proporcionan medidas sustitutivas del tiempo real empleado en la ejecución de varios tipos de sentencias. Los resultados indican que la llamada/retorno de procedimientos es la operación que consume más tiempo en los programas típicos escritos en HLL.

El lector debe tener claro el significado de la Tabla 12.2. Esta tabla indica la importancia relativa de varios tipos de sentencias de un HLL, cuando ese HLL se compila para una arquitectura típica con un repertorio de instrucciones actual. Posiblemente, alguna otra arquitectura puede producir diferentes resultados. Sin embargo, este estudio produce resultados representativos de las arquitecturas contemporáneas de repertorio complejo de instrucciones (CISC). De ahí que estos resultados puedan servir de orientación a quienes busquen formas más eficientes de dar soporte a los HLL.

## OPERANDOS

Se han hecho muchos menos estudios sobre la aparición de los distintos tipos de operandos, a pesar de la importancia de este tema. Hay varios aspectos que son significativos.

El estudio de Patterson ya citado [PATT82a], también consideró la frecuencia dinámica de aparición de distintas clases de variables (Tabla 12.3). Los resultados, coherentes para programas en Pascal y en C, muestran que la mayoría de las referencias se hacen a variables escalares simples. Además, más del 80 % de los datos escalares eran variables locales (al procedimiento). Asimismo, las referencias a matrices/estructuras requieren una referencia previa a su índice o puntero, que nuevamente suele ser un dato escalar local. De este modo, hay un predominio de referencias a operandos escalares, y éstos están muy localizados.

El estudio de Patterson examinó el comportamiento dinámico de los programas en HLL de manera independiente de la arquitectura subyacente. Como se discutió antes, es preciso analizar las arquitecturas reales para examinar el comportamiento del programa con mayor profundidad. Un estudio, [LUND77], examinó dinámicamente las instrucciones del DEC-10, y encontró que cada instrucción referencia, en promedio, 0,5 operandos de memoria y 1,4 registros. En [HUCK83] se ofrecen resultados semejantes para programas en C, Pascal y FORTRAN, ejecutados en los computadores S/370, PDP-11 y VAX. Naturalmente, estas cifras dependen mucho de la arquitectura y del compilador, pero sirven para ilustrar la frecuencia de acceso a los operandos.

Tabla 12.3. Porcentaje dinámico de operandos

|                      | Pascal | C  | Promedio |
|----------------------|--------|----|----------|
| Constantes enteras   | 16     | 23 | 20       |
| VARIABLES ESCALARES  | 58     | 53 | 55       |
| Matrices/estructuras | 26     | 24 | 25       |

Estos últimos estudios reflejan la importancia de una arquitectura que se preste a un rápido acceso a operandos, dado que es una operación que se realiza con mucha frecuencia. El estudio de Patterson indica qué un candidato fundamental a optimizar, es el mecanismo de almacenamiento y acceso a variables escalares locales.

## LLAMADAS A PROCEDIMIENTOS

Hemos visto que las llamadas y retornos de procedimientos constituyen un aspecto importante de los programas en HLL. Los datos (Tabla 12.2) indican que son las operaciones que consumen más tiempo en programas en HLL compilados. Así, será ventajoso considerar formas de implementar estas operaciones eficientemente. Hay dos aspectos importantes: el número de parámetros y variables que trata un procedimiento, y la profundidad de anidamiento.

El estudio de Tanenbaum [TANE78] encontró que al 98 % de los procedimientos llamados dinámicamente se les pasaban menos de seis argumentos, y que el 92 % de ellos usaban menos de seis variables escalares locales. El equipo de RISC de Berkeley presentó resultados parecidos [KATE83], como se puede ver en la Tabla 12.4. Estos resultados muestran que el número de palabras necesarias por cada activación de un procedimiento no es muy grande. Los resultados expuestos anteriormente indicaban que una alta proporción de referencias a operandos se hacía a variables locales escalares. Estos estudios muestran que aquellas referencias se limitan, de hecho, a relativamente pocas variables.

El mismo grupo de Berkeley también estudió los patrones seguidos por las llamadas y retornos de procedimientos de programas en HLL. Encuentran que es poco común tener una larga secuencia ininterrumpida de llamadas a procedimientos seguida por la correspondiente secuencia de retornos. Vieron, más bien, que un programa permanece confinado en una ventana bastante estrecha de profundidad de invocación a procedimientos. Esto lo ilustró la Figura 4.29, que se discutió en el Capítulo 4. Tales resultados refuerzan la conclusión de que las referencias a operandos están muy localizadas.

Tabla 12.4. Argumentos de procedimientos y variables locales escalares

| Porcentaje de llamadas a procedimientos ejecutadas con: | Compilador, intérprete y composición de textos | Pequeños programas no numéricos |
|---------------------------------------------------------|------------------------------------------------|---------------------------------|
| > 3 argumentos                                          | 0-7 %                                          | 0-5 %                           |
| > 5 argumentos                                          | 0-3 %                                          | 0 %                             |
| > 8 palabras para argumentos y datos escalares locales  | 1-20 %                                         | 0-6 %                           |
| > 12 palabras para argumentos y datos escalares locales | 1-6 %                                          | 0-3 %                           |

## CONSECUENCIAS

Varios grupos han estudiado resultados como los que se acaban de exponer, y han llegado a la conclusión de que intentar realizar una arquitectura de repertorio de instrucciones cercano al de los HLL, no es la estrategia de diseño más efectiva. En su lugar, se puede ofrecer mejor soporte para los HLL optimizando las prestaciones de las características de los programas típicos en HLL que más tiempo consumen.

Partiendo del trabajo de varios investigadores, surgen tres elementos que, por lo general, caracterizan las arquitecturas RISC. Primero, usar un gran número de registros, o un compilador que optimice el tratamiento de éstos. Su finalidad es optimizar las referencias a operandos. Los estudios recién discutidos muestran que hay varias referencias por instrucción de HLL, y que hay una alta proporción de sentencias de transferencia (asignación). Esto, asociado con la localidad y predominio de las referencias a datos escalares, sugiere que las prestaciones pueden mejorarse reduciendo las referencias a memoria, a costa de más referencias a registros. Debido a la localidad de estas referencias, parece práctico un conjunto de registros ampliado.

En segundo lugar, hay que prestar una cuidadosa atención al diseño de los cauces de instrucciones. Debido a la alta proporción de instrucciones de bifurcación condicional y de llamada a procedimientos, un cauce de instrucciones sencillo será ineficiente. Esto se manifiesta debido a que una gran proporción de instrucciones se precantan, pero nunca llegan a ejecutarse.

Por último, es recomendable un repertorio de instrucciones simplificado (reducido). Este punto no es tan obvio como los otros, pero debería quedar más claro tras la siguiente discusión.

## 22 UTILIZACIÓN DE UN AMPLIO BANCO DE REGISTROS

Los resultados resumidos en la Sección 12.1 indican lo deseable que es un rápido acceso a los operandos. Hemos visto que hay una gran proporción de sentencias de asignación en programas escritos en HLL, y muchas de estas son de la forma sencilla A → B. Además, hay un número significativo de accesos a operandos por cada sentencia de HLL. Si juntamos estos resultados con el hecho de que la mayoría de los accesos se hacen a datos escalares locales, se puede pensar en una fuerte dependencia del almacenamiento en registros.

La razón de que esté indicado ese almacenamiento en registros es que éstos constituyen el dispositivo de almacenamiento más rápido disponible; más que la memoria principal y que la cache. El banco de registros es pequeño físicamente, está en el mismo chip que la ALU y la unidad de control, y emplea direcciones mucho más cortas que las de la cache y la memoria. Por tanto, se necesita una estrategia que permita que los operandos a los que se accede con mayor frecuencia se encuentren en registros, y que se minimicen las operaciones registro-memoria.

Son posibles dos aproximaciones básicas: una basada en software y la otra en hardware. La aproximación por software consiste en confiar al compilador la maximización del uso de los registros. El compilador intentará asignar registros a las variables que se usen más en un período de tiempo dado. Esta aproximación requiere el uso de sofisticados algoritmos de análisis de programas. La aproximación por hardware consiste sencillamente en usar más registros, de manera que más variables puedan mantenerse en registros durante períodos de tiempo más largos.

En esta sección, examinaremos la aproximación por hardware. Este enfoque se inició en el grupo de RISC de Berkeley [PATT82a], se usó en el primer producto RISC comercial, el Pyramid [RAGA83], y se utiliza en la actualidad en la conocida arquitectura SPARC.

## VENTANAS DE REGISTROS

A primera vista, el uso de un conjunto amplio de registros debería reducir la necesidad de acceder a memoria. La labor del diseño es organizar los registros de tal modo que se alcance esa meta.

Ya que la mayoría de las referencias a operandos se hacen a datos escalares locales, la solución evidente es almacenar éstos en registros, reservando tal vez varios registros para variables globales. El problema es que la definición de *local* varía con cada llamada y retorno de procedimiento, operaciones que suceden con frecuencia. En cada llamada, las variables locales deben guardarse desde los registros en la memoria, para que los registros puedan ser reutilizados por el programa llamado. Además, deben pasarse parámetros. En el retorno, las variables del programa padre tienen que ser restablecidas (cargadas de nuevo en los registros) y hay que devolver los resultados al programa padre.

La solución se basa en otros dos resultados presentados en la Sección 12.1. En primer lugar, un procedimiento típico emplea sólo unos pocos parámetros de llamada y variables locales (Tabla 12.4). En segundo lugar, la profundidad de activación de procedimientos fluctúa dentro de un rango relativamente pequeño (Figura 4.29). Para explotar estas propiedades, se usan múltiples conjuntos pequeños de registros, cada uno asignado a un procedimiento distinto. Una llamada a un procedimiento hace que el procesador commute automáticamente a una ventana de registros distinta de tamaño fijo, en lugar de salvaguardar los registros en memoria. Las ventanas de procedimientos adyacentes están parcialmente solapadas para permitir el paso de parámetros.

El concepto se ilustra en la Figura 12.1. En todo momento sólo es visible una ventana de registros, y se direcciona como si fuera el único conjunto de registros (por ejemplo, direcciones 0 a  $N - 1$ ). La ventana se divide en tres áreas de tamaño fijo. Los registros de parámetros contienen parámetros pasados al procedimiento en curso desde el procedimiento que lo llamó, y los resultados a devolver a éste. Los registros locales se usan para variables locales, según la asignación que realice el compilador. Los registros temporales se usan para intercambiar parámetros y resultados con el siguiente nivel más bajo (el procedimiento llamado por el procedimiento en curso). Los registros temporales de un nivel son físicamente los mismos que los registros de parámetros del nivel más bajo adyacente. Este sólapeamiento posibilita que los parámetros se pasen sin que exista una transferencia de datos real.



Figura 12.1. Ventanas de registros solapadas.

Para manejar cualquier posible patrón de llamadas y retornos, el número de ventana registros tendría que ser ilimitado. En lugar de eso, las ventanas de registros se pueden para contener unas pocas (las más recientes) activaciones de procedimientos. Las activaciones más antiguas han de salvaguardarse en memoria y restaurarse más tarde, cuando la profundidad de anidamiento disminuya. De este modo, la organización real del banco de registros es un buffer circular de ventanas solapadas.

Esta organización se muestra en la Figura 12.2, que representa un buffer circular de 8 ventanas. El buffer está lleno hasta una profundidad de 4 (A llamó a B; B llamó a C; y C llamó a D), siendo D el procedimiento activo. El puntero de ventana en curso (*«current window pointer»*, CWP) apunta a la ventana del procedimiento activo actualmente. Las referencias de una instrucción máquina a registros se transforman con este puntero para determinar el registro físico real. El puntero de ventana salvada (*«saved-window pointer»*, SWP) identifica la ventana guardada en memoria más recientemente. Si el procedimiento D llama ahora al procedimiento E, los argumentos para E se colocan en los registros temporales de I (el solapamiento entre w3 y w2), y el CWP se avanza en una ventana.

Si el procedimiento E hace después una llamada al procedimiento F, ésta no puede llevarse a cabo con el estado actual del buffer. Esto se debe a que la ventana de F se solapa con la de A. Si F comienza a cargar sus registros temporales, como preparación para una llamada, esto sobrescribirá los registros de parámetros de A (A.in). Por tanto, cuando al incrementar el CWP (módulo 6), éste llega a ser igual a SWP, tiene lugar una interrupción, y la ventana de A se guarda. Sólo hay que guardar las dos primeras partes (A.in y A.loc). Después se incrementa el SWP, y la llamada a F sigue adelante. En los retornos ocurre una interrupción similar. Por ejemplo, con posterioridad a la activación de F, cuando B retorna a A, el CWP se decremente y llega a ser igual a SWP. Esto causa una interrupción, cuyo resultado es la restauración de la ventana de A.

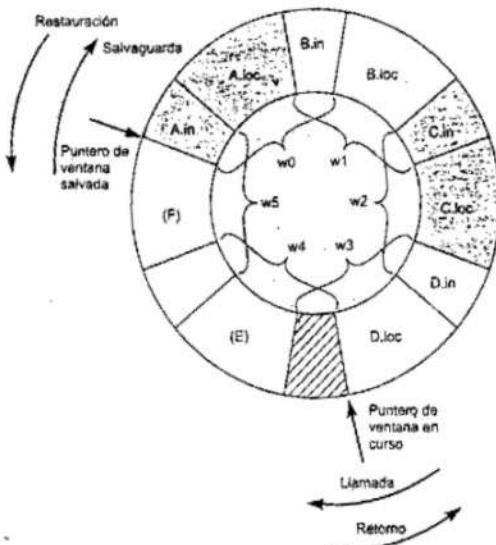


Figura 12.2. Organización de las ventanas solapadas como un buffer circular.

De lo anterior, se puede deducir que un banco de registros de  $N$  ventanas puede contener sólo  $N - 1$  activaciones de procedimientos. El valor de  $N$  no tiene que ser muy grande. Como mencionamos en el apéndice 4A, un estudio, [TAMI83], encontró que, con 8 ventanas, se requiere una salvaguarda o restauración sólo en el 1% de las llamadas o retornos. Los computadores RISC de Berkeley usan 8 ventanas de 16 registros cada una. El computador Pyramid emplea 16 ventanas de 32 registros cada una.

## VARIABLES GLOBALES

El esquema de ventanas recién descrito proporciona una organización eficiente para el almacenamiento de variables escalares locales en registros. Sin embargo, este esquema no trata la necesidad de almacenar las variables globales: aquellas a las que accede más de un procedimiento. Se sugieren dos opciones. La primera es que el compilador asigne posiciones de memoria a las variables declaradas como globales en un HLL, y que todas las instrucciones máquina que refieran esas variables usen operandos referenciados en memoria. Esto es sencillo desde los puntos de vista hardware y software (compilador). No obstante, para variables globales a las que se accede frecuentemente, este esquema es ineficiente.

Una alternativa es incorporar al procesador un conjunto de registros globales. Estos registros serían fijos en cuanto a su número, y accesibles a todos los procedimientos. Se puede usar un esquema de numeración unificado para simplificar el formato de instrucción. Por ejemplo, las referencias a los registros desde el 0 hasta el 7 pueden indicar registros globales únicos, y las referencias a los registros del 8 al 31 pueden transformarse para seleccionar los registros físicos de la ventana en curso. Esto conlleva un hardware añadido, que se encarga de adaptar la división en el direccionamiento de los registros. Además, el compilador ha de decidir qué variables globales deben ser asignadas a registros.

## UN AMPLIO BANCO DE REGISTROS FRENTÉ A UNA CACHE

El banco de registros organizado en ventanas funciona como un buffer pequeño y rápido, que contiene un subconjunto de todas las variables que probablemente se usen mucho. Desde este punto de vista, el banco de registros se comporta de manera muy similar a una memoria cache. Surge por tanto la cuestión de si sería más sencillo y mejor, usar una cache y un tradicional banco de registros pequeño.

La Tabla 12.5 compara algunas características de las dos soluciones. El banco de registros basado en ventanas contiene todas las variables escalares locales (excepto en el caso poco frecuente de desbordamiento de ventana) de las  $N - 1$  activaciones de procedimientos más recientes. La cache contiene una selección de las variables escalares usadas recientemente.

Tabla 12.5. Características de las organizaciones de un banco de registros amplio y de cache

| Banco de registros amplio                                         | Cache                                                         |
|-------------------------------------------------------------------|---------------------------------------------------------------|
| Todos los datos escalares locales                                 | Datos escalares locales recientemente usados                  |
| Variables individuales                                            | Bloques de memoria                                            |
| Variables globales asignadas por el compilador                    | Variables globales usadas recientemente                       |
| Salvaguarda/restauración basadas en la profundidad de anidamiento | Salvaguarda/restauración basadas en el algoritmo de reemplazo |
| Direccionamiento de registros                                     | Direccionamiento de memoria                                   |

mente. El banco de registros debería ahorrar tiempo, ya que guarda todas las variables escalares locales. Por otra parte, la cache puede hacer un uso más eficiente del espacio, ya que reacciona ante las situaciones dinámicamente. Además, las caches generalmente tratan todas las referencias a memoria del mismo modo, incluyendo las instrucciones y otros tipos de datos. Así, es posible un ahorro en estas otras áreas con la cache, y no con un banco de registros.

Un banco de registros puede hacer un uso ineficiente del espacio, puesto que no todos los procedimientos necesitarán el espacio de ventana completo asignado a ellos. Por otro lado, la cache adolece de otro tipo de ineficiencia: los datos se leen en la cache por bloques. Mientras que el banco de registros contiene sólo las variables en uso, la cache lee en un bloque de datos algo, o mucho, que no se usará.

La cache es capaz de manipular tanto variables globales como locales. Por regla general, hay muchos datos escalares globales, pero, de ellos, sólo unos pocos se usan mucho [KATE83]. Una cache descubrirá dinámicamente esas variables, y las almacenará. Si el banco de registros basado en ventanas se complementa con registros globales, también puede contener algunos datos escalares globales. Sin embargo, para un compilador es difícil determinar qué datos globales se usarán mucho.

Con el banco de registros, las transferencias de datos entre registros y memoria quedan determinada por la profundidad de anidamiento de los procedimientos. Como esta profundidad normalmente fluctúa en un estrecho margen, el acceso a memoria es relativamente poco frecuente. Casi todas las memorias cache son asociativas por conjuntos con un tamaño de conjunto pequeño. De este modo, existe el peligro de que otros datos o instrucciones sobrescriban las variables usadas con frecuencia.

Si nos basamos en lo discutido hasta aquí, la elección entre un amplio banco de registros basado en ventanas y una cache no está clara. Existe una característica, no obstante, en la cual la aproximación de los registros es claramente superior, y que nos indica que un sistema basado en cache será notablemente más lento. La distinción viene dada por la sobrecarga de direccionamiento experimentada por cada una de las dos aproximaciones.

La Figura 12.3 ilustra la diferencia. Para referenciar un dato escalar local en un banco de registros basado en ventanas, se usan un número de registro «virtual» y un número de ventana. Los dos pueden proporcionarse a un decodificador relativamente sencillo para seleccionar uno de los registros físicos. Para referenciar una posición de memoria en la cache, hay que generar una dirección de memoria completa. La complejidad de esta operación depende del modo de direccionamiento. En una cache asociativa por conjuntos, una parte de la dirección se usa para leer un número de palabras y etiquetas igual al tamaño del conjunto. Otra parte de la dirección se compara con las etiquetas, y se selecciona una de las palabras leídas. Debe quedar claro que, incluso en el caso de que la cache sea tan rápida como el banco de registros, el tiempo de acceso será considerablemente más grande. Por consiguiente, desde el punto de vista de las prestaciones, el banco de registros basado en ventanas es superior para datos escalares locales. Se puede conseguir una mejora adicional en las prestaciones añadiendo una cache sólo para instrucciones.

### 12.3. OPTIMIZACIÓN DE REGISTROS BASADA EN EL COMPILADOR

Supongamos ahora que disponemos de una máquina RISC que contiene únicamente un pequeño número de registros (por ejemplo, 16-32). En tal caso, el uso optimizado de registros es responsabilidad del compilador. Un programa escrito en un lenguaje de alto nivel no tiene, por supuesto, referencias explícitas a los registros. En su lugar, las cantidades son referidas simbólicamente en el programa. El objetivo del compilador es mantener en registros, en lugar

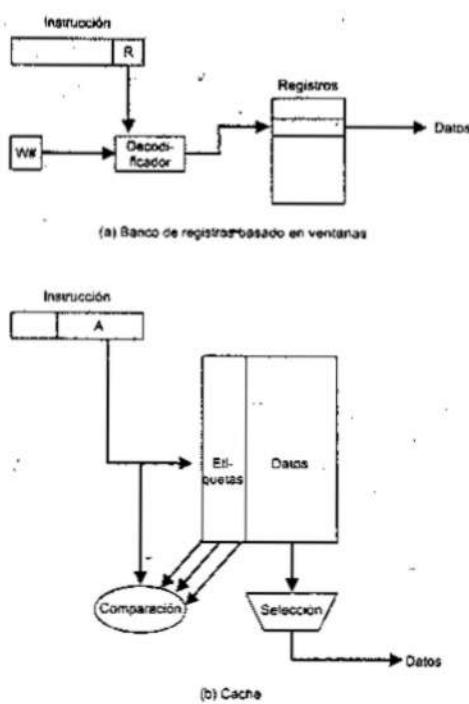


Figura 12.3. Referencia a un dato escalar.

de en memoria, los operandos necesarios para tantos cálculos como sea posible, y minimizar las operaciones de carga y almacenamiento.

Por lo general se sigue el siguiente enfoque. Cada cantidad del programa candidata para residir en un registro se asigna a un registro simbólico o virtual. El compilador entonces asigna el número ilimitado de registros simbólicos a un número fijo de registros reales. Los registros simbólicos cuya utilización no se solape pueden compartir el mismo registro real. Si en una parte concreta del programa hay más cantidades a tratar que registros reales, algunas de las cantidades se asignan a posiciones de memoria. Se usan las instrucciones de carga y almacenamiento para colocar temporalmente cantidades en registros en las operaciones de cálculo.

Lo esencial de la labor de optimización es decidir qué cantidades tienen que ser asignadas a registros en cualquier punto determinado del programa. La técnica usada más comúnmente en los compiladores para RISC se conoce como «coloreado de grafos», una técnica que procede de la disciplina de topología [CHAI82, CHOW86, COUT86, CHOW90].

El problema del coloreado de grafos es el siguiente. Dado un grafo que consta de nodos y arcos, asignar colores a los nodos de manera que nodos adyacentes tengan colores diferentes, y hacerlo de tal forma que se minimice el número de colores distintos. Este problema se adapta al problema de los compiladores de la siguiente forma. En primer lugar, el programa se analiza para construir un grafo de interferencias entre registros. Los nodos del grafo son los registros simbólicos. Si dos registros simbólicos están «vivos» durante el mismo fragmento

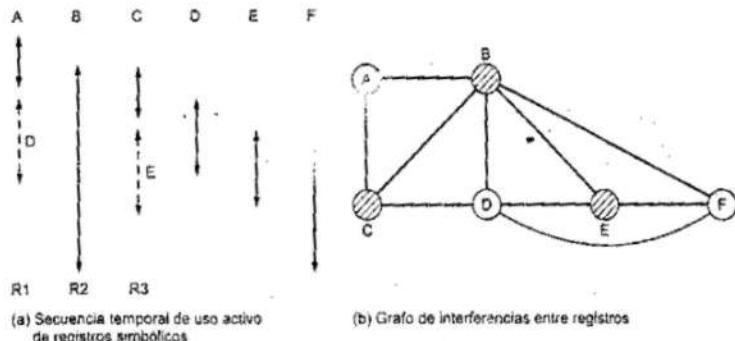


Figura 12.4. La técnica de coloreado de grafos.

to de programa, entonces se unen por un arco para representar su interferencia. Se intenta colorear el grafo con  $n$  colores, donde  $n$  es el número de registros. Si este proceso no tiene éxito completo, los nodos que no se pueden colorear se colocan en memoria, y se tienen que usar cargas y almacenamientos para crear espacio para las cantidades afectadas cuando éstas se necesiten.

La Figura 12.4 es un ejemplo sencillo de este proceso. Imaginemos un programa con seis registros simbólicos, que tiene que ser compilado para tres registros reales. La Figura 12.4a muestra la secuencia temporal de uso activo de cada registro simbólico, y la parte (b) muestra el grafo de interdependencias entre registros (se usan sombreados y tramas en lugar de colores). Hay que intentar un posible coloreado con tres colores. Un registro simbólico, el F, se queda sin colorear, y hay que gestionarlo con el uso de cargas y almacenamientos.

Generalmente, existe un compromiso entre el uso de un conjunto de registros grande y la optimización de registros basada en el compilador. Por ejemplo, [BRAD91a] presenta un estudio que modeló una arquitectura RISC con características similares al-Motorola 88000 y al MIPS R2000. Los investigadores variaron el número de registros de 16 a 128, y consideraron, tanto la utilización de todos los registros para uso general, como la separación de registros para enteros y coma flotante. Su estudio mostró que, incluso con una optimización de registros sencilla, se saca poco provecho a la utilización de más de 64 registros. Si se usan técnicas de optimización de registros razonablemente sofisticadas, con más de 32 registros sólo hay una mejora escasa de las prestaciones. Por último, observaron que con un pequeño número de registros (por ejemplo, 16), una máquina con una organización de registros compartidos funciona más rápido que una con una organización separada. Se pueden extraer conclusiones similares de [HUGU91], que expone un estudio centrado principalmente en la optimización del uso de un número pequeño de registros, en lugar de en la comparación del uso de grandes conjuntos de registros con esfuerzos en la optimización.

#### 12.4 ARQUITECTURA DE REPERTORIO REDUCIDO DE INSTRUCCIONES

En esta sección, consideraremos la motivación y algunas de las características generales de la arquitectura de repertorio reducido de instrucciones. Más adelante, en este capítulo, se estudiarán ejemplos específicos. Comenzamos con una discusión de las motivaciones de las arquitecturas contemporáneas de repertorio complejo de instrucciones.

## ¿POR QUÉ CISC?

Hemos mencionado la tendencia hacia repertorios de instrucciones más ricos, que incluyen un número mayor de instrucciones e instrucciones más complejas. Esta tendencia ha sido motivada por dos razones principales: el deseo de simplificar los compiladores, y el deseo de mejorar las prestaciones. Sirvió de base a estas razones el cambio de los programadores hacia los lenguajes de alto nivel (HLL), que hizo que los arquitectos intentaran diseñar máquinas que proporcionaran mejor soporte para los HLL.

No es la intención de este capítulo decir que los diseñadores de CISC tomaron la dirección equivocada. Verdaderamente, debido a que la tecnología continúa evolucionando, y a que existen arquitecturas a lo largo de todo un espectro en vez de en dos categorías puras, es poco probable que surja nunca una valoración del tipo blanco o negro. Por tanto, los comentarios que siguen sólo pretenden señalar algunos de los peligros potenciales de la aproximación CISC, y proporcionar cierta comprensión de la motivación de los partidarios de los RISC.

La primera de las razones citadas, la simplificación de los compiladores, parece obvia. La labor del escritor de compiladores es generar una secuencia de instrucciones máquina para cada sentencia de HLL. Si existen instrucciones máquina que se parezcan a sentencias del HLL, la tarea se simplifica. Este razonamiento ha sido puesto en duda por los investigadores de los RISC ([HENN82], [RAD183] y [PATT82b]). Estos han encontrado que las instrucciones máquina complejas son con frecuencia difíciles de aprovechar, ya que el compilador debe descubrir aquellos casos que se ajustan perfectamente a la construcción. La tarea de optimizar el código generado para minimizar su tamaño, reducir el número de instrucciones ejecutadas y mejorar la segmentación, es mucho más difícil con un repertorio complejo de instrucciones. Como prueba de esto, los estudios citados antes en este capítulo indican que la mayoría de las instrucciones de un programa compilado son comparativamente las más sencillas.

La otra razón importante mencionada es la esperanza de que un CISC produzca programas más pequeños y rápidos. Examinemos los dos aspectos de esta afirmación: que los programas sean más pequeños y que se ejecuten más rápido.

Los programas más pequeños tienen dos ventajas. En primer lugar, como el programa ocupa menos memoria, hay un ahorro de este recurso. Como la memoria hoy día es tan barata, la ventaja potencial ya no es primordial. Tiene mayor importancia el hecho de que programas más pequeños mejoren las prestaciones, lo que ocurre por dos motivos. El primero es que, el que haya menos instrucciones significa que hay que captar menos bytes de instrucciones. El segundo es que, en un entorno paginado los programas más pequeños ocupan menos páginas, reduciendo las faltas de página.

El problema de este razonamiento es que está lejos de ser cierto que un programa para un CISC sea más pequeño que el correspondiente programa para un RISC. En muchos casos, el programa para el CISC, expresado en lenguaje máquina simbólico, puede ser más corto (esto es, tener menos instrucciones), pero el número de bits de memoria que ocupa no tiene por qué ser más pequeño. La Tabla 12.6 muestra los resultados de tres estudios que compararon el tamaño de programas en C compilados en varias máquinas, incluyendo el RISC I, que tiene una arquitectura de repertorio reducido de instrucciones. Observe que hay poco o ningún ahorro cuando se usa un CISC en lugar de un RISC. Es también interesante advertir que el VAX, que tiene un repertorio de instrucciones mucho más complejo que el PDP-11, logra muy poco ahorro respecto a este último. Los investigadores de IBM confirmaron estos resultados [RAD183], constatando que el IBM 801 (un RISC) producía código cuyo tamaño era 0,9 veces el del código de un IBM S/370. El estudio utilizó un conjunto de programas en PL/I.

Tabla 12.6. Tamaño del código, relativo al RISC I

|            | [PATT82a]<br>11 programas en C | [KATE83]<br>12 programas en C | [HEAT84]<br>5 programas en C |
|------------|--------------------------------|-------------------------------|------------------------------|
| RISC I     | 1,0                            | 1,0                           | 1,0                          |
| VAX-11/780 | 0,8                            | 0,67                          |                              |
| M68000     | 0,9                            |                               | 0,9                          |
| Z8002      | 1,2                            |                               | 1,12                         |
| PDP-11/70  | 0,9                            | 0,71                          |                              |

Hay varias razones para estos resultados bastante sorprendentes. Hemos mencionado ya que los compiladores en los CISC tienden a elegir las instrucciones más sencillas, de manera que la concisión de las instrucciones complejas raramente entra en juego. También, debido a que hay más instrucciones en un CISC, son precisos códigos de operación más largos, produciendo instrucciones más largas. Por último, los RISC tienden a acentuar las referencias a registros en lugar de a memoria, y las primeras necesitan menos bits. Un ejemplo de este último efecto se discute dentro de poco.

De este modo, las expectativas de que un CISC presente programas más pequeños (con las ventajas que implicaría), pueden no llevarse a cabo. El segundo factor que motivaba repertorios de instrucciones cada vez más complejos era que la ejecución de instrucciones fuera más rápida. Parece tener sentido el que una operación compleja de un HLL se ejecute más rápido como una única instrucción máquina, que como una sucesión de instrucciones más primitivas. Sin embargo, debido a la propensión a usar las instrucciones más sencillas, esto puede no ser así. La unidad de control completa debe hacerse más compleja, y/o la memoria de control del microprograma ha de hacerse más grande, para proveer un repertorio de instrucciones más rico. Cualquiera de los dos factores aumenta el tiempo de ejecución de las instrucciones simples.

De hecho, algunos investigadores han encontrado que la aceleración en la ejecución de funciones complejas se debe, no tanto a la potencia de las instrucciones máquina complejas, como a su residencia en el rápido almacenamiento de control [RAD183]. En realidad, el almacenamiento de control actúa como una cache de instrucciones. De este modo, el arquitecto del hardware podría intentar determinar qué subrutinas o funciones se usarán con mayor frecuencia, y asignarlas al almacenamiento de control, implementándolas en microcódigo. La realidad no es muy alentadora. En los sistemas S/390, instrucciones tales como Translate (traducir) y Extended-Precision-Floating-Point-Divide (dividir en coma flotante con precisión ampliada) residen en almacenamiento de alta velocidad, mientras que la secuencia involucrada en establecer llamadas a procedimientos o en iniciar un gestor de interrupción se encuentra en la memoria principal, que es más lenta.

Por tanto, no está nada claro que la tendencia hacia repertorios de instrucciones de complejidad creciente sea apropiada. Ello ha llevado a varios grupos a seguir el camino opuesto.

#### CARACTERÍSTICAS DE LAS ARQUITECTURAS DE REPERTORIO REDUCIDO DE INSTRUCCIONES

Aunque existen diferentes aproximaciones a la arquitectura de repertorio reducido de instrucciones, hay ciertas características comunes a todas ellas:

- Una instrucción por ciclo.
- Operaciones registro a registro.

- Modos de direccionamiento sencillos.
- Formatos de instrucción sencillos.

Ahora haremos una breve discusión de estas características. Más adelante, en este capítulo, se examinarán ejemplos concretos.

La primera característica enumerada es que se ejecuta una instrucción máquina cada ciclo máquina. Un *ciclo máquina* se define como el tiempo que se tarda en captar dos operandos desde dos registros, realizar una operación de la ALU y almacenar el resultado en un registro. Así, las instrucciones máquina de un RISC no deberían ser más complicadas que las microinstrucciones de las máquinas CISC (tratadas en la parte cuatro), y deberían ejecutarse más o menos igual de rápido. Con instrucciones sencillas y de un ciclo, hay poca o ninguna necesidad de microcódigo; las instrucciones máquina pueden estar cableadas. Tales instrucciones deben ejecutarse más rápido que las instrucciones máquina comparables de otras máquinas, ya que no hay que acceder a la memoria de control de microprograma durante la ejecución de la instrucción.

Una segunda característica es que la mayoría de las operaciones deben ser del tipo *registro a registro*, con la excepción de sencillas operaciones LOAD y STORE para acceder a memoria. Esta forma de diseño simplifica el repertorio de instrucciones y, por tanto, la unidad de control. Por ejemplo, un repertorio de instrucciones RISC puede incluir sólo una o dos instrucciones ADD (por ejemplo, suma entera y suma con acarreo); el VAX tiene 25 instrucciones ADD diferentes. Otra ventaja es que tal arquitectura fomenta la optimización del uso de registros, ya que los operandos accedidos frecuentemente permanecen en el almacenamiento de alta velocidad.

Este énfasis en las operaciones registro a registro es único de los diseños RISC. Otras máquinas contemporáneas tienen tales instrucciones, pero incluyen también operaciones memoria a memoria y operaciones mixtas registro/memoria. En los años setenta, antes de la aparición de los RISC, se hicieron intentos de comparar estas aproximaciones. La Figura 12.5a muestra la aproximación tomada. Las arquitecturas hipotéticas se evaluaron según el tamaño del programa y el tráfico de memoria, expresado en número de bits. Resultados como éste llevaron a un investigador a sugerir que las arquitecturas futuras no deberían contener registros en absoluto [MYER78]. Uno se pregunta qué habría pensado, en su momento, de la máquina RISC comercializada por Pyramid, ¡con nada menos que 528 registros!

Lo que se pasó por alto en estos estudios fue un reconocimiento de que hay un acceso frecuente a un número pequeño de datos escalares locales, y de que, con un amplio banco de registros o un compilador optimizador, la mayoría de los operandos pueden permanecer en registros durante largos períodos de tiempo. Por eso, la Figura 12.5b puede ser una comparación más justa.

Una tercera característica es el uso de modos de direccionamiento sencillos. Casi todas las instrucciones RISC usan el sencillo direccionamiento a registro. Se pueden incluir varios modos adicionales, como el desplazamiento y el relativo al contador de programa. Otros modos más complejos se pueden sintetizar por software a partir de los simples. Nuevamente, esta característica de diseño simplifica el repertorio de instrucciones y la unidad de control.

Una última característica común es el uso de formatos de instrucción sencillos. Generalmente, sólo se usa un formato o unos pocos. La longitud de las instrucciones es fija, y alineada en los límites de una palabra. Las posiciones de los campos, especialmente la del código de operación, son fijas. Este tipo de diseño tiene varias ventajas. Con campos fijos, la decodificación del código de operación y el acceso a los operandos en registros pueden tener lugar simultáneamente. Los formatos sencillos simplifican la unidad de control. Se optimiza la captación de instrucciones, ya que se captan unidades de una palabra de longitud. El alinea-

| 8   | 16 | 16 | 16 |
|-----|----|----|----|
| Add | B  | C  | A  |

Memoria a memoria

$I = 56; D = 96; M = 152$

| 8             | 4 | 16 |
|---------------|---|----|
| Load[rB]      | B |    |
| Load[rC]      | B |    |
| Add[rA] rB rC |   |    |
| Store[rA]     |   | A  |

Registro a memoria

(a)  $A \leftarrow B + C$ 

| 8   | 16 | 16 | 16 |
|-----|----|----|----|
| Add | B  | C  | A  |
| Add | A  | C  | B  |
| Sub | B  | D  | D  |

Memoria a memoria

$I = 168; D = 288; M = 456$

| 8   | 4  | 4  | 4  |
|-----|----|----|----|
| Add | rA | rB | rC |
| Add | rB | rA | rC |
| Sub | rD | rD | rB |

Registro a memoria  
 $I = 104; D = 96; M = 200$ (b)  $A \leftarrow B + C; B \leftarrow A + C; D \leftarrow D - B$  $I = \text{Tamaño total de las instrucciones ejecutadas}$  $D = \text{Tamaño total de los datos en esas instrucciones}$  $M = I + D = \text{Tráfico total con memoria}$ 

Figura 12.5. Dos comparaciones de las aproximaciones registro a registro y memoria a memoria.

miento en el límite de una palabra significa también que una única instrucción no traspasa los límites de una página.

Estas características pueden evaluarse conjuntamente para determinar las ventajas potenciales de la aproximación RISC. Las ventajas se pueden separar en dos categorías: las relacionadas con las prestaciones y las relacionadas con la implementación VLSI.

Respecto a las prestaciones, se puede presentar cierta cantidad de «pruebas indicativas». En primer lugar, se pueden desarrollar compiladores optimizadores más eficaces. Con instrucciones más primitivas, hay más ocasiones de sacar funciones fuera-de-bucles, reorganizar código buscando una mayor eficiencia, maximizar la utilización de registros, etc. Es posible incluso calcular partes de instrucciones complejas en tiempo de compilación. Por ejemplo, la instrucción Move Characters (MVC) del S/390, transfiere una cadena de caracteres de una posición a otra. Cada vez que se ejecuta, la transferencia dependerá de la longitud de la cadena, de si las posiciones se solapan y en qué direcciones, y de cuáles son las características de alineación. En la mayoría de los casos, esto se conocerá en tiempo de compilación. Por tanto, el compilador podría producir una secuencia optimizada de instrucciones primitivas para esta función.

Un segundo punto, ya mencionado, es que la mayoría de las instrucciones generadas por un compilador son, de todos modos, relativamente sencillas. Parece razonable que una unidad de control, construida expresamente para estas instrucciones y que usara poco o ningún microcódigo, podría ejecutarlas más rápido que un CISC comparable.

Un tercer punto tiene que ver con el uso de segmentación de las instrucciones. Los investigadores de RISC creen que la técnica de segmentación de las instrucciones se puede aplicar mucho más eficazmente a un repertorio reducido de instrucciones. Más adelante, examinaremos este punto en detalle.

Un último punto, algo menos importante, es que los programas RISC deberían ser más sensibles a las interrupciones, ya que éstas se comprueban entre operaciones bastante elemen-

tales. Las arquitecturas con instrucciones complejas, o bien restringen las interrupciones a los límites de instrucción, o bien tienen que definir puntos interrumpibles específicos e implementar mecanismos para reanudar la ejecución de una instrucción.

No está demostrado que aumenten las prestaciones con una arquitectura de repertorio reducido de instrucciones. Se han hecho varios estudios, pero no con máquinas de tecnología y potencia comparables. Además, la mayoría de los estudios no han intentado separar los efectos de un repertorio reducido de instrucciones de los de un banco de registros extenso. La «prueba indicaria», sin embargo, es sugestiva.

La segunda área con una ventaja potencial es más evidente, y tiene que ver con la implementación VLSI. Cuando se usa VLSI, el diseño y la implementación del procesador son esencialmente distintos. Los procesadores tradicionales, tales como el IBM S/390 y el VAX, constan de una o más tarjetas de circuitos impresos, que contienen chips SSI y MSI estandarizados. Con la llegada de LSI y VLSI, es posible meter un procesador completo en un único chip. Con un procesador de un único chip hay dos motivaciones para seguir una estrategia RISC. En primer lugar, está la cuestión de las prestaciones. Los retardos dentro del chip son mucho menores que los retardos entre chips, de modo que tiene sentido dedicar el escaso espacio del chip a aquellas actividades que ocurren frecuentemente. Hemos visto que las instrucciones sencillas y el acceso a datos escalares locales son, de hecho, las actividades más frecuentes. Los chips RISC de Berkeley se diseñaron con esta consideración en mente. Mientras que un microprocesador de un único chip típico dedica aproximadamente la mitad de su área a la memoria de control del microcódigo, el RISC I dedica sólo el 6 % de su área a la unidad de control [SHER84].

Un segundo asunto relacionado con VLSI es el tiempo de diseño e implementación. Un procesador VLSI es difícil de desarrollar. En lugar de confiar en los componentes SSI y MSI disponibles, el diseñador debe realizar el diseño, trazado y modelado del circuito en el nivel de dispositivos. Con una arquitectura de repertorio reducido de instrucciones, este proceso es mucho más fácil, como lo demuestra la Tabla 12.7 [FITZ81]. Si, además, las prestaciones del chip RISC son equivalentes a los microprocesadores CISC comparables, las ventajas de la aproximación RISC se hacen evidentes.

### CARACTERÍSTICAS CISC FRENTE A RISC

Tras el entusiasmo inicial por las máquinas RISC, ha habido una creciente convicción de que (1) los diseños RISC pueden sacar provecho de la inclusión de algunas características CISC, y de que (2) los diseños CISC pueden sacar provecho de la inclusión de algunas características RISC. El resultado es que los diseños RISC más recientes, especialmente el PowerPC, no son ya RISC «puros», y que los diseños CISC más recientes, particularmente el Pentium II, incorporan ciertas características RISC.

Tabla 12.7. Esfuerzo de diseño y trazado de algunos microprocesadores

| CPU            | Transistores (miles) | Diseño (personas/mes) | Trazado (personas/mes) |
|----------------|----------------------|-----------------------|------------------------|
| RISC I         | 44                   | 15                    | 12                     |
| RISC II        | 41                   | 18                    | 12                     |
| M68000         | 68                   | 100                   | 70                     |
| Z8000          | 18                   | 60                    | 70                     |
| Intel iAPX-432 | 110                  | 170                   | 90                     |

Tabla 12.8. Características de algunos procesadores

| Procesador  | Número de tamaños de instrucción diferentes | Tamaño de instrucción máximo en bytes | Número de modos de direccionamiento | Direccionamiento indirecto | Carga/almacenamiento con cálculo aritmético asociado | Número máximo de operandos en memoria | Direccionamiento no es alineando permitido | Número máximo de usos de la MMU | Número de bits por especificador de registro entero | Número de bits por especificador de registro de coma flotante |
|-------------|---------------------------------------------|---------------------------------------|-------------------------------------|----------------------------|------------------------------------------------------|---------------------------------------|--------------------------------------------|---------------------------------|-----------------------------------------------------|---------------------------------------------------------------|
| AMD29000    | 1                                           | 4                                     | 1                                   | no                         | no                                                   | 1                                     | no                                         | 1                               | 8                                                   | 3*                                                            |
| MIPS R2000  | 1                                           | 4                                     | 1                                   | no                         | no                                                   | 1                                     | no                                         | 1                               | 5                                                   | 4                                                             |
| SPARC       | 1                                           | 4                                     | 2                                   | no                         | no                                                   | 1                                     | no                                         | 1                               | 5                                                   | 4                                                             |
| MC68000     | 1                                           | 4                                     | 3                                   | no                         | no                                                   | 1                                     | no                                         | 1                               | 5                                                   | 4                                                             |
| HP PA       | 1                                           | 4                                     | 10*                                 | no                         | no                                                   | 1                                     | no                                         | 1                               | 5                                                   | 4                                                             |
| IBM RT/PC   | 2*                                          | 4                                     | 1                                   | no                         | no                                                   | 1                                     | no                                         | 1                               | 4*                                                  | 3*                                                            |
| IBM RS/6000 | 1                                           | 4                                     | 4                                   | no                         | no                                                   | 1                                     | sí                                         | 1                               | 5                                                   | 5                                                             |
| Intel i860  | 1                                           | 4                                     | 4                                   | no                         | no                                                   | 1                                     | no                                         | 1                               | 5                                                   | 4                                                             |
| IBM 3090    | 4                                           | 8                                     | 2*                                  | no*                        | sí                                                   | 2                                     | sí                                         | 4                               | 4                                                   | 2                                                             |
| Intel 80486 | 12                                          | 12                                    | 15                                  | no*                        | sí                                                   | 2                                     | sí                                         | 4                               | 3                                                   | 3                                                             |
| NSC 32016   | 21                                          | 21                                    | 23                                  | sí                         | sí                                                   | 2                                     | sí                                         | 4                               | 3                                                   | 3                                                             |
| MC68040     | 11                                          | 22                                    | 44                                  | sí                         | sí                                                   | 2                                     | sí                                         | 8                               | 4                                                   | 3                                                             |
| VAX         | 56                                          | 58                                    | 22                                  | sí                         | sí                                                   | 6                                     | sí                                         | 24                              | 4                                                   | 0                                                             |
| Clipper     | 4*                                          | 8*                                    | 9*                                  | no                         | no                                                   | 1                                     | no                                         | 2                               | 4*                                                  | 3*                                                            |
| Intel 80960 | 2*                                          | 8*                                    | 9*                                  | no                         | no                                                   | 1                                     | sí*                                        | —                               | 5                                                   | 3*                                                            |

\* RISC que no se ajusta a esta característica

\* CISC que no se ajusta a esta característica

Una comparación interesante, en [MASH94], se adentra en este asunto. La Tabla 12.8 lista varios procesadores y los compara según varias características. Para el objetivo de esta comparación, se considera típico de un RISC lo siguiente:

1. Un único tamaño de instrucción.
2. Ese tamaño es típicamente 4 bytes.
3. Un pequeño número de modos de direccionamiento de datos, normalmente menor que 5. Este parámetro es difícil de precisar. En la tabla, los modos registro y literal no se han contado, y los diferentes formatos con diferentes tamaños de desplazamiento se han contado por separado.
4. No se usa direccionamiento indirecto que requiera efectuar un acceso a memoria para conseguir la dirección de memoria de otro operando.
5. No hay operaciones que combinen carga/almacenamiento con cálculos aritméticos (por ejemplo, suma desde memoria o suma a memoria).

6. No se direcciona más de un operando de memoria por instrucción.
7. Las operaciones de carga/almacenamiento no admiten un alineamiento de datos arbitrario.
8. Un número máximo de usos de la unidad de gestión de memoria («memory management unit», MMU) de una dirección de dato en cada instrucción.
9. El número de bits de un campo designador de registro entero, es de cinco o más. Esto quiere decir que, en un momento dado, se pueden referenciar explícitamente, por lo menos 32 registros enteros.
10. El número de bits de un campo designador de registro de coma flotante es de cuatro o más. Esto quiere decir que por lo menos 16 registros de coma flotante se pueden referenciar explícitamente en un momento dado.

Los puntos 1 a 3 indican la complejidad de la decodificación de instrucciones. Los puntos 4 a 8 indican la facilidad o dificultad de la segmentación, especialmente por la presencia de requisitos de memoria virtual. Los puntos 9 y 10 se relacionan con la habilidad de sacar partido de los compiladores.

En la tabla, los primeros ocho procesadores poseen claramente arquitectura RISC, los siguientes cinco son, sin duda, CISC, y los últimos dos procesadores son considerados a menudo como RISC que, en realidad, tienen características CISC.

## 12.5 SEGMENTACIÓN DE CAUCE EN RISC

### SEGMENTACIÓN CON INSTRUCCIONES REGULARES

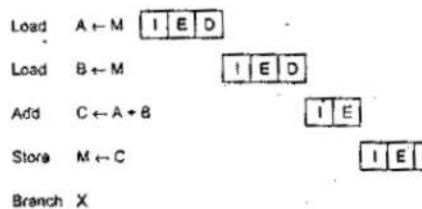
Como discutimos en la Sección 11.4, para aumentar las prestaciones se usa con frecuencia la segmentación de instrucciones. Reconsideraremos este asunto en el contexto de la arquitectura RISC. La mayoría de las instrucciones son del tipo registro a registro, y un ciclo de instrucción tiene las dos siguientes fases:

- I: Captación de instrucción.
- E: Ejecución. Realiza una operación de la ALU con registros como entrada y salida.

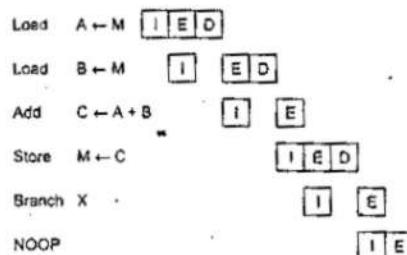
Las operaciones de carga y almacenamiento requieren tres fases:

- I: Captación de instrucción.
- E: Ejecución. Calcula una dirección de memoria.
- D: Memoria. Operación registro a memoria o memoria a registro.

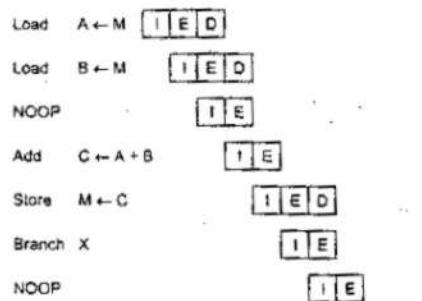
La Figura 12.6a representa la temporización de una secuencia de instrucciones que no usan segmentación. Sin duda, se trata de un proceso dispendioso. Incluso una segmentación muy simple puede mejorar las prestaciones sustancialmente. La Figura 12.6b muestra un esquema de segmentación de dos vías, en el cual las fases I y E de dos instrucciones diferentes se pueden ejecutar simultáneamente. Este esquema ofrece el doble de velocidad de ejecución que un esquema serie. Hay dos problemas que impiden que se consiga la máxima aceleración. El primero es que suponemos que se usa una memoria de un único puerto, y que sólo es posible un acceso a memoria en cada fase. Esto requiere la inserción de un estado de espera en algunas instrucciones. El segundo problema consiste en que una instrucción de bifurcación interrumpe al flujo secuencial de ejecución. Para adaptarse a estos problemas con la mínima circuitería, el compilador o ensamblador puede insertar una instrucción NOOP en el flujo de instrucciones.



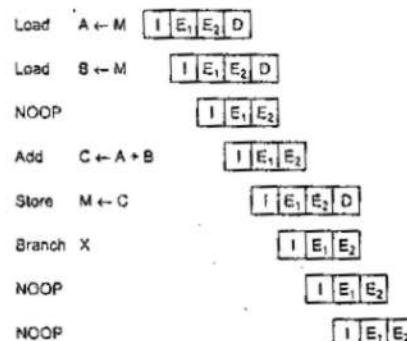
(a) Ejecución secuencial



(b) Temporización con un cauce de dos vías



(c) Temporización con un cauce de tres vías



(d) Temporización con un cauce de cuatro vías

Figura 12.6. Efectos de la segmentación.

La segmentación puede ser mejorada permitiendo dos accesos a memoria por fase. Esto produce la secuencia que se muestra en la Figura 12.6c. Ahora, hasta tres instrucciones pueden solaparse, y la mejora es, como mucho, un factor de tres. Nuevamente, las instrucciones de bifurcación hacen que la aceleración sea un poco menor que la máxima posible. Hay que advertir también, que las dependencias de datos tienen su efecto. Si una instrucción necesita un operando que es modificado por la instrucción precedente, se necesita un retardo. También esto puede ser llevado a cabo por un NOOP.

La segmentación discutida hasta aquí funciona mejor si las tres fases son aproximadamente de la misma duración. Como la fase E usualmente implica una operación de la ALU, puede ser más larga. En ese caso, podemos dividirla en dos subfases:

- E<sub>1</sub>: Lectura del banco de registros.
- E<sub>2</sub>: Operación de la ALU y escritura en el registro.

Dada la simplicidad y regularidad del repertorio de instrucciones, el diseño de la organización en tres o cuatro fases se realiza fácilmente. La Figura 12.6d muestra el resultado con

un cauce de cuatro vías. Hasta cuatro instrucciones pueden estar en curso en un momento dado, y la aceleración potencial máxima es un factor de cuatro. Observe que de nuevo hay que usar NOOP para los retardos por bifurcaciones y dependencias de datos.

## OPTIMIZACIÓN DE LA SEGMENTACIÓN

Dada la naturaleza sencilla y regular de las instrucciones RISC, los esquemas de segmentación se pueden emplear eficientemente. Hay poca variación en la duración de la ejecución de instrucciones, y el cauce puede adaptarse para reflejar este hecho. Sin embargo, hemos visto que las dependencias de datos y bifurcaciones reducen la velocidad de ejecución global.

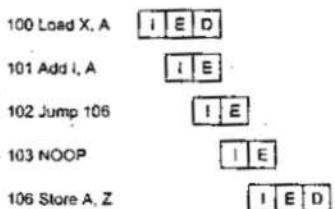
Para compensar estas dependencias se han desarrollado técnicas de reorganización del código. Consideremos primero las instrucciones de salto. El *salto retardado*, que es una forma de incrementar la eficacia de la segmentación, utiliza un salto que no tiene lugar hasta después de que se ejecute la siguiente instrucción (de ahí que se llame *retardado*). Este curioso procedimiento se ilustra en la Tabla 12.9. En la columna «salto normal», vemos un programa normal en lenguaje máquina con instrucciones simbólicas. Después de que se ejecute la instrucción de la dirección 102, la siguiente instrucción a ejecutar es la de la dirección 105. Para regularizar el cauce, se inserta un NOOP después de este salto. No obstante, se pueden incrementar las prestaciones si se intercambian las instrucciones 101 y 102. La Figura 12.7 muestra el resultado. La instrucción JUMP se capta antes de la instrucción ADD. Hay que observar, sin embargo, que la instrucción ADD se capta antes de que la ejecución de la instrucción JUMP tenga la oportunidad de modificar el contador de programa. Por consiguiente, se conserva la semántica original del programa.

Este intercambio de instrucciones funcionará con éxito con saltos incondicionales, llamadas y retornos. Para bifurcaciones condicionales, el procedimiento no se puede aplicar a ciegas. Si la condición comprobada por la bifurcación puede alterarse por la instrucción inmediatamente precedente, el compilador ha de abstenerse de hacer el intercambio y, en su lugar, debe insertar un NOOP. En caso contrario, el compilador puede intentar insertar una instrucción útil después del salto. La experiencia con los sistemas RISC de Berkeley e IBM 801 es que la mayoría de instrucciones de bifurcación condicional pueden optimizarse de esta forma ([PATT82a], [RAD83]).

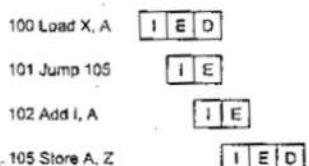
Un tipo de táctica similar, llamada «carga retardada», se puede usar con las instrucciones LOAD. En las instrucciones LOAD, el procesador bloquea el registro destino de la carga. Después, el procesador continúa la ejecución del flujo de instrucciones hasta que alcanza una instrucción que necesite ese registro, deteniéndose hasta que la carga finalice. Si el compilador puede reorganizar las instrucciones de manera que se pueda hacer un trabajo útil mientras la carga está en el cauce, la eficiencia aumentará.

Tabla 12.9. Salto normal y retardado

| Dirección | Salto normal |     | Salto retardado |     | Salto retardado optimizado |     |
|-----------|--------------|-----|-----------------|-----|----------------------------|-----|
| 100       | LOAD         | X,A | LOAD            | X,A | LOAD                       | X,A |
| 101       | ADD          | 1,A | ADD             | 1,A | JUMP                       | 105 |
| 102       | JUMP         | 105 | JUMP            | 106 | ADD                        | 1,A |
| 103       | ADD          | A,B | NOOP            |     | ADD                        | A,B |
| 104       | SUB          | C,B | ADD             | A,B | SUB                        | C,B |
| 105       | STORE        | A,Z | SUB             | C,B | STORE                      | A,Z |
| 106       |              |     | STORE           | A,Z |                            |     |



(a) Inserción de NOOP



(b) Inversión del orden de las instrucciones

Figura 12.7. Uso del salto retardado.

Como nota final, hay que señalar que el diseño del cauce de instrucciones no debe ser llevado a cabo independientemente de otras técnicas de optimización aplicadas al sistema. Por ejemplo, [BRAD91b] muestra que la planificación de instrucciones para el cauce y la asignación dinámica de registros tienen que considerarse conjuntamente para lograr la mayor eficiencia.

### 12.6. MIPS R4000

Uno de los primeros conjuntos de chips RISC disponibles comercialmente, fue desarrollado por MIPS Technology Inc. El sistema se inspiró en un sistema experimental, que también usaba el nombre MIPS, desarrollado en Stanford [HENN84]. En esta sección estudiamos el MIPS R4000. Tiene sustancialmente la misma arquitectura y repertorio de instrucciones de los anteriores diseños MIPS: R2000, R3000, y R6000. La diferencia más significativa es que el R4000 usa 64 en lugar de 32 bits para los buses de datos internos y externos y para las direcciones, los registros, y la ALU.

El uso de 64 bits tiene diversas ventajas sobre una arquitectura de 32 bits. Permite un espacio de direccionamiento más grande (suficientemente grande, para que un sistema operativo coloque más de un terabyte de ficheros directamente en la memoria virtual para acceder fácilmente). Siendo corrientes en este momento discos de 1 gigabyte y mayores, el espacio de direcciones de 4 gigabytes de una máquina de 32 bits se vuelve limitado. También, la capacidad de 64 bits permite al R4000 procesar datos, tales como números en coma flotante IEEE de simple precisión y cadenas de hasta 8 caracteres, de una sola vez.

El chip procesador R4000 está dividido en dos secciones: una contiene la CPU, y la otra

contiene un coprocesador de gestión de memoria. El procesador tiene una arquitectura muy sencilla. El propósito fue diseñar un sistema, en el cual la lógica de ejecución de instrucciones fuera lo más sencilla posible, dejando espacio disponible para la lógica que aumentara las prestaciones (por ejemplo, la unidad de gestión de memoria completa).

El procesador contiene 32 registros de 64 bits. También está provisto de hasta 128 kbytes de cache de alta velocidad, la mitad para instrucciones y la mitad para datos. Esta cache relativamente grande (el IBM 3090 está provisto de 128 a 256 kbytes de cache) permite que el sistema mantenga grandes conjuntos de código de programa y datos locales al procesador, descargando el bus de memoria principal, y evitando la necesidad de un banco de registros grande con la lógica de ventanas asociada.

## REPERTORIO DE INSTRUCCIONES

La Tabla 12.10 lista el repertorio de instrucciones básico de todos los procesadores de la serie MIPS R. La Tabla 12.11 lista las instrucciones adicionales implementadas en el R4000. Todas las instrucciones del procesador se codifican en un único formato de palabra de 32 bits. Todas las operaciones con datos son del tipo registro a registro: las únicas referencias a memoria son operaciones puras de carga/almacenamiento.

El R4000 no utiliza códigos de condición. Si una instrucción genera una condición, los indicadores correspondientes se almacenan en un registro de uso general. Esto elimina la necesidad de una lógica especial para tratar con códigos de condición, ya que éstos afectan al mecanismo de segmentación y a la reordenación de instrucciones por el compilador. En lugar de eso, se emplea el mecanismo ya implementado, que se ocupa de las dependencias de valores de registros. Además, las condiciones asignadas en el banco de registros son propensas a las mismas optimizaciones del compilador, en cuanto a asignación y reutilización, que cualquier otro valor almacenado en un registro.

Como la mayoría de las máquinas de tipo RISC, el MIPS usa una longitud de instrucción fija de 32 bits. Esta única longitud de instrucción simplifica la captación y decodificación de instrucciones, y también simplifica la interacción entre la captación de instrucciones y la unidad de gestión de memoria virtual (esto es, las instrucciones no atraviesan los límites de una palabra o una página). Los tres formatos de instrucción (Figura 12.8) comparten un formato común para los códigos de operación y las referencias a registros, lo que simplifica la decodificación de instrucciones. La acción de instrucciones más complejas se puede sintetizar en tiempo de compilación.

Sólo se implementa en hardware el modo de direccionamiento a memoria que se usa con más frecuencia. Todas las referencias a memoria consisten en un desplazamiento de 16 bits y un registro de 32 bits. Por ejemplo, la instrucción «cargar palabra» (load word) es de la forma siguiente:

*lw r2, 128(r3)*      carga la palabra de la dirección indicada en el registro 3 más un desplazamiento de 128 en el registro 2

Cualquiera de los 32 registros de uso general puede usarse como registro base. El registro r0 siempre contiene 0.

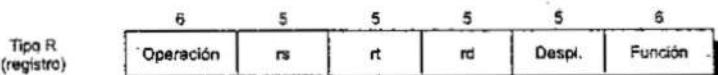
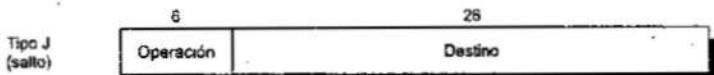
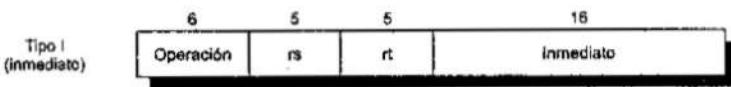
El compilador usa múltiples instrucciones máquina para sintetizar los modos de direccionamiento de las máquinas convencionales. Se dan algunos ejemplos en la Tabla 12.12 [CHOW87]. La tabla muestra el uso de la instrucción lui (load upper immediate, «cargar superior inmediato»). La instrucción carga la mitad superior de un registro con un valor inmediato de 16 bits, poniendo la mitad inferior a cero.

Tabla 12.10. Repertorio de instrucciones de la serie R de MIPS

| OP                                                            | Descripción                                     | OP                                              | Descripción                                  |
|---------------------------------------------------------------|-------------------------------------------------|-------------------------------------------------|----------------------------------------------|
| <b>Instrucciones de carga/almacenamiento</b>                  |                                                 | <b>Instrucciones de multiplicación/división</b> |                                              |
| LB                                                            | Cargar byte                                     | MULT                                            | Multiplicar                                  |
| LBU                                                           | Cargar byte sin signo                           | MULTU                                           | Multiplicar signo                            |
| LH                                                            | Cargar media palabra                            | DIV                                             | Dividir                                      |
| LHU                                                           | Cargar media palabra sin signo                  | DIVU                                            | Dividir sin signo                            |
| LW                                                            | Cargar palabra                                  | MFHI                                            | Transferir desde parte alta                  |
| LWL                                                           | Cargar palabra a la izquierda                   | MTHI                                            | Transferir a parte alta                      |
| LWR                                                           | Cargar palabra a la derecha                     | MFLO                                            | Transferir desde parte baja                  |
| SB                                                            | Almacenar byte                                  | MTLO                                            | Transferir a parte baja                      |
| SH                                                            | Almacenar media palabra                         |                                                 |                                              |
| SW                                                            | Almacenar palabra                               |                                                 |                                              |
| SWL                                                           | Almacenar palabra a la izquierda                |                                                 |                                              |
| SWR                                                           | Almacenar palabra a la derecha                  |                                                 |                                              |
| <b>Instrucciones aritméticas (inmediatas con la ALU)</b>      |                                                 | <b>Instrucciones de salto y bifurcación</b>     |                                              |
| ADDI                                                          | Sumar inmediato                                 | J                                               | Salir                                        |
| ADDIU                                                         | Sumar inmediato sin signo                       | JAL                                             | Salir y enlazar                              |
| SLTI                                                          | Poner a uno si menor inmediato                  | JR                                              | Salir a registro                             |
| SLTIU                                                         | Poner a uno si menor inmediato sin signo        | JALR                                            | Salir y enlazar registro                     |
|                                                               |                                                 | BEQ                                             | Bifurcar sin igual                           |
| ANDI                                                          | Y inmediato                                     | BNE                                             | Bifurcar si distinto                         |
| ORI                                                           | O inmediato                                     | BLEZ                                            | Bifurcar si menor o igual que cero           |
| XORI                                                          | O exclusivo inmediato                           | BGTZ                                            | Bifurcar si mayor que 0                      |
| LUI                                                           | Cargar superior inmediato                       | BLTZ                                            | Bifurcar si menor que cero                   |
|                                                               |                                                 | BGEZ                                            | Bifurcar si mayor o igual que cero           |
|                                                               |                                                 | BLTZAL                                          | Bifurcar si menor que cero y enlazar         |
|                                                               |                                                 | BGEZAL                                          | Bifurcar si mayor o igual que cero y enlazar |
| <b>Instrucciones aritméticas (3 operandos, tipo registro)</b> |                                                 | <b>Instrucciones del coprocesador</b>           |                                              |
| ADD                                                           | Sumar                                           | LWCz                                            | Cargar palabra en el coprocesador            |
| ADDU                                                          | Sumar sin signo                                 | SWCz                                            | Almacenar palabra en el coprocesador         |
| SUB                                                           | Restar                                          | MTCz                                            | Transferir al coprocesador                   |
| SUBU                                                          | Restar sin signo                                | MFCz                                            | Transferir desde el coprocesador             |
| SLT                                                           | Poner a uno si menor                            | CTCz                                            | Transferir control al coprocesador           |
| SLTU                                                          | Poner a uno si menor sin signo                  | CFCz                                            | Transferir control desde el coprocesador     |
| AND                                                           | Y                                               | COPz                                            | Operación del coprocesador                   |
| OR                                                            | O                                               | BCzT                                            | Bifurcar si coprocesador z es cierto         |
| XOR                                                           | O exclusiva                                     | BCzF                                            | Bifurcar si coprocesador z es falso          |
| NOR                                                           | No O                                            |                                                 |                                              |
| <b>Instrucciones de desplazamiento</b>                        |                                                 | <b>Instrucciones especiales</b>                 |                                              |
| SLL                                                           | Desplazamiento lógico a la izquierda            | SYSCALL                                         | Llamada al sistema                           |
| SRL                                                           | Desplazamiento lógico a la derecha              | BREAK                                           | Ruptura                                      |
| SRA                                                           | Desplazamiento aritmético a la derecha          |                                                 |                                              |
| SLLV                                                          | Desplazamiento lógico a la izquierda variable   |                                                 |                                              |
| SRLV                                                          | Desplazamiento lógico a la derecha variable     |                                                 |                                              |
| SRAV                                                          | Desplazamiento aritmético a la derecha variable |                                                 |                                              |

Tabla 12.11. Instrucciones adicionales del R4000

| OP                                           | Descripción                                                | OP    | Descripción                                      |
|----------------------------------------------|------------------------------------------------------------|-------|--------------------------------------------------|
| <b>Instrucciones de carga/almacenamiento</b> |                                                            |       | <b>Instrucciones de excepción</b>                |
| LL                                           | Carga enlazada                                             | TGE   | Interceptar si mayor o igual                     |
| SC                                           | Almacenamiento condicional                                 | TGEU  | Interceptar si mayor o igual sin signo           |
| SYNC                                         | Sincronización                                             | TLT   | Interceptar si menor                             |
|                                              |                                                            | TLTU  | Interceptar si menor sin signo                   |
|                                              |                                                            | TEQ   | Interceptar si igual                             |
|                                              |                                                            | TNE   | Interceptar si distinto                          |
|                                              |                                                            | TGEI  | Interceptar si mayor o igual inmediato           |
|                                              |                                                            | TGEIU | Interceptar si mayor o igual sin signo inmediato |
|                                              |                                                            | TLTI  | Interceptar si menor inmediato                   |
|                                              |                                                            | TLTIU | Interceptar si menor sin signo inmediato         |
|                                              |                                                            | TEQI  | Interceptar si igual inmediato                   |
|                                              |                                                            | TNEI  | Interceptar si distinto inmediato                |
| <b>Instrucciones de salto y bifurcación</b>  |                                                            |       | <b>Instrucciones del coprocesador</b>            |
| BEQL                                         | Bifurcar si igual probablemente                            | LDCz  | Cargar doble en el coprocesador                  |
| BNEL                                         | Bifurcar si distinto probablemente                         | SDCz  | Almacenar doble en el coprocesador               |
| BLEZL                                        | Bifurcar si menor o igual que cero probablemente           |       |                                                  |
| BGTZL                                        | Bifurcar si mayor que cero probablemente                   |       |                                                  |
| BLTZL                                        | Bifurcar si menor que cero probablemente                   |       |                                                  |
| BGEZL                                        | Bifurcar si mayor o igual que cero probablemente           |       |                                                  |
| BLTZALL                                      | Bifurcar si mayor o igual que cero y enlazar probablemente |       |                                                  |
| BCzTL                                        | Bifurcar si coprocesador z cierto probablemente            |       |                                                  |
| CDzFL                                        | Bifurcar si coprocesador z falso probablemente             |       |                                                  |



- Operación Código de operación  
 rs Designador de registro fuente  
 rt Designador de registro fuente/destino  
 Inmediato Dato inmediato, salto, o desplazamiento  
 Destino Dirección destino de salto  
 rd Designador de registro destino  
 Despl. Número de desplazamientos  
 Función Designador de función de la ALU/desplazador

Figura 12.8. Formatos de instrucción del MIPS.

**Tabla 12.12.** Síntesis de otros modos de direccionamiento a partir de los modos de direccionamiento del MIPS

| Instrucción aparente            | Instrucción real                                                                                  |
|---------------------------------|---------------------------------------------------------------------------------------------------|
| lw r2, <despl. de 16 bits>      | lw r2, <despl. de 16 bits> (r0)                                                                   |
| lw r2, <despl. de 32 bits>      | lui r1, <16 bits altos del despl. ><br>lw r2, <16 bits bajos del despl. > (r1)                    |
| lw r2, <despl. de 32 bits> (r4) | lui r1, <16 bits altos del despl. ><br>addu r1, r1, r4<br>lw r2, <16 bits bajos del despl. > (r1) |

### CAUCE DE INSTRUCCIONES

Gracias a su arquitectura de instrucciones simplificada, el MIPS puede lograr una segmentación muy eficiente. Es instructivo estudiar la evolución del cauce del MIPS, ya que ilustra la evolución de la segmentación en los RISC en general.

Los sistemas RISC experimentales iniciales, y la primera generación de los procesadores RISC comerciales, lograban velocidades de ejecución que se aproximaban a una instrucción por ciclo de reloj del sistema. Para mejorar estas prestaciones, han evolucionado dos clases de procesadores, que ejecutan múltiples instrucciones por ciclo de reloj: las arquitecturas superescalar y supersegmentada. Esencialmente, una arquitectura superescalar reproduce exactamente cada etapa del cauce de manera que dos o más instrucciones en la misma etapa del cauce se puedan procesar simultáneamente. Una arquitectura supersegmentada es una que utiliza más etapas, y de grano más fino, en el cauce. Con más etapas puede haber más instrucciones en el cauce al mismo tiempo, aumentando el paralelismo.

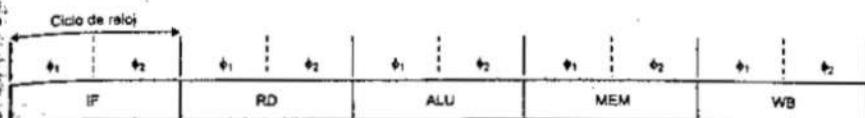
Las dos alternativas tienen limitaciones. En la segmentación superescalar, la dependencia entre instrucciones de cauces diferentes pueden reducir la velocidad del sistema. Además, se necesita lógica adicional para coordinar estas dependencias. En la supersegmentación, hay ciertos gastos extra asociados con la transferencia de instrucciones de una etapa a la siguiente.

El Capítulo 13 se dedica al estudio de la arquitectura superescalar. El MIPS R4000 es un buen ejemplo de arquitectura RISC supersegmentada.

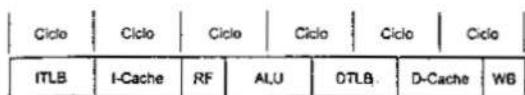
La Figura 12.9a muestra los cauces de instrucciones del R3000. En el R3000, el cauce avanza una vez por ciclo de reloj. El compilador del MIPS es capaz de reordenar las instrucciones para rellenar con código los retardos insertados del 70 al 90 % de las veces. Todas las instrucciones siguen la misma secuencia de cinco etapas del cauce:

- Captación de instrucción
- Captación de operando fuente del banco de registros
- Operación en la ALU o generación de dirección de operando
- Referencia a dato en memoria
- Escritura en el banco de registros

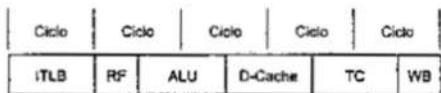
Como ilustra la Figura 12.9a, no sólo hay paralelismo debido a la segmentación, sino también paralelismo en la ejecución de una única instrucción. El ciclo de reloj de 60 ns se divide en dos fases de 30 ns. Las operaciones externas de acceso a instrucciones y datos de la



(a) Cauce detallado del R3000



(b) Cauce modificado del R3000 con duraciones reducidas



(c) Cauce optimizado del R3000 con accesos en paralelo al TBL y a la cache

|         |                                                 |
|---------|-------------------------------------------------|
| IF      | = Captación de instrucción                      |
| RD      | = Lectura                                       |
| MEM     | = Acceso a memoria                              |
| WB      | = Escritura                                     |
| I-Cache | = Acceso a la cache de instrucciones            |
| RF      | = Captación de operando de registro             |
| D-Cache | = Acceso a la cache de datos                    |
| ITLB    | = Traducción de dirección de instrucción        |
| IDEC    | = Decodificación de instrucción                 |
| IA      | = Cálculo de dirección de instrucción           |
| DA      | = Cálculo de dirección virtual de un dato       |
| DTLB    | = Traducción de dirección de un dato            |
| TC      | = Comprobación de etiqueta de la cache de datos |

Figura 12.9. Mejoras del cauce del R3000.

cache requieren 60 ns cada una, igual que las principales operaciones internas (OP, DA, IA). La decodificación de instrucciones es una operación más sencilla, y requiere sólo una fase de 30 ns, solapada con la captación de registros de la misma instrucción. El cálculo de una dirección en una instrucción de bifurcación también se superpone a la decodificación de la instrucción y a la captación de registros, de modo que una bifurcación en la instrucción  $i$  pueda direccionar el acceso I-Cache de la instrucción  $i + 2$ . De un modo parecido, una carga en la instrucción  $i$  capta datos usados inmediatamente por la fase OP de la instrucción  $i + 1$ , mientras que un resultado de la ALU o de un desplazamiento se pasa directamente a la instrucción  $i + 1$  sin retardo alguno. Este estrecho acoplamiento entre las instrucciones conduce a un cauce muy eficiente.

Detalladamente, por tanto, cada ciclo de reloj se divide en fases separadas, que llamamos  $\phi_1$  y  $\phi_2$ . Las funciones realizadas en cada fase se resumen en la Tabla 12.13.

El R4000 incorpora varios avances técnicos sobre el R3000. La utilización de una tecnología más avanzada posibilita que se reduzca a la mitad (30 ns) el período de reloj, y que el tiempo de acceso al banco de registros también se reduzca a la mitad. Además, la mayor densidad del circuito permite que las caches de instrucciones y de datos se incorporen en el propio chip. Antes de examinar el cauce final del R4000, consideraremos como se puede modificar el cauce del R3000 para mejorar las prestaciones, usando la tecnología del R4000.

Tabla 12.13. Etapas del cauce del R3000

| Etapa del cauce | Fase    | Función                                                                                                                                                            |
|-----------------|---------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| IF              | φ1      | Usando el TLB, traducir la dirección virtual de una instrucción a dirección física (después de una decisión de bifurcación).                                       |
| IF              | φ2      | Enviar la dirección física a la cache de instrucciones.                                                                                                            |
| RD              | φ1      | Devolver una instrucción desde la cache de instrucciones.                                                                                                          |
| RD              | φ2      | Decodificar la instrucción.<br>Leer del banco de registros.                                                                                                        |
| ALU             | φ1 + φ2 | Si hay una bifurcación, calcular la dirección destino del salto.<br>Si se trata de una operación registro a registro, se ejecuta la operación aritmética o lógica. |
| ALU             | φ1      | Si se trata de una bifurcación, decidir si se produce o no el salto.                                                                                               |
| ALU             | φ2      | Si se trata de una referencia a memoria (carga o almacenamiento), calcular la dirección virtual del dato.                                                          |
| ALU             | φ1      | Si se trata de una referencia a memoria, traducir la dirección virtual del dato a dirección física, usando el TLB.                                                 |
| MEM             | φ1      | Si se trata de una referencia a memoria, enviar la dirección física a la cache de datos.                                                                           |
| MEM             | φ2      | Si se trata de una referencia a memoria, devolver el dato desde la cache de datos, y comprobar las etiquetas.                                                      |
| WB              | φ1      | Escribir en el banco de registros.                                                                                                                                 |

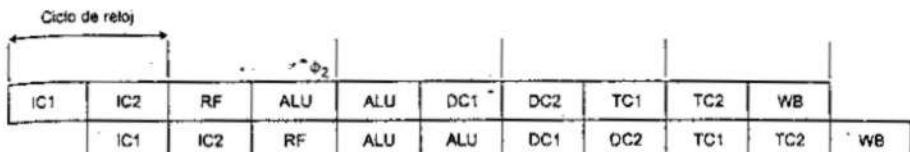
La Figura 12.9b muestra un primer paso. Recuerde que los ciclos en esta figura duran la mitad que los de la Figura 12.9a. Debido a que están en el mismo chip, las etapas de acceso a las caches de instrucciones y de datos suponen sólo la mitad de tiempo, de tal modo que aún duran un único ciclo de reloj. Además, debido al aumento de velocidad en el acceso al banco de registros, la lectura y escritura de un registro sigue durando sólo la mitad de un ciclo de reloj.

Como las caches del R4000 están integradas en el chip, la traducción de dirección virtual a física puede retrasar el acceso a cache. Este retardo se reduce implementando caches indexadas virtualmente, yiendo hacia una paralelización del acceso a la cache y de la traducción de la dirección. La Figura 12.9c muestra el cauce del R3000 optimizado con esta mejora. A causa de la compresión de eventos, la comprobación de etiquetas de la cache de datos se realiza por separado en el ciclo posterior al de acceso a la cache.

En un sistema supersegmentado, el hardware existente se usa varias veces por ciclo, insertando registros de separación para dividir cada etapa del cauce. Básicamente, cada etapa del cauce funciona en un múltiplo de la frecuencia de reloj base, dependiendo el múltiplo del grado de supersegmentación. La tecnología del R4000 tiene una velocidad y una densidad que permite una supersegmentación de grado 2. La Figura 12.10a muestra el cauce del R3000 optimizado usando esta supersegmentación. Observe que se trata esencialmente de la misma estructura dinámica de la Figura 12.9c.

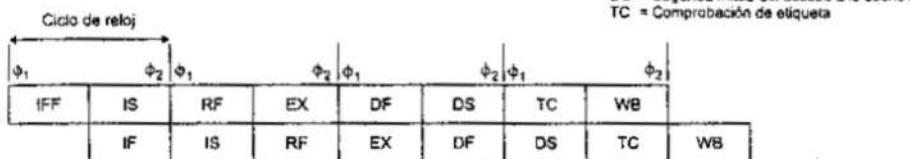
Se pueden llevar a cabo más mejoras. Para el R4000, se diseñó un sumador mucho más grande y especializado. Esto posibilita la ejecución de operaciones de la ALU a doble velocidad. Otras mejoras permiten doblar la velocidad de ejecución de cargas y almacenamientos. El cauce resultante se muestra en la Figura 12.10b.

El R4000 tiene ocho etapas en el cauce, lo que quiere decir que puede haber hasta ocho instrucciones en el cauce al mismo tiempo. El cauce avanza a un ritmo de dos etapas por ciclo de reloj. Las ocho etapas del cauce son:



(a) Implementación supersegmentada del cauce optimizado del R3000

IFF = Primera mitad de la captación de instrucción  
 IS = Segunda mitad de la captación de instrucción  
 RF = Captación de operandos de registros  
 EX = Ejecución de instrucción  
 IC = Cache de instrucciones  
 DC = Cache de datos  
 DF = Primera mitad del acceso a la cache de datos  
 DS = Segunda mitad del acceso a la cache de datos  
 TC = Comprobación de etiqueta



(b) Cauce del R4000

Figura 12.10. Cauce supersegmentado teórico del R3000 y real en el R4000.

- **Primera mitad de la captación de instrucción:** La dirección virtual se da a la cache de instrucciones y al buffer de traducción anticipada.
- **Segunda mitad de la captación de instrucción:** Se obtiene la instrucción de la cache de instrucciones y el TLB genera la dirección física.
- **Banco de registros:** Ocurren tres actividades en paralelo:
  - La instrucción se decodifica, y se comprueban condiciones de interbloqueo (es decir si esta instrucción depende del resultado de la instrucción precedente).
  - Se comprueba la etiqueta en la cache de instrucciones.
  - Se captan los operandos del banco de registros.
- **Ejecución de instrucción:** Puede ocurrir una de estas tres actividades:
  - Si la instrucción es una operación registro a registro, la ALU lleva a cabo la operación aritmética o lógica.
  - Si la instrucción es una carga o un almacenamiento, se calcula la dirección virtual del dato.
  - Si la instrucción es una bifurcación, la dirección virtual del destino del salto se calcula, y se comprueban las condiciones de bifurcación.
- **Primera mitad del acceso a la cache de datos:** La dirección virtual se da a la cache de datos y al TLB.
- **Segunda mitad del acceso a la cache de datos:** Se obtiene el dato de la cache de datos y el TLB genera la dirección física.
- **Comprobación de etiquetas:** Se comprueban las etiquetas en la cache en el caso de cambios y almacenamientos.
- **Escritura:** El resultado de la instrucción se escribe en el banco de registros.

**12.7. SPARC**

Las siglas SPARC (Scalable Processor Architecture, «arquitectura de procesador escalable») hacen referencia a una arquitectura definida por Sun Microsystems. Sun desarrolló su propia implementación de SPARC, pero también autoriza a otros vendedores a fabricar máquinas compatibles con SPARC. La arquitectura SPARC se inspira en la máquina RISC I de Berkeley, y su repertorio de instrucciones y organización de registros están basados exactamente en el modelo RISC de Berkeley.

**CONJUNTO DE REGISTROS DEL SPARC**

Como el RISC de Berkeley, el SPARC utiliza ventanas de registros. Cada ventana consta de 24 registros, y el número total de ventanas depende de la implementación, y varía de 2 a 32 ventanas. La Figura 12.11 ilustra una implementación que admite 8 ventanas, con un total de 136 registros físicos; como indica la discusión de la Sección 12.1, éste parece un número razonable de ventanas. Los registros físicos del 0 al 7 son registros globales compartidos por todos los procedimientos. Cada proceso ve los registros lógicos del 0 al 31. Los registros lógicos del 24 al 3, a los que se denomina *entradas*, son compartidos con el procedimiento que hace la llamada (padre); y los registros lógicos del 8 al 15, llamados *salidas*, son compartidos con el procedimiento llamado (hijo). Estas dos partes se solapan con otras ventanas. Los registros lógicos del 16 al 23, denominados *locales*, no se comparten ni se superponen con otras ventanas. Nuevamente, como indica la discusión de la Sección 4.1, la disponibilidad de 8 registros para pasar parámetros, debería ser suficiente en la mayoría de los casos (véase por ejemplo, la Tabla 12.4).

La Figura 12.12 es otra representación de la superposición de registros. El procedimiento que llama, coloca los parámetros a pasar en los registros *salidas*; el procedimiento llamado trata estos mismos registros físicos como sus registros *entradas*. El procesador guarda un puntero de ventana en curso (*current-window pointer*, CWP), en el registro de estado del procesador (*processor status register*, PSR), que apunta a la ventana del procedimiento en ejecución. La máscara de ventana no válida (*window invalid mask*, WIM), también en el PSR, indica qué ventana no es válida.

Con la arquitectura de registros del SPARC, normalmente no es necesario guardar y restaurar ningún registro en una llamada a un procedimiento. El compilador se simplifica, porque sólo ha de preocuparse de la asignación eficiente de los registros locales de un procedimiento, y no de la asignación de registros entre procedimientos.

**REPERTORIO DE INSTRUCCIONES**

La Tabla 12.14 lista las instrucciones de la arquitectura SPARC. La mayoría de las instrucciones referían solamente operandos en registros. Las instrucciones registro a registro tienen tres operandos, y se pueden expresar de la forma siguiente:

$$R_d \leftarrow R_{s1} \text{ op } S2$$

$R_d$  y  $R_{s1}$  son referencias a registros;  $S2$  puede hacer referencia a un registro o a un operando inmediato de 13 bits. El registro cero ( $R_0$ ) está cableado al valor 0. Este formato se adapta bien a los programas típicos, que tienen una alta proporción de datos escalares locales y constantes.

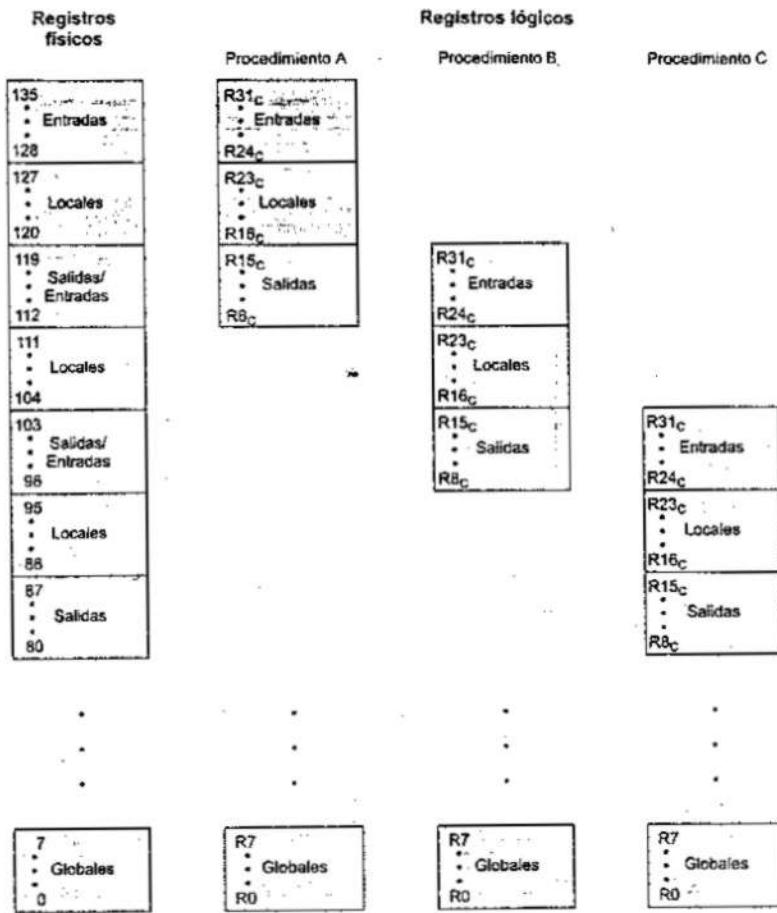


Figura 12.11. Disposición de las ventanas de registros del SPARC con tres procedimientos.

Las operaciones disponibles en la ALU se pueden agrupar como sigue:

- Suma entera (con o sin acarreo).
- Resta entera (con o sin adeudo).
- Operaciones booleanas bit a bit: Y, O, O exclusiva y sus negaciones.
- Desplazamientos lógico a la izquierda, lógico a la derecha, y aritmético a la derecha.

Todas estas instrucciones, excepto los desplazamientos, pueden ajustar opcionalmente los cuatro códigos de condición (CERO, NEGATIVO, DESBORDAMIENTO, ACARREO). Los enteros se representan con 32 bits en complemento a dos.

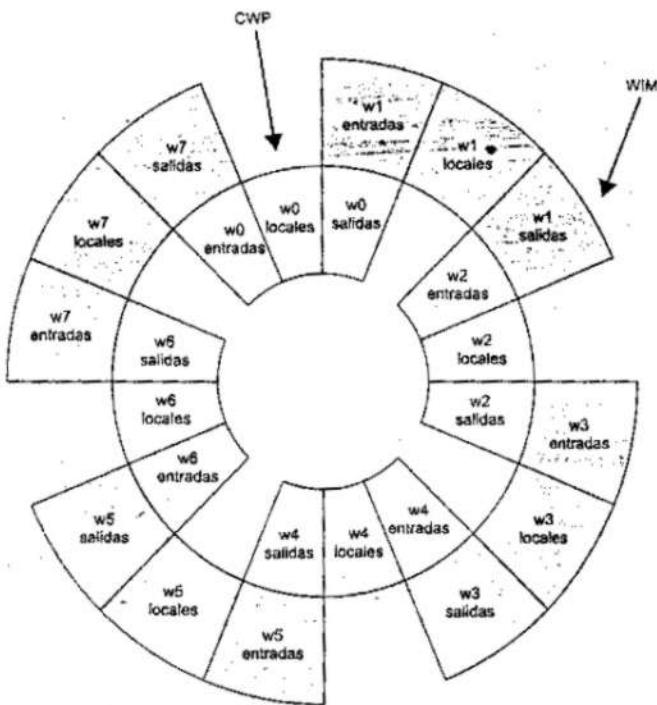


Figura 12.12. Ocho ventanas de registros que forman una pila circular en el SPARC.

Únicamente las sencillas instrucciones de carga y almacenamiento referencian la memoria. Hay instrucciones de carga y almacenamiento separadas para palabras (32 bits), dobles palabras, medias palabras y bytes. Para los dos últimos casos, hay instrucciones que cargan estas cantidades como números con signo o sin signo. En el caso de números con signo, se extiende el signo para llenar el registro destino de 32 bits. En el caso de números sin signo, éste se rellena con ceros.

El único modo de direccionamiento disponible, aparte del modo registro, es el modo de desplazamiento. Esto es, la dirección efectiva de un operando consiste en una dirección contenida en un registro más un desplazamiento:

$$EA = (R_{s1}) + S2$$

$$\circ EA = (R_{s1}) + (R_{s2})$$

dependiendo de si el segundo operando es un dato inmediato o una referencia a registro. Para realizar una carga o un almacenamiento, se añade una fase extra al ciclo de instrucción. Durante la segunda fase, la dirección de memoria se calcula usando la ALU; la carga o almacenamiento tiene lugar en la tercera fase. Este modo de direccionamiento sencillo es bastante versátil, y puede usarse para sintetizar otros modos de direccionamiento, como se indica en la Tabla 12.15.

Tabla 12.14. Repertorio de instrucciones del SPARC

| OP                                           | Descripción                            | OP                                        | Descripción                                  |
|----------------------------------------------|----------------------------------------|-------------------------------------------|----------------------------------------------|
| <b>Instrucciones de carga/almacenamiento</b> |                                        | <b>Instrucciones aritméticas</b>          |                                              |
| LDSB                                         | Cargar byte con signo                  | ADD                                       | Sumar                                        |
| LDSH                                         | Cargar media palabra con signo         | ADDC                                      | Sumar, ajustar icc                           |
| LDUB                                         | Cargar byte sin signo                  | ADDX                                      | Sumar con acarreo                            |
| LDUH                                         | Cargar media palabra sin signo         | ADDXCC                                    | Sumar con acarreo, ajustar icc               |
| LD                                           | Cargar palabra                         | SUB                                       | Restar                                       |
| LDI                                          | Cargar doble palabra                   | SUBCC                                     | Restar, ajustar icc                          |
| STB                                          | Almacenar byte                         | SUBX                                      | Restar con adeudo                            |
| STH                                          | Almacenar media palabra                | SUBXCC                                    | Restar con adeudo, ajustar icc               |
| STD                                          | Almacenar palabra                      | MULSCC                                    | Paso de multiplicación, ajustar icc          |
| STDI                                         | Almacenar doble palabra                | <b>Instrucciones de salto/bifurcación</b> |                                              |
| <b>Instrucciones de desplazamiento</b>       |                                        | BCC                                       | Bifurcar si condición                        |
| SLL                                          | Desplazamiento lógico a la izquierda   | FBCC                                      | Bifurcar si condición de coma flotante       |
| SRL                                          | Desplazamiento lógico a la derecha     | CBCC                                      | Bifurcar si condición del procesador         |
| SRA                                          | Desplazamiento aritmético a la derecha | CALL                                      | Llamar a procedimiento                       |
| <b>Instrucciones booleanas</b>               |                                        | JMP                                       | Saltar y enlazar                             |
| AND                                          | Y                                      | TCC                                       | Interceptar si condición                     |
| ANDCC                                        | AND, ajustar icc                       | SAVE                                      | Avanzar ventana de registros                 |
| ANDN                                         | NO Y                                   | RESTORE                                   | Mover ventanas hacia atrás                   |
| ANDNCC                                       | NO Y, ajustar icc                      | RETT                                      | Retornar de interceptación                   |
| OR                                           | O                                      | <b>Instrucciones diversas</b>             |                                              |
| ORCC                                         | O, ajustar icc                         | SETHI                                     | Fijar los 22 bits altos                      |
| ORN                                          | NO O                                   | UNIMP                                     | Instrucción no implementada (interceptación) |
| ORNCC                                        | NO O, ajustar icc                      | RD                                        | Ler un registro especial                     |
| XOR                                          | O exclusiva                            | WR                                        | Escribir en un registro especial             |
| XORCC                                        | O exclusiva, ajustar icc               | IFLUSH                                    | Vaciar la cache de instrucciones             |
| XNOR                                         | NO O exclusiva                         |                                           |                                              |
| XNORCC                                       | NO O exclusiva, ajustar icc            |                                           |                                              |

Tabla 12.15. Síntesis de otros modos de direccionamiento a partir de los del SPARC

| Modo                           | Algoritmo    | Equivalente del SPARC             | Tipo de instrucción   |
|--------------------------------|--------------|-----------------------------------|-----------------------|
| Inmediato                      | operando = A | S2                                | Registro a registro   |
| Directo                        | EA = A       | R <sub>g</sub> + S2               | Carga, almacenamiento |
| Registro                       | EA = R       | R <sub>g1</sub> , R <sub>g2</sub> | Registro a registro   |
| Indirecto a través de registro | EA = (R)     | R <sub>g1</sub> + 0 <sup>2</sup>  | Carga, almacenamiento |
| Desplazamiento                 | EA = (R) + A | R <sub>g1</sub> + S2              | Carga, almacenamiento |

Es instructivo comparar la capacidad de direccionamiento del SPARC con la del MIPS. El MIPS utiliza un desplazamiento de 16 bits, mientras que el SPARC utiliza 13 bits. Por otra parte, el MIPS no permite que se construya una dirección a partir de los contenidos de dos registros.

### FORMATO DE INSTRUCCIÓN

Como el MIPS R4000, el SPARC emplea un conjunto sencillo de formatos de instrucción de 32 bits (Figura 12.13). Todas las instrucciones comienzan con un código de operación de 2 bits. En ciertas instrucciones, este código se amplía con bits de código de operación adiciona-

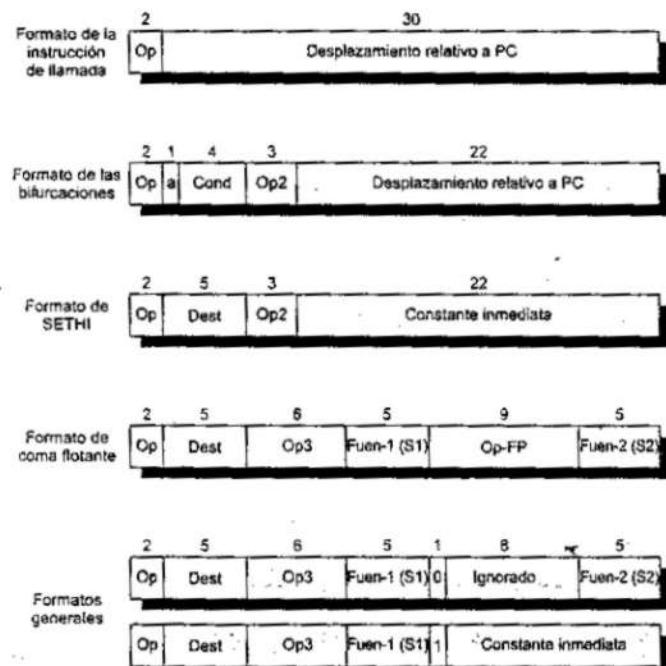


Figura 12.13. Formatos de instrucción del SPARC.

les en otra parte del formato. En la instrucción de llamada, un operando inmediato de 30 bits se amplía con dos bits a cero a la derecha, para formar una dirección de 32 bits relativa a PC en complemento a dos. Las instrucciones se alinean en límites de 32 bits, por lo que basta esta forma de direccionamiento.

La instrucción de bifurcación incluye un campo de condición de 4 bits, que corresponden a los bits de códigos de condición normales, de modo que puede comprobarse cualquier combinación de condiciones. A la dirección de 22 bits relativa a PC se añaden dos bits a cero por la derecha para formar una dirección relativa de 24 bits en complemento a dos. Cuando el bit de anulación no está a uno, la instrucción después de la de bifurcación, siempre se ejecuta, sin tener en cuenta si se produce el salto. Ésta es la típica operación de salto retardado que se encuentra en muchas máquinas RISC y que se describe en la Sección 12.5 (véase Figura 12.7). Sin embargo, cuando el bit de anulación está a uno, sólo se ejecuta la instrucción siguiente a la de bifurcación, si se produce el salto. El procesador puede suprimir el efecto de esta instrucción, aunque ya esté en el cauce. Este bit de anulación es útil, porque hace más fácil al compilador la tarea de llenar el ciclo de retardo que sigue a una bifurcación condicional. La instrucción destino del salto siempre se puede poner en el ciclo de retardo, ya que si no se produce el salto, esa instrucción puede anularse. El motivo de que esta técnica sea conveniente es que en las bifurcaciones condicionales generalmente se produce el salto más de la mitad de las veces.

La instrucción SETHI es una instrucción especial que se usa para cargar o almacenar un valor de 32 bits. Es necesaria para cargar y almacenar constantes grandes y direcciones. La instrucción SETHI asigna los 22 bits de su operando inmediato a los 22 bits de orden superior de un registro, y llena con ceros los 10 bits de orden inferior. En uno de los formatos generales puede especificarse una constante de hasta 13 bits, y esa instrucción puede usarse para llenar los restantes 10 bits del registro. Una instrucción de carga o almacenamiento, también puede usarse para conseguir un modo de direccionamiento directo. Para cargar un valor de la posición K de memoria, podríamos usar las siguientes instrucciones del SPARC:

|                        |                                                                          |
|------------------------|--------------------------------------------------------------------------|
| sethi %hi(K), %r8      | : cargar los 22 bits más altos de la dirección<br>de K en el registro r8 |
| ld [%r8 + %lo(K)], %r8 | : cargar el contenido de la posición K en r8                             |

Las macros %hi y %lo se usan para definir operandos inmediatos que consistan en los bits de dirección adecuados de una posición de memoria. Este uso de SETHI es similar al de la instrucción LUI del MIPS (Tabla 12.12).

El formato de coma flotante se usa para operaciones en coma flotante. Se designan dos registros fuente y uno destino.

Por último, todas las demás operaciones que incluyen cargas, almacenamientos, operaciones aritméticas y lógicas, usan uno de los dos últimos formatos que se muestran en la Figura 12.13. Uno de los formatos utiliza dos registros fuente y uno destino, mientras que el otro utiliza un registro fuente, un operando inmediato de 13 bits y un registro destino.

## 12.8. LA CONTROVERSIAS ENTRE RISC Y CISC

Durante muchos años, la tendencia general en la arquitectura y organización de computadores ha sido incrementar la complejidad del procesador: más instrucciones, más modos de direccionamiento, más registros especializados, etc. El movimiento RISC representa una ruptura fundamental con la filosofía que hay detrás de esa tendencia. Naturalmente, la aparición

de los sistemas RISC y la publicación de artículos por sus defensores ensalzando las virtudes de los RISC, ha llevado a una reacción contra lo que podría llamarse la corriente principal de la arquitectura de computadores.

El trabajo que se ha hecho para evaluar las ventajas de la aproximación RISC se puede agrupar en dos categorías:

- **Cuantitativa:** Intentos de comparar el tamaño de los programas y su velocidad de ejecución, en máquinas RISC y CISC de similar tecnología.
- **Cualitativa:** Revisión de asuntos tales como soporte de lenguajes de alto nivel y uso óptimo de los recursos VLSI.

La mayoría del trabajo de la evaluación cuantitativa lo han hecho aquellos que trabajan en sistemas RISC [PATT82b, HEAT84, PATT84], y ha sido, por lo general, favorable a la aproximación RISC. Hay otros que han examinado este asunto y no han acabado de convencerse [COLW85a, FLYN87, DAVI87]. Surgen varios problemas cuando se intentan realizar comparaciones [SERL86]:

- No hay una pareja de máquinas RISC y CISC que sean comparables en cuanto a coste del ciclo de vida, nivel de tecnología, complejidad a nivel de puertas, sofisticación del compilador, soporte para el sistema operativo, etc.
- No existe un conjunto de programas de prueba definitivo. Las prestaciones varían según el programa.
- Es difícil separar los efectos del hardware de los efectos debidos a la habilidad en el diseño del compilador.
- La mayor parte de los análisis comparativos con RISC se han hecho con máquinas «de juguete», en vez de con productos comerciales. Además, la mayoría de las máquinas comerciales anuncianadas como RISC poseen una mezcla de características RISC y CISC. Por tanto, es difícil una comparación equitativa con una máquina CISC comercial de «juego limpio» (por ejemplo, VAX, Pentium).

La valoración cualitativa es, casi por definición, subjetiva. Varios investigadores han fijado su atención en tal valoración [COLW85a, WALL85], pero los resultados son, en el mejor de los casos, ambiguos, claramente susceptibles de refutación [PATT85b] y, por supuesto, de contrarrefutación [COLW85b].

En los años más recientes, la controversia RISC frente a CISC se ha sosegado en gran parte. Esto se debe a que ha habido una convergencia progresiva de las tecnologías. Conforme la densidad de integración y la velocidad bruta del hardware han aumentado, los sistemas RISC se han vuelto más complejos. Al mismo tiempo, en un esfuerzo por exprimir las prestaciones al máximo, los diseños CISC se han concentrado en cuestiones asociadas tradicionalmente a los RISC, tales como un mayor número de registros de uso general y un creciente énfasis en el diseño del cauce de instrucciones.

#### 12.9. LECTURAS RECOMENDADAS

Algunos libros de texto con una buena cobertura de los conceptos RISC son [WARD90], [PATT98], y [HENN96].

[KANE92] trata en detalle la máquina comercial MIPS. [MIRA92] proporciona una buena visión general del MIPS R4000. [BASH91] discute la evolución desde la segmentación del R3000 hasta la supersegmentación del R4000. [DEWA90] cubre el SPARC con algo de detalle.

- BASH91 Bassteen, A.; Lui, I.; y Mullan, J. «A Superpipeline Approach to the MIPS Architecture.» *Proceedings. COMPCON Spring '91*, February, 1991.
- DEWA90 Dewar, R., y Smosna, M. *Microprocessors: A Programmer's View*. New York: McGraw-Hill, 1990.
- HENN96 Hennessy, J., y Patterson, D. *Computer Architecture: A Quantitative Approach*. San Mateo, CA: Morgan Kaufmann, 1996.
- KANE92 Kane, G., y Heinrich, J. *MIPS RISC Architecture*. Englewood Cliffs, NJ: Prentice-Hall, 1992.
- MIRA92 Mirapuri, S.; Woodacre, M.; y Vasseghi, N. «The MIPS R4000 Processor.» *IEEE Micro*, April, 1992.
- PATT98 Patterson, D., y Hennessy, J. *Computer Organization and Design: The Hardware/Software Interface*. San Mateo, CA: Morgan Kaufmann, 1998.
- WARD90 Ward, S., y Halstead, R. *Computation Structures*. Cambridge, MA: MIT Press, 1990.

## TÉTOS: PROBLEMAS

- 12.1. Considere el patrón de llamada/retorno de la Figura 4.29. ¿Cuántos desbordamientos y desbordamientos hacia cero (cada uno de los cuales causa una salvaguarda/restauración de registros) ocurrirán con un tamaño de ventana de
- 5?
  - 8?
  - 16?
- 12.2. En la discusión de la Figura 12.2 se explicó que sólo las dos primeras partes de una ventana se guardaban o restauraban. ¿Por qué no es necesario guardar los registros temporales?
- 12.3. Queremos determinar el tiempo de ejecución de un programa dado, usando los diversos esquemas de segmentación estudiados en la Sección 12.5. Sean:
- $N$  = Número de instrucciones ejecutadas.
- $D$  = Número de accesos a memoria.
- $J$  = Número de instrucciones de salto.
- Para el sencillo esquema secuencial (Figura 12.6a), el tiempo de ejecución es  $2N + D$  fases. Obtenga las fórmulas del tiempo de ejecución para la segmentación en dos vías, tres vías y cuatro vías.
- 12.4. Considere el siguiente fragmento de código en un lenguaje de alto nivel:

```
for I in 1 . . . 100 loop
 S ← S + Q(I).VAL
end loop;
```

Suponga que  $Q$  es una matriz de registros de 32 bytes, y que el campo  $VAL$  está en los primeros 4 bytes de cada registro. Usando código del 80x86, podemos compilar este fragmento de programa como sigue:

```

 MOV ECX, 1 ;usar el registro ECX para contener I
LP: IMUL EAX, ECX, 32 ;poner el desplazamiento en EAX
 MOV EBX, Q[EAX] ;cargar el campo VAL
 ADO S, EBX ;sumar a S
 INC ECX ;incrementar I
 CMP ECX, 100 ;comparar I con 100
 JBE LP ;seguir en el bucle hasta que I>100

```

Este programa utiliza la instrucción IMUL, que multiplica el segundo operando por el valor inmediato del tercer operando y lleva el resultado al primer operando (véase Problema 10.13). A un defensor de los RISC le gustaría demostrar que un compilador ingenioso puede eliminar instrucciones complejas innecesarias, tales como IMUL. Dé una demostración escribiendo de nuevo el programa para el 80x86 anterior sin usar la instrucción IMUL.

**12.5.** Considere el siguiente bucle:

```

S := 0
for K := 1 to 100 do
 S := S - 1;

```

Una traducción sencilla de este bucle a un lenguaje ensamblador genérico se parecería a esto:

```

LD R1, 0 ;almacenar el valor de S en R1
LD R2, 1 ;almacenar el valor de K en R2
LP: SUB R1, R1, R2 ;S := S - 1
 BEQ R2, 100, EXIT ;salir del bucle si K = 100
 ADD R2, R2, 1 ;si no, incrementar K
 JMP LP ;volver al principio del bucle

```

Un compilador de una máquina RISC introducirá ciclos de retardo en este código para que el procesador pueda emplear el mecanismo de salto retardado. La instrucción JMP es fácil de tratar, porque siempre va seguida de la instrucción SUB; por tanto, podemos simplemente poner una copia de la instrucción SUB en el ciclo de retardo tras la instrucción JMP. La instrucción BEQ plantea una dificultad. No podemos dejar el código como está, porque la instrucción ADD sería ejecutada una vez más de la cuenta. Por consiguiente, hace falta una instrucción NOP. Muestre el código resultante.

**12.6.** Añada filas referentes a los siguientes procesadores en la Tabla 12.8:

- Pentium II.
- PowerPC.

**12.7.** En muchos casos, las instrucciones máquina usuales que no se incluyen como parte del repertorio de instrucciones del MIPS se pueden sintetizar con una única instrucción del MIPS. Muéstrela para las siguientes:

- Transferencia entre dos registros.
- Incrementar, decrementar.
- Complementar.

- d) Negar.  
e) Poner a cero.
- 12.8. Una implementación del SPARC tiene  $K$  ventanas de registros. ¿Cuál es el número de registros físicos  $N$ ?
- 12.9. El SPARC carece de varias instrucciones que se encuentran generalmente en las máquinas CISC. Algunas de ellas se simulan fácilmente usando el registro R0, que siempre vale 0, o un operando constante. Estas instrucciones simuladas se llaman «pseudoinstrucciones» y el compilador del SPARC las reconoce. Muestre cómo simular las pseudoinstrucciones siguientes con una única instrucción del SPARC. En todas ellas, «fuen» y «dest» se refieren a registros. (Pista: Un almacenamiento en R0 no tiene efecto.)
- |                         |             |             |
|-------------------------|-------------|-------------|
| a) MOV fuen, dest       | d) NOT dest | g) DEC dest |
| b) COMPARE fuen1, fuen2 | e) NEG dest | h) CLR dest |
| c) TEST fuen1           | f) INC dest | i) NOP      |
- 12.10. Considere el fragmento de código siguiente:

```
if K > 10
 L := K + 1
else
 L := K - 1;
```

Una traducción directa de esta sentencia a ensamblador de SPARC podría tener la siguiente forma:

```

sethi %hi(K), %r8 ; cargar los 22 bits altos de la
 ; dirección de K en el registro r8
ld [%r8 + %lo(K)], %r8 ; cargar contenido de la posición K en r8
cmp %r8, 10 ; comparar contenido de r8 con 10
ble L1 ; bifurcar si (r8) ≤ 10
nop
sethi %hi(K), %r9 ; cargar los 22 bits altos de la
 ; dirección de K en el registro r9
ld [%r9 + %lo(K)], %r9 ; cargar contenido de la posición K en r9
inc %r9 ; sumar 1 a (r9)
sethi %hi(L), %r10
st %r9, [%r10 + %lo(L)] ; almacenar (r9) en la posición L
b L2
nop
L1: sethi %hi(%r11), %r11
ld [%r11 + %lo(K)], %r12 ; cargar contenido de la posición K en r12
dec %r12 ; restar uno a (r12)
sethi %hi(L), %r13
st %r12, [%r13 + %lo(L)] ; almacenar (r12) en la posición L
L2:
```

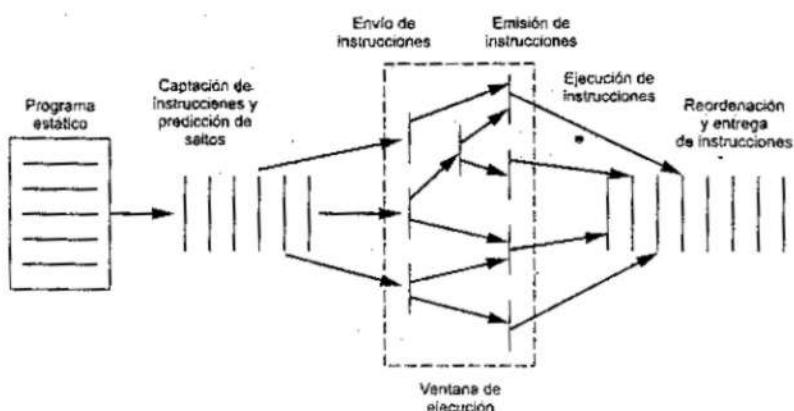
El código contiene un nop después de cada instrucción de salto para permitir la operación de salto retardado.

- a) Las optimizaciones de un compilador normal, que no tienen nada que hacer las máquinas RISC, son en general efectivas para poder realizar dos transformaciones en el código precedente. Observe que dos de las cargas no son necesari y que los dos almacenamientos se pueden unir si el almacenamiento se mueve otra posición dentro del código. Muestre el programa después de hacer estos d cambios.
- b) Es posible ahora realizar algunas optimizaciones propias del SPARC. El n después del ble puede reemplazarse moviendo otra instrucción a ese ciclo de r tardo, y activando el bit de anulación en la instrucción ble (expresado como ble L1). Muestre el programa tras este cambio.
- c) Ahora hay dos instrucciones innecesariás. Elimínelas y muestre el programa r sultante.

## CAPÍTULO 13

# Paralelismo a nivel de instrucciones, y procesadores superescalares

- 13.1. Visión de conjunto
- 13.2. Cuestiones relacionadas con el diseño
- 13.3. Pentium II
- 13.4. PowerPC
- 13.5. MIPS R10000
- 13.6. UltraSPARC-II
- 13.7. IA-64/MERCED
- 13.8. Lecturas y sitios Web recomendados
- 13.9. Problemas



- Un procesador superescalalar es aquél que usa múltiples cauces de instrucciones independientes. Cada cauce consta de múltiples etapas, de modo que puede tratar varias instrucciones a la vez. El hecho de que haya varios cauces introduce un nuevo nivel de paralelismo, permitiendo que varios flujos de instrucciones se procesen simultáneamente. Un procesador superescalalar saca provecho de lo que se conoce como «paralelismo a nivel de instrucciones», que hace referencia al grado en que las instrucciones de un programa pueden ejecutarse en paralelo.
- Típicamente, un procesador superescalalar capta varias instrucciones a la vez y, a continuación, intenta encontrar instrucciones cercanas que sean independientes entre sí y puedan, por consiguiente, ejecutarse en paralelo. Si la entrada de una instrucción depende de la salida de una instrucción precedente, la segunda instrucción no puede completar su ejecución al mismo tiempo ni antes que la primera. Una vez que se han identificado tales dependencias, el procesador puede emitir y completar instrucciones en un orden diferente al del código máquina original.
- El procesador puede eliminar algunas dependencias innecesarias mediante el uso de registros adicionales y el renombramiento de las referencias a registros en el código original.
- Mientras que los procesadores RISC puros emplean con frecuencia saltos retardados para maximizar la utilización del cauce de instrucciones, este método es menos apropiado para las máquinas superescalares. En lugar de eso, la mayoría de las máquinas superescalares emplean métodos tradicionales de predicción de saltos para aumentar su rendimiento.

**U**na implementación superescalar de la arquitectura de un procesador es aquella en la que las instrucciones comunes (aritmética entera y de coma flotante, cargas, almacenamientos y bifurcaciones condicionales) pueden iniciar su ejecución simultáneamente y ejecutarse de manera independiente. Estas implementaciones plantean complejos problemas de diseño relacionados con el cauce de instrucciones.

El diseño superescalar aparece en escena muy cerca de la arquitectura RISC. Aunque la arquitectura de repertorio de instrucciones simplificado de una máquina RISC se preste fácilmente a utilizar técnicas superescalares, la aproximación superescalar se puede usar tanto en una arquitectura RISC como en una CISC.

Mientras el período de gestación desde el comienzo de la auténtica investigación en RISC, con el IBM 801 y el RISC-I de Berkeley, hasta la llegada de máquinas RISC comerciales fue de siete u ocho años, las primeras máquinas superescalares estuvieron disponibles comercialmente tan sólo un año o dos después de que se acuñara el término *superescalar*. La aproximación superescalar se ha convertido en el método usual para implementar microprocesadores de altas prestaciones.

En este capítulo, comenzamos con una visión de conjunto de la aproximación superescalar, contrastándola con la supersegmentación. Después, se presentan las cuestiones de diseño relacionadas con la implementación superescalar. Más adelante, estudiamos varios ejemplos importantes de arquitecturas superescalares. Por último, examinamos la nueva arquitectura IA-64, que puede considerarse un diseño superescalar mejorado.

### 13.1. VISIÓN DE CONJUNTO

El término *superescalar*, acuñado en 1987 [AGER87], hace referencia a una máquina diseñada para mejorar la velocidad de ejecución de las instrucciones escalares. El nombre contrasta el propósito de este esfuerzo frente a los procesadores vectoriales, que se estudian en el Capítulo 16. En la mayoría de las aplicaciones, la mayor parte de las operaciones se realizan con cantidades escalares. Así pues, la aproximación superescalar representa el siguiente paso en la evolución de los procesadores de uso general de altas prestaciones.

Lo esencial del enfoque superescalar es su habilidad para ejecutar instrucciones de manera independiente en diferentes cauces. El concepto puede llevarse más lejos, permitiendo que las instrucciones se ejecuten en un orden diferente al del programa. La Figura 13.1 muestra, en términos generales, el planteamiento superescalar. Hay múltiples unidades funcionales, cada una de las cuales está implementada como un cauce segmentado, que admiten la ejecución en paralelo de varias instrucciones. En este ejemplo, dos operaciones enteras, dos de coma flotante y una de memoria (carga o almacenamiento), pueden estar ejecutándose al mismo tiempo.

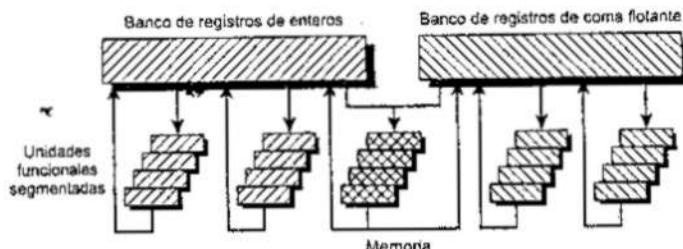


Figura 13.1. Organización superescalar usual [COME95].

**Tabla 13.1.** Incrementos de velocidad referenciados para máquinas del tipo superescalar

| Referencia | Incremento de velocidad |
|------------|-------------------------|
| [TJAD70]   | 1,8                     |
| [KUCK72]   | 8                       |
| [WEIS84]   | 1,58                    |
| [ACOS86]   | 2,7                     |
| [SOHI90]   | 1,8                     |
| [SMIT89]   | 2,3                     |
| [JOUP88b]  | 2,2                     |
| [LEE91]    | 7                       |

Muchos investigadores han estudiado procesadores de tipo superescalar, y su investigación indica que es posible cierto grado de mejora de las prestaciones. La Tabla 13.1 presenta las mejoras en velocidad reseñadas. Las diferencias en los resultados se deben, tanto a las diferencias en el hardware de las máquinas simuladas, como a las de las aplicaciones que se probaron.

### SUPERESCALAR FRENTA A SUPERSEGMENTADO

Una solución alternativa para alcanzar mayores prestaciones es la llamada supersegmentación, un término acuñado en 1988 [JOUP88]. La supersegmentación aprovecha el hecho de que muchas etapas del cauce realizan tareas que requieren menos de la mitad de un ciclo de reloj. De este modo, se dobla la velocidad de reloj interna, lo que permite la realización de dos tareas en un ciclo de reloj externo. Hemos visto un ejemplo de esta aproximación en el MIPS R4000.

La Figura 13.2 compara las dos aproximaciones. La parte superior del diagrama ilustra un cauce normal, usado como base de la comparación. El cauce base emite una instrucción por ciclo de reloj, y puede ejecutar una etapa del cauce en cada ciclo. El cauce tiene cuatro etapas: captación de instrucción, decodificación de la operación, ejecución de la operación y escritura del resultado. La etapa de ejecución se ha destacado con una trama por motivos de claridad. Observe que, aunque se ejecuten varias instrucciones concurrentemente, sólo hay una instrucción en la etapa de ejecución en un determinado instante.

La siguiente parte del diagrama muestra una implementación supersegmentada que es capaz de ejecutar dos etapas del cauce por ciclo de reloj. Una forma alternativa de enfocar esto consiste en que las funciones realizadas en cada etapa se pueden dividir en dos partes no solapadas, y que cada una se ejecuta en medio ciclo de reloj. Se dice que una implementación de un cauce supersegmentado que se comporta de esta forma, es de grado 2. Por último, la parte inferior del diagrama muestra una implementación superescalar capaz de ejecutar en paralelo dos instrucciones en cada etapa. Naturalmente, también son posibles implementaciones supersegmentadas y superescalares de mayor grado.

Las dos realizaciones, supersegmentada y superescalar, representadas en la Figura 13.2, ejecutan el mismo número de instrucciones en el mismo tiempo cuando funcionan de forma ininterrumpida. El procesador supersegmentado se queda atrás con respecto al procesador superescalar al comienzo del programa y en cada destino de un salto.

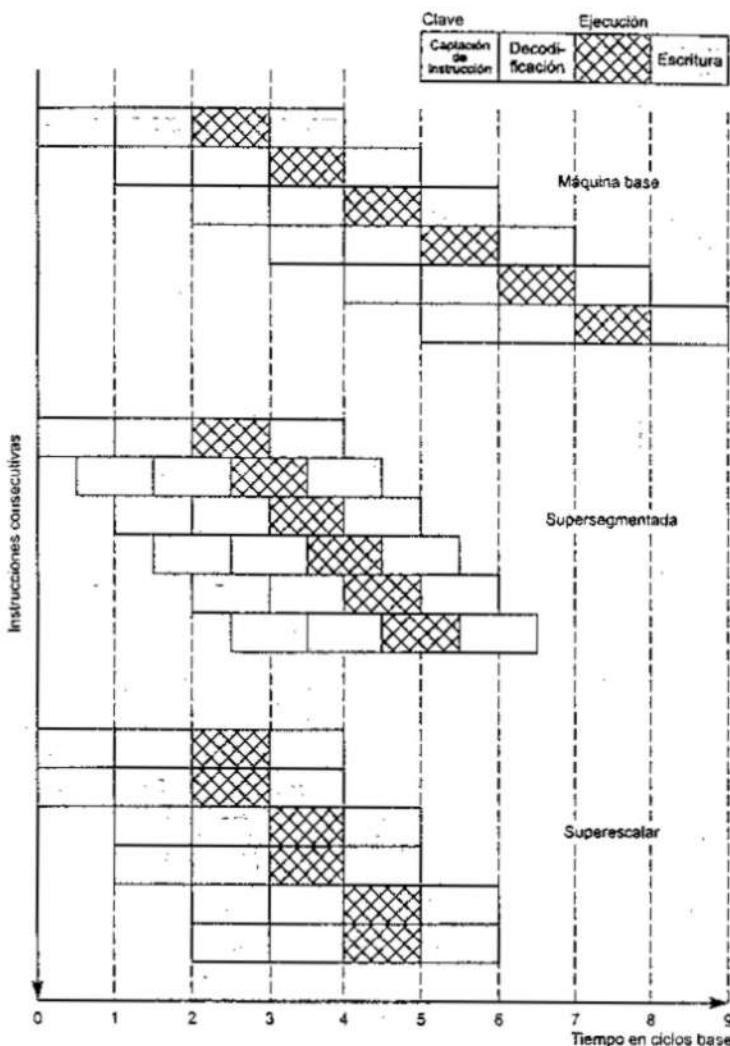


Figura 13.2. Comparación de las aproximaciones superescalares y supersegmentadas.

## LIMITACIONES

La aproximación superescalar depende de la habilidad para ejecutar múltiples instrucciones en paralelo. La expresión **paralelismo a nivel de instrucciones** se refiere al grado en el que, en promedio, las instrucciones de un programa se pueden ejecutar en paralelo. Para maximizar el paralelismo a nivel de instrucciones, se puede usar una combinación de optimizaciones rea-

lizadas por el compilador y de técnicas hardware. Antes de examinar las técnicas de diseño utilizadas en las máquinas superescalares para aumentar el paralelismo a nivel de instrucciones, debemos considerar las limitaciones fundamentales del paralelismo a las que el sistema tiene que enfrentarse. [JOHN91] enumera cinco limitaciones:

- Dependencia de datos verdadera.
- Dependencia relativa al procedimiento.
- Conflictos en los recursos.
- Dependencia de salida.
- Antidependencia.

En lo que resta de esta sección, examinamos las tres primeras limitaciones. El estudio de las dos últimas debe esperar a algunos desarrollos de la siguiente sección.

### Dependencia de datos verdadera

Consideremos la siguiente secuencia:

```
add r1, r2 ;cargar el registro r1 con el contenido de r2 más
 el contenido de r1
move r3, r1 ;cargar el registro r3 con el contenido de r1
```

La segunda instrucción se puede captar y decodificar, pero no se puede ejecutar hasta que finalice la ejecución de la primera instrucción. El motivo es que la segunda instrucción necesita un dato producido por la primera instrucción. A esta situación se hace referencia como «dependencia de datos verdadera» (también llamada «dependencia de flujo» o «dependencia escritura/lectura»).

La Figura 13.3 ilustra esta dependencia en una máquina superescalar de grado 2. Si no hay dependencias, se pueden captar y ejecutar dos instrucciones en paralelo. En caso de que exista dependencia de datos entre la primera y la segunda instrucción, se retrasa la segunda instrucción tantos ciclos de reloj como sea necesario para eliminar la dependencia. En general, cualquier instrucción debe retrasarse hasta que todos sus valores de entrada estén disponibles.

En un cauce escalar simple, la secuencia de instrucciones anterior no causaría ningún retraso. Sin embargo, consideremos la siguiente secuencia, en la cual una de las cargas se hace desde la memoria, en lugar de desde un registro:

```
load r1, ef ;cargar el registro r1 con el contenido de la dirección
 de memoria efectiva ef
move r3, r1 ;cargar el registro r3 con el contenido de r1
```

Un procesador RISC típico tarda dos o más ciclos en realizar una carga desde memoria, debido al tiempo de acceso a memoria o cache externas al chip. Una forma de compensar este retraso consiste en que el compilador reordene las instrucciones, de tal modo que una o más instrucciones posteriores que no dependan de la carga desde memoria puedan empezar a fluir a través del cauce. Este esquema es menos efectivo en el caso de un cauce superescalar; las instrucciones independientes que se ejecutan durante la carga se ejecutan probablemente en el primer ciclo de ésta, dejando al procesador sin nada que hacer hasta que concluya la carga.

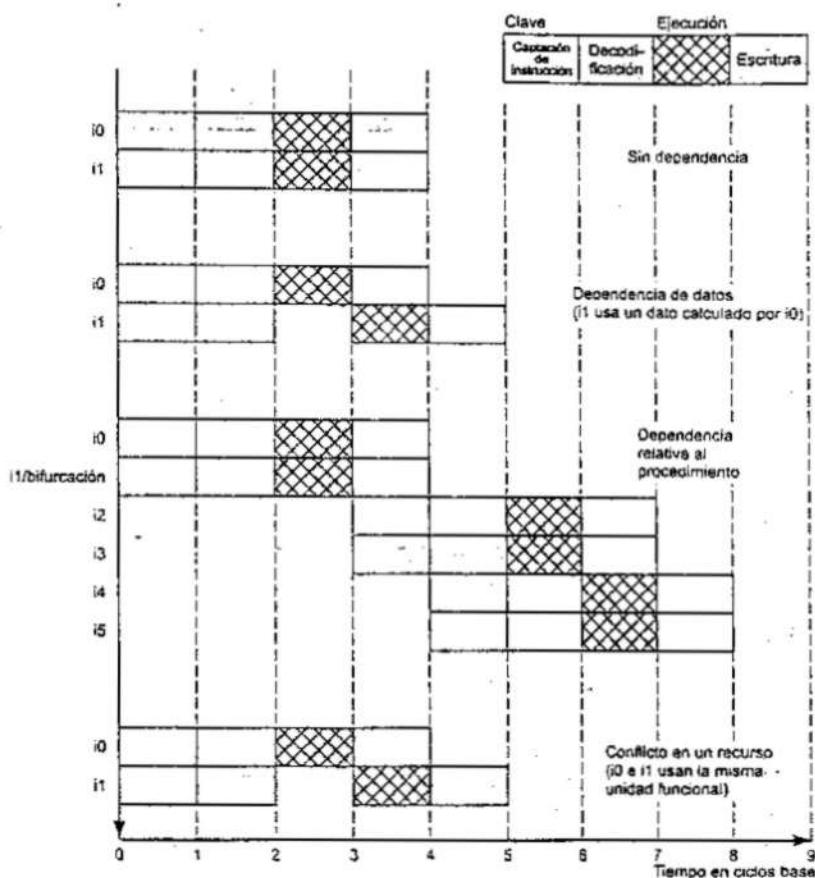


Figura 13.3. Efecto de las dependencias.

#### Dependencias relativas al procedimiento

Según se discutió en el Capítulo 11, la presencia de bifurcaciones en una secuencia de instrucciones complica el funcionamiento del cauce. Las instrucciones que siguen a una bifurcación (en la que se puede saltar o no) tienen una dependencia relativa al procedimiento en esa bifurcación, y no pueden ejecutarse hasta que ésta lo haga. La Figura 13.3 ilustra el efecto de una bifurcación en un cauce superescalar de grado 2.

Como hemos visto, este tipo de dependencia relativa al procedimiento también afecta a un cauce escalar. Además, las consecuencias para un cauce superescalar son más graves, ya que se pierde un mayor número de oportunidades de comenzar a ejecutar instrucciones en cada ciclo de retraso.

Si se usan instrucciones de longitud variable, surge otro tipo de dependencia relativa al procedimiento. Puesto que no se conoce la longitud de una instrucción concreta, ésta ha de

decodificarse al menos parcialmente, antes de captar la siguiente instrucción. Ello impide la captación simultánea necesaria en un cauce superescalal. Ésta es una de las razones por las que las técnicas superescalares se aplican más fácilmente a arquitecturas RISC o similares, que tienen una longitud de instrucción fija.

### Conflictos en los recursos

Un conflicto en un recurso es una pugna de dos o más instrucciones por el mismo recurso al mismo tiempo. Ejemplos de recursos son las memorias, las cachés, los buses, los puertos del banco de registros y las unidades funcionales (por ejemplo, un sumador de la ALU).

Desde el punto de vista del cauce, un conflicto en los recursos presenta el mismo comportamiento que una dependencia de datos (Figura 13.3). Sin embargo, hay algunas diferencias. En primer lugar, los conflictos en los recursos pueden superarse duplicando éstos, mientras que una dependencia de datos verdadera no se puede eliminar. Además, cuando una operación tarda mucho tiempo en finalizar, los conflictos en los recursos se pueden minimizar segmentando la unidad funcional apropiada.

## 13.2. CUESTIONES RELACIONADAS CON EL DISEÑO

### PARALELISMO A NIVEL DE INSTRUCCIONES Y PARALELISMO DE LA MÁQUINA

[JOUP89a] hace una importante distinción entre dos conceptos relacionados: el paralelismo a nivel de instrucciones y el paralelismo de la máquina. Existe paralelismo a nivel de instrucciones cuando las instrucciones de una secuencia son independientes y, por tanto, pueden ejecutarse en paralelo solapándose.

Como ejemplo del concepto de paralelismo a nivel de instrucciones, consideremos los dos siguientes fragmentos de código [JOUP89b]:

|                  |                  |
|------------------|------------------|
| Load R1 ← R2     | Add R3 ← R3, "1" |
| Add R3 ← R3, "1" | Add R4 ← R3, R2  |
| Add R4 ← R4, R2  | Store {R4} ← R0  |

Las tres instrucciones de la izquierda son independientes y, en teoría, las tres podrían ejecutarse en paralelo. Por contraste, las tres instrucciones de la derecha no pueden ejecutarse en paralelo, porque la segunda instrucción usa el resultado de la primera, y la tercera instrucción usa el resultado de la segunda.

El paralelismo a nivel de instrucciones depende de la frecuencia de dependencias de datos verdaderas y dependencias relativas al procedimiento que haya en el código. Estos factores dependen a su vez de la arquitectura del repertorio de instrucciones y de la aplicación. El paralelismo a nivel de instrucciones depende también de lo que [JOUP89a] llama «espera de una operación»: el tiempo que transcurre hasta que el resultado de una instrucción está disponible para ser usado como operando de una instrucción posterior. La espera determina cuánto retraso causará una dependencia de datos o relativa al procedimiento.

El paralelismo de la máquina es una medida de la capacidad del procesador para sacar partido al paralelismo a nivel de instrucciones. El paralelismo de la máquina depende del número de instrucciones que pueden captarse y ejecutarse al mismo tiempo (número de cauces paralelos), y de la velocidad y sofisticación del mecanismo que usa el procesador para localizar instrucciones independientes.

Tanto el paralelismo a nivel de instrucciones como el paralelismo de la máquina son factores importantes para mejorar las prestaciones. Un programa puede no tener el suficiente nivel de paralelismo a nivel de instrucciones como para sacar el máximo partido al paralelismo de la máquina. El empleo de una arquitectura con instrucciones de longitud fija, como en un RISC, aumenta el paralelismo a nivel de instrucciones. Por otra parte, un escaso paralelismo de la máquina limitará las prestaciones, sin que importe la naturaleza del programa.

## POLÍTICAS DE EMISIÓN DE INSTRUCCIONES

Como mencionamos antes, el paralelismo de la máquina no es sencillamente una cuestión de tener múltiples réplicas de cada etapa del cauce. El procesador, además, tiene que ser capaz de identificar el paralelismo a nivel de instrucciones, y organizar la captación, decodificación y ejecución de las instrucciones en paralelo. [JOHN91] utiliza el término **emisión de instrucciones** para referirse al proceso de iniciar la ejecución de instrucciones en las unidades funcionales del procesador, y el término **política de emisión de instrucciones** para referirse al protocolo usado para emitir instrucciones.

Esencialmente, el procesador intenta localizar instrucciones más allá del punto de ejecución en curso, que puedan introducirse en el cauce y ejecutarse. Con respecto a esto, son importantes tres ordenaciones:

- El orden en que se captan las instrucciones.
- El orden en que se ejecutan las instrucciones.
- El orden en que las instrucciones actualizan los contenidos de los registros y de las posiciones de memoria.

Cuanto más sofisticado sea el procesador, menos limitado estará por la estrecha relación entre estas ordenaciones. Para optimizar la utilización de los diversos elementos del cauce, el procesador tendrá que alterar uno o más de estos órdenes con respecto al orden que se encuentra en una ejecución secuencial estricta. La única restricción que tiene el procesador es que el resultado debe ser correcto. De este modo, el procesador debe acomodar las diversas dependencias y conflictos discutidos antes.

En términos generales, podemos agrupar las políticas de emisión de instrucciones de los procesadores superescalares en las siguientes categorías:

- Emisión en orden y finalización en orden.
- Emisión en orden y finalización desordenada.
- Emisión desordenada y finalización desordenada.

### Emisión en orden y finalización en orden

La política de emisión de instrucciones más sencilla es emitir instrucciones en el orden exacto en que lo haría una ejecución secuencial (emisión en orden), y escribir los resultados en ese mismo orden (finalización en orden). Ni siquiera los cauces escalares siguen una política tan ingenua. No obstante, es útil considerar esta política como base con la cual comparar otras aproximaciones más sofisticadas.

La Figura 13.4a ofrece un ejemplo de esta política. Suponemos un cauce supercalar capaz de captar y decodificar dos instrucciones a la vez con tres unidades funcionales independientes (por ejemplo, aritmética entera, aritmética de coma flotante), y con dos copias de la etapa de escritura del cauce. El ejemplo supone las siguientes restricciones para un fragmento de código de seis instrucciones:

| Decodificación | Ejecución | Escritura | Ciclo |
|----------------|-----------|-----------|-------|
| I1 I2          | I1 I2     | I1 I2     | 1     |
| I3 I4          | I3 I4     | I3 I4     | 2     |
| I3 I4          | I3 I4     | I3 I4     | 3     |
| I4 I4          | I4 I4     | I4 I4     | 4     |
| I5 I6          | I5 I6     | I5 I6     | 5     |
| I6             | I6        | I6        | 6     |
| I6             | I6        | I6        | 7     |
| I5 I6          | I5 I6     | I5 I6     | 8     |

(a) Emisión en orden y finalización en orden

| Decodificación | Ejecución | Escritura | Ciclo |
|----------------|-----------|-----------|-------|
| I1 I1          | I1 I1     | I1 I1     | 1     |
| I3 I4          | I1 I2     | I1 I2     | 2     |
| I3 I4          | I1 I3     | I2 I3     | 3     |
| I5 I6          | I2 I4     | I1 I3     | 4     |
| I5 I6          | I3 I5     | I4 I5     | 5     |
| I6             | I4 I6     | I5 I6     | 6     |
| I6             | I6        | I6        | 7     |

(b) Emisión en orden y finalización desordenada

| Decodificación | Ventana  | Ejecución | Escritura   | Ciclo |
|----------------|----------|-----------|-------------|-------|
| I1 I2          | I1 I2    | I1 I2     | I1 I2       | 1     |
| I3 I4          | I3 I4    | I1 I3     | I2 I3       | 2     |
| I5 I6          | I4 I5 I6 | I4 I5 I6  | I1 I4 I5 I6 | 3     |
| I6             | I5       | I5        | I5          | 4     |
| I6             |          |           |             | 5     |
|                |          |           |             | 6     |

(c) Emisión desordenada y finalización desordenada

Figura 13.4. Políticas de emisión y finalización de instrucciones en un cauce superescalares.

- I1 necesita dos ciclos para ejecutarse.
- I3 e I4 compiten por la misma unidad funcional.
- I5 depende de un valor producido por I4.
- I5 e I6 compiten por una unidad funcional.

Las instrucciones se captan de dos en dos y se pasan a la unidad de decodificación. Como las instrucciones se captan por parejas, las dos siguientes instrucciones tienen que esperar hasta que la pareja de etapas de decodificación del cauce se encuentre vacía. Para garantizar la finalización en orden, cuando hay una pugna por una unidad funcional o cuando una unidad funcional necesita más de un ciclo para generar un resultado, la emisión de instrucciones se detiene temporalmente.

En este ejemplo, el tiempo transcurrido desde la decodificación de la primera instrucción hasta la escritura de los últimos resultados es de ocho ciclos.

### Emisión en orden y finalización desordenada

La finalización desordenada se usa en los procesadores RISC escalares para mejorar la velocidad de las instrucciones que necesitan muchos ciclos. La Figura 13.4b ilustra su uso en un procesador superescalar. La instrucción I2 se puede ejecutar hasta su conclusión antes de que acabe I1. Ello permite a I3 terminar antes, con el resultado neto de ahorrar un ciclo.

Con la finalización desordenada, puede haber cualquier número de instrucciones en la etapa de ejecución en un momento dado, hasta alcanzar el máximo grado de paralelismo de la máquina, ocupando todas las unidades funcionales. La emisión de instrucciones se para cuando hay una pugna por un recurso, una dependencia de datos o una dependencia relativa al procedimiento.

A parte de las limitaciones anteriores, surge una nueva dependencia, a la cual nos referimos anteriormente como dependencia de salida (también llamada dependencia escritura-escritura). El siguiente fragmento de código ilustra esta dependencia («op» representa cualquier operación):

```
I1: R3 ← R3 op R5
I2: R4 ← R3 + 1
I3: R3 ← R5 + 1
I4: R7 ← R3 op R4
```

La instrucción I2 no puede ejecutarse antes que la instrucción I1, ya que necesita el resultado almacenado en el registro R3 por I1; éste es un ejemplo de dependencia de datos verdadera, como se describió en la Sección 13.1. De un modo parecido, I4 debe esperar a I3, porque usa un resultado producido por ésta. ¿Qué ocurre con la relación entre I1 e I3? Aquí no hay dependencia de datos, tal y como la hemos definido. Sin embargo, si I3 se ejecuta hasta el final antes que I1, se captará un valor incorrecto del contenido de R3 para la ejecución de I4. Por consiguiente, I3 debe terminar después de I1 para producir el valor correcto de salida. Para asegurar esto, la emisión de la tercera instrucción debe detenerse si su resultado puede ser sobrescrito más tarde por una instrucción anterior que tarda más en finalizar.

La finalización desordenada necesita una lógica de emisión de instrucciones más compleja que la finalización en orden. Además, es más difícil ocuparse de las interrupciones y excepciones. Cuando ocurre una interrupción, la ejecución de instrucciones se suspende en el punto actual, para reanudarse posteriormente. El procesador debe asegurar que la reanudación tiene en cuenta que, en el momento de la interrupción, algunas instrucciones posteriores a la instrucción que provocó dicha interrupción pueden haber finalizado ya.

### Emisión desordenada y finalización desordenada

Con la emisión en orden, el procesador sólo decodificará instrucciones hasta el punto de dependencia o conflicto. No ~~se~~ decodifican más instrucciones hasta que el conflicto se resuelve. Por consiguiente, el procesador no puede buscar más allá del punto de conflicto, instrucciones que podrían ser independientes de las que hay en el cauce y que podrían introducirse provechosamente en éste.

Para permitir la emisión desordenada, es necesario desacoplar las etapas del cauce de decodificación y ejecución. Esto se hace mediante un buffer llamado ventana de instrucciones. Con esta organización, cuando un procesador termina de decodificar una instrucción, coloca ésta en la ventana de instrucciones. Mientras el buffer no se llene, el procesador puede continuar captando y decodificando nuevas instrucciones. Cuando una unidad funcional de la eta-

pa de ejecución queda disponible, se puede emitir una instrucción desde la ventana de instrucciones a la etapa de ejecución. Cualquier instrucción puede emitirse, siempre que (1) necesita la unidad funcional particular que está disponible y (2) ningún conflicto ni dependencia bloquee.

El resultado de esta organización es que el procesador tiene capacidad de anticipación, que le permite identificar instrucciones independientes que pueden introducirse en la etapa de ejecución. Las instrucciones se emiten desde la ventana de instrucciones, sin que se tenga en cuenta su orden original en el programa. Como antes, la única restricción es que el programa funcione correctamente.

La Figura 13.4c ilustra esta política. En cada ciclo se captan dos instrucciones y se lleva a la etapa de decodificación. En cada ciclo, sujetas a la restricción del tamaño del buffer, se transfieren dos instrucciones desde la etapa de decodificación a la ventana de instrucciones. En este ejemplo, es posible emitir la instrucción I6 delante de la I5 (recuerde que I5 depende de I4, pero I6 no). De este modo se ahorra un ciclo en las etapas de ejecución y de escritura; el ahorro de principio a fin, comparado con la Figura 13.4b, es de un ciclo.

La ventana de instrucciones se representa en la Figura 13.4c para ilustrar su papel. No obstante, esta ventana no es una etapa adicional del cauce. Que una instrucción esté en la ventana indica sencillamente que el procesador tiene suficiente información sobre esa instrucción como para decidir si puede emitirse.

La política de emisión desordenada y finalización desordenada está sujeta a las mismas restricciones descritas anteriormente. Una instrucción no puede emitirse si viola una dependencia o conflicto. La diferencia es que ahora hay más instrucciones dispuestas a ser emitidas, reduciendo la probabilidad de que una etapa del cauce tenga que pararse. Además, surge una nueva dependencia, a la que nos referimos antes como *antidependencia* (también llamada dependencia lectura-escritura). El fragmento de código considerado antes ilustra esta dependencia:

```
I1: R3 ← R3 op R5
I2: R4 ← R3 + 1
I3: R3 ← R5 + 1
I4: R7 ← R3 op R4
```

La instrucción I3 no puede finalizar antes de que la instrucción I2 comience a ejecutarse y haya captado sus operandos. Esto es así debido a que I3 actualiza el registro R3, que es un operando fuente de I2. El término *antidependencia* se usa porque la restricción es similar a la de la dependencia verdadera, pero a la inversa: en lugar de que la primera instrucción produzca un valor que usa la segunda instrucción, la segunda instrucción destruye un valor que usa la primera instrucción.

## RENOMBRAMIENTO DE REGISTROS

Hemos visto que permitir la emisión desordenada de instrucciones y/o la finalización desordenada, puede dar origen a dependencias de salida y antidependencias. Estas dependencias son distintas de las dependencias de datos verdaderas y de los conflictos en los recursos, que reflejan el flujo de datos a través de un programa y la secuencia de ejecución. Las dependencias de salida y las antidependencias, por otra parte, surgen porque los valores de los registros no pueden reflejar ya la secuencia de valores dictada por el flujo del programa.

Cuando las instrucciones se emiten y se completan secuencialmente, es posible especificar el contenido de cada registro en cada punto de la ejecución. Cuando se usan técnicas de desordenación, los valores de los registros no pueden conocerse completamente en cada instante temporal, considerando sólo la secuencia de instrucciones dictada por el programa. En

realidad, los valores entran en conflicto por el uso de los registros, y el procesador debe resolver tales conflictos deteniendo ocasionalmente alguna etapa del cauce.

Las antidependencias y las dependencias de salida son dos ejemplos de conflictos de almacenamiento. Varias instrucciones compiten por el uso de los mismos registros, generando restricciones en el cauce que reducen las prestaciones. El problema se agudiza cuando se utilizan técnicas de optimización de registros (estudiadas en el Capítulo 12), ya que estas técnicas del compilador intentan maximizar el uso de registros, maximizando por tanto el número de conflictos de almacenamiento.

Hay un método para hacer frente a este tipo de conflictos de almacenamiento, que se basa en una solución tradicional para los conflictos en los recursos: la duplicación de recursos. En este contexto, la técnica se conoce como renombramiento de registros. Básicamente, el hardware del procesador asigna dinámicamente los registros, que están asociados con los valores que necesitan las instrucciones en diversos instantes de tiempo. Cuando se crea un nuevo valor de registro (es decir, cuando se ejecuta una instrucción que tiene un registro como operando destino), se asigna un nuevo registro para ese valor. Las instrucciones posteriores que accedan a ese valor como operando fuente en ese registro, tienen que sufrir un proceso de renombramiento: las referencias a registros de esas instrucciones han de revisarse para referenciar el registro que contiene el valor que se necesita. De este modo, las referencias a un mismo registro original en diferentes instrucciones, pueden referirse a distintos registros reales, suponiendo diferentes valores.

Consideremos cómo se podría usar el renombramiento de registros en el fragmento de código que estamos examinando:

|     |                                         |
|-----|-----------------------------------------|
| I1: | $R3_b \leftarrow R3_a \text{ op } R5_a$ |
| I2: | $R4_b \leftarrow R3_b + 1$              |
| I3: | $R3_c \leftarrow R5_a + 1$              |
| I4: | $R7_b \leftarrow R3_c \text{ op } R4_b$ |

La referencia a un registro sin el subíndice alude a una referencia a un registro lógico encontrada en la instrucción. La referencia a un registro con el subíndice alude a un registro hardware asignado para contener un nuevo valor. Cuando se hace una nueva asignación para un registro lógico particular, se hace que las referencias de instrucciones posteriores a ese registro lógico como operando fuente, se refieran al registro hardware asignado más recientemente (reciente en términos de la secuencia de instrucciones del programa).

En este ejemplo, la creación del registro  $R3_c$  en la instrucción I3, evita la antidependencia de la segunda instrucción y la dependencia de salida de la primera instrucción, y no impide que I4 acceda a un valor correcto. El resultado es que I3 puede emitirse inmediatamente; sin renombramiento, I3 no puede emitirse hasta que la primera instrucción haya finalizado y la segunda instrucción se haya emitido.

## PARALELISMO DE LA MÁQUINA

En los párrafos precedentes hemos estudiado tres técnicas hardware que se pueden usar en un procesador superescalar para aumentar las prestaciones: duplicación de recursos, emisión desordenada y renombramiento. En [SMIT89] se presentó un estudio que aclara la relación entre estas técnicas. El estudio utilizó una simulación que modelaba una máquina de las características del MIPS R2000, aumentada con algunas características superescalares. Se simularon diferentes secuencias de programa.

La Figura 13.5 muestra los resultados. En las dos gráficas, el eje vertical corresponde al incremento de velocidad medio de la máquina superescalar respecto a la máquina escalar.

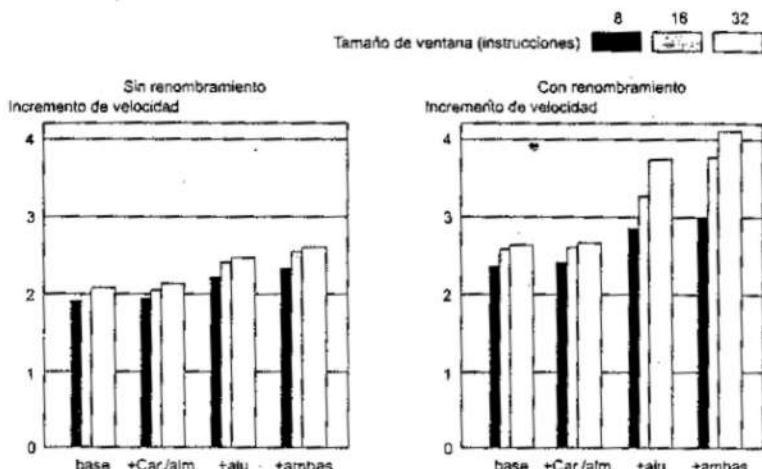


Figura 13.5. Incremento de velocidad de diversas organizaciones de una máquina, sin dependencias relativas al procedimiento.

El eje horizontal muestra los resultados para cuatro organizaciones del procesador alternativas. La máquina base no duplica ninguna de las unidades funcionales, pero puede emitir instrucciones desordenadamente. La segunda configuración duplica la unidad funcional de carga/almacenamiento que accede a la cache de datos. La tercera configuración duplica la ALU, y la cuarta configuración duplica, tanto la unidad de carga/almacenamiento, como la ALU. En las dos gráficas, los resultados se muestran para tamaños de ventana de instrucciones de 8, 16 y 32 instrucciones, que determinan el grado de anticipación que puede tener el procesador. La diferencia entre las dos gráficas es que en la segunda se permite el renombramiento de registros. Esto es equivalente a decir que la primera gráfica refleja una máquina limitada por todas las dependencias, mientras que la segunda gráfica corresponde a una máquina limitada sólo por las dependencias verdaderas.

Las dos gráficas combinadas ofrecen algunas conclusiones importantes. La primera es que, probablemente, no merece la pena añadir unidades funcionales sin renombramiento de registros. Hay algunas mejoras de poca importancia en las prestaciones, pero con el coste de una complejidad del hardware aumentada. Con el renombramiento de registros, que elimina las antidependencias y las dependencias de salida, se logran ganancias notables, añadiendo más unidades funcionales. Observe, no obstante, que hay una diferencia significativa en la ganancia alcanzable con el uso de una ventana de 8 instrucciones, frente a la que se consigue usando una ventana de instrucciones mayor. Esto indica que, si la ventana de instrucciones es demasiado pequeña, las dependencias de datos impiden la utilización efectiva de las unidades funcionales adicionales; es importante que el procesador pueda mirar hacia delante bastante lejos en busca de instrucciones independientes que permitan aprovechar más el hardware.

## PREDICCIÓN DE SALTOS

Cualquier máquina segmentada de altas prestaciones debe estudiar la cuestión del tratamiento de las bifurcaciones. Por ejemplo, el Intel 80486 soluciona el problema captando, tanto la

siguiente instrucción secuencial a la de bifurcación, como la instrucción destino del salto. Sin embargo, como hay dos etapas en el cauce entre la precaptación y la ejecución, esta estrategia incurre en un retardo de dos ciclos cuando se produce el salto.

Con la llegada de las máquinas RISC, se exploró la estrategia de salto retardado. Ésta permite al procesador calcular el resultado de las instrucciones de bifurcación condicional antes de que se precapte cualquier instrucción inservible. Con este método, el procesador siempre ejecuta la instrucción que sigue inmediatamente a la de bifurcación. Esto mantiene lleno el cauce mientras el procesador capta un nuevo flujo de instrucciones.

Con el desarrollo de las máquinas superescalares, la estrategia de salto retardado ha perdido interés. El motivo es que hay que ejecutar múltiples instrucciones en el ciclo de retardo, lo que plantea varios problemas relacionados con las dependencias entre instrucciones. De este modo, las máquinas superescalares han regresado a las técnicas de predicción de saltos anteriores a las de los RISC. Algunas, como el PowerPC 601, usan una técnica sencilla de predicción de saltos estática. Los procesadores más sofisticados, como el PowerPC 620 y el Pentium II, usan predicción dinámica de saltos basada en el análisis de la historia de las bifurcaciones.

## EJECUCIÓN SUPERESCALAR

Estamos ahora en condiciones de dar una visión de conjunto de la ejecución superescalar de programas, ilustrada en la Figura 13.6. El programa que se va a ejecutar consiste en una secuencia lineal de instrucciones. Se trata del programa estático, tal como fue escrito por el programador o generado por el compilador. El proceso de captación de instrucciones, que incluye la predicción de saltos, se usa para formar un flujo dinámico de instrucciones. Se examinan las dependencias de este flujo, y el procesador puede eliminar las que sean artificiales. El procesador envía entonces las instrucciones a una ventana de ejecución. En esta ventana, las instrucciones ya no forman un flujo secuencial, sino que están estructuradas de acuerdo a sus dependencias de datos verdaderas. El procesador lleva a cabo la etapa de ejecución de cada instrucción en un orden determinado por las dependencias de datos verdaderas y la disponibilidad de los recursos hardware. Por último, las instrucciones se vuelven a poner conceptualmente en un orden secuencial y sus resultados se registran.

Al último paso mencionado en el párrafo precedente se le llama *entregar* (commit), o *retirar* (retire), la instrucción. Este paso es necesario por la siguiente razón: debido al uso de múltiples cauces paralelos, las instrucciones pueden terminar en un orden diferente al que muestran en el programa estático. Además, la utilización de predicción de saltos y ejecución especulativa significa que algunas instrucciones pueden completar su ejecución y después ser desechadas porque la bifurcación que llevaba a ellas no se produjo. Por consiguiente, el almacenamiento permanente y los registros visibles por el programa no se pueden actualizar inmediatamente después de que las instrucciones finalicen su ejecución. Los resultados han de mantenerse en algún tipo de almacenamiento temporal que sea utilizable por instrucciones dependientes, y después convertido en permanente, una vez que se determine que el modelo secuencial habría ejecutado la instrucción.

## IMPLEMENTACIÓN SUPERESCALAR

Basándonos en lo discutido hasta ahora, podemos hacer algunos comentarios generales sobre el hardware que requiere el procesador en la aproximación superescalar. En [SMIT95] se enumeran los siguientes elementos principales:

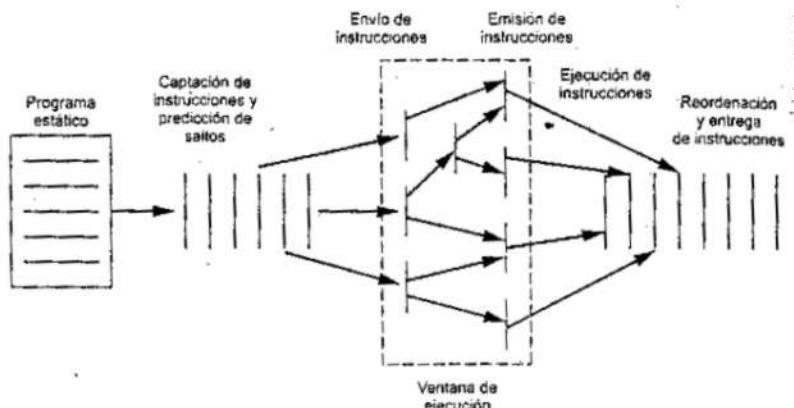


Figura 13.6. Representación conceptual del procesamiento superescalar [SMIT95].

- Estrategias de captación de instrucciones que captan simultáneamente múltiples instrucciones, a menudo prediciendo los resultados de las instrucciones de bifurcación condicional y captando más allá de ellas. Estas funciones requieren la utilización de múltiples etapas de captación y decodificación, y lógica de predicción de saltos.
- Lógica para determinar dependencias verdaderas entre valores de registros, y mecanismos para comunicar esos valores a donde sean necesarios durante la ejecución.
- Mecanismos para iniciar, o emitir, múltiples instrucciones en paralelo.
- Recursos para la ejecución en paralelo de múltiples instrucciones, que incluyan múltiples unidades funcionales segmentadas y jerarquías de memoria capaces de atender múltiples referencias a memoria.
- Mecanismos para entregar el estado del procesador en un orden correcto.

### 13.3. PENTIUM II

Aunque el concepto de diseño superescalar se asocia generalmente con la arquitectura RISC, se pueden aplicar los mismos principios superescalares a una máquina CISC. El ejemplo más notable de ello, tal vez sea el Pentium II. Es interesante observar la evolución de los conceptos superescalares en la línea Intel. El 80486 era una clara máquina tradicional CISC, sin elementos superescalares. El Pentium original tenía unas características superescalares modestas, que consistían en la utilización de dos unidades de ejecución de enteros independientes. El Pentium Pro introdujo un diseño completamente superescalar. El Pentium II tiene básicamente la misma organización superescalar que el Pentium Pro, con la adición de unidades de ejecución MMX.

En la Figura 4.23 se mostró un diagrama de bloques general del Pentium II. Los componentes esenciales de la organización superescalar son: la unidad de captación y decodificación de instrucciones, la unidad de envío y ejecución, y la unidad de retiro. Examinaremos cada una de ellas después de una breve visión de conjunto.

| IFU1 | IFU2 | IFU3 | ID1 | ID2 | RAT | ROB | DIS | EX | RET1 | RET2 |
|------|------|------|-----|-----|-----|-----|-----|----|------|------|
|------|------|------|-----|-----|-----|-----|-----|----|------|------|

IFU = Unidad de captación de instrucciones (Instruction Fetch Unit)

ID = Decodificación de instrucciones (Instruction Decode)

RAT = Asignador de registros (Register Allocator)

ROB = Buffer de reordenación (Reorder Buffer)

DIS = Unidad de envío (Dispatcher)

EX = Etapa de ejecución (Execute Stage)

RET = Unidad de retiro (Retire Unit)

Figura 13.7. Cauce segmentado del Pentium II.

El funcionamiento del Pentium II se puede resumir como sigue:

1. El procesador capta instrucciones de memoria en el orden en que aparecen en el programa estático.
2. Cada instrucción se traduce en una o más instrucciones RISC de tamaño fijo, conocidas como «microoperaciones», o «microops».
3. El procesador ejecuta las microops en una organización de cauce superescalar, de modo que las microops se pueden ejecutar desordenadas.
4. El procesador entrega los resultados de la ejecución de cada microop al conjunto de registros, en el orden del flujo del programa original.

En realidad, la arquitectura del Pentium II consta de una envoltura CISC con un núcleo RISC interno. Las microops RISC internas pasan a través de un cauce con, al menos, 11 etapas (Figura 13.7); en algunos casos, las microops necesitan múltiples etapas de ejecución. Lo que se traduce en un cauce aún más largo. Esto contrasta con el cauce de cinco etapas (Figura 11.18) utilizado en los procesadores Intel x86 y en el Pentium.

## UNIDAD DE CAPTACIÓN Y DECODIFICACIÓN DE INSTRUCCIONES

La Figura 13.8 es un esquema simplificado de la unidad de captación/decodificación del Pentium II. La operación de captación consta de tres etapas encauzadas. En primer lugar, la etapa IFU1 capta instrucciones desde la cache de instrucciones, una línea (32 bytes) cada vez. La unidad «Siguiente IP» proporciona la dirección de la siguiente instrucción a captar, y se capta en el buffer IFU1 la línea de cache que contiene la instrucción. Esta operación no se calcula sencillamente incrementando el puntero, porque podría haber un salto o una interrupción pendiente que movie el puntero a una posición diferente.

Después, el contenido del buffer IFU1 pasa a IFU2 (16 bytes cada vez). Esta unidad lleva a cabo dos operaciones en paralelo. IFU2 examina los bytes para determinar los límites de las instrucciones; ésta es una operación necesaria, debido a que las instrucciones del Pentium son de longitud variable. Si alguna de las instrucciones es de salto, la unidad pasa la dirección de memoria correspondiente a la unidad de predicción dinámica de saltos. IFU2 pasa después el bloque de 16 bytes a IFU3, que es responsable de alinear las instrucciones para presentarlas al decodificador apropiado.

Para comprender el funcionamiento de IFU3, hemos de describir la primera etapa de la unidad de decodificación de instrucciones, ID1. Esta etapa es capaz de manejar tres instrucciones máquina del Pentium II en paralelo. ID1 traduce cada instrucción máquina en de una a cuatro microops, cada una de las cuales es una instrucción RISC de 118 bits. Observe que, en

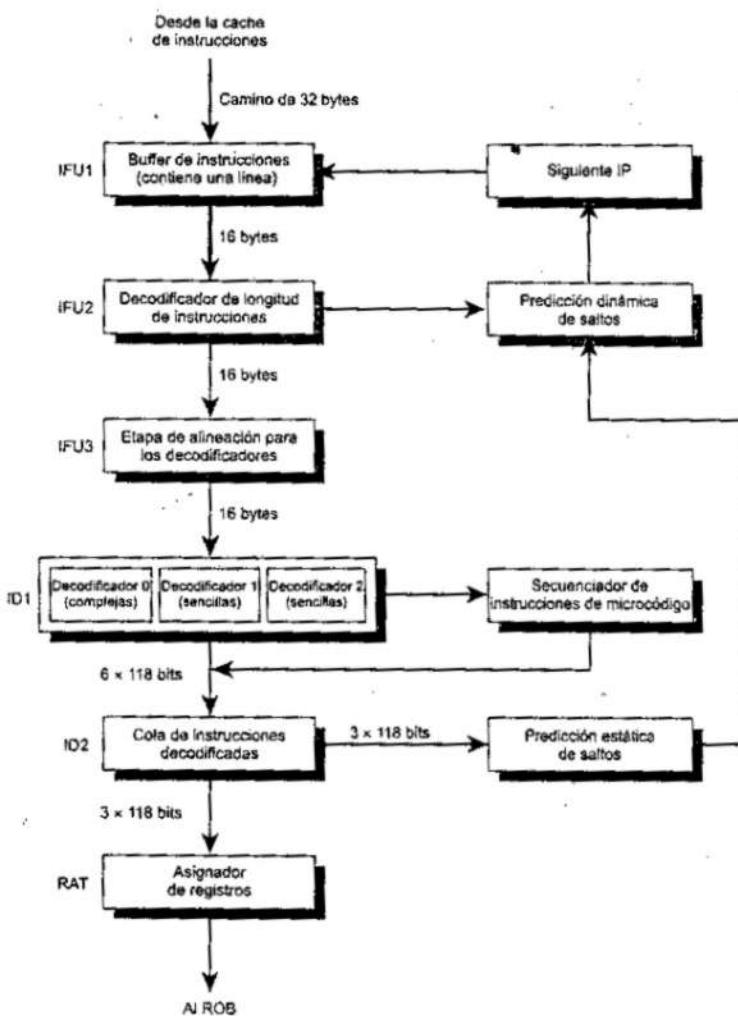


Figura 13.8. Unidad de captación/decodificación del Pentium II.

comparación, las máquinas RISC más puras tienen una longitud de instrucción de sólo 32 bits. La mayor longitud de las microops es necesaria para adaptarse a las operaciones del Pentium, de mayor complejidad. Con todo, las microops son más fáciles de manejar que las instrucciones originales de las que derivan. ID1 contiene tres decodificadores. El primero de ellos puede manejar instrucciones del Pentium que se traduzcan en hasta cuatro microops. El segundo y el tercer decodificador sólo pueden manejar instrucciones sencillas del Pentium que correspondan a una única microop; éstas incluyen instrucciones sencillas registro a regis-

tro, e instrucciones de carga. Para adaptarse a la estructura de ID1, IFU3 rota el contenido de su buffer si es preciso, de manera que la primera instrucción del buffer de 16 bytes sea compleja, y las siguientes dos instrucciones sean sencillas. Si las tres instrucciones son sencillas, no hace falta la rotación. Si más de una instrucción es compleja, las instrucciones deben introducirse en ID1 en etapas, de tal modo que al segundo y al tercer decodificador no se les dé una instrucción compleja.

Hay unas pocas instrucciones que requieren más de cuatro microops. Estas instrucciones se transfieren al secuenciador de instrucciones de microcódigo («microcode instruction sequencer», MIS), que es una ROM de microcódigo que contiene las secuencias de microops (cinco o más) asociadas a una instrucción máquina compleja. Por ejemplo, una instrucción de cadena puede traducirse en una secuencia repetitiva muy larga, incluso de cientos de microops. Por tanto, el MIS es una unidad micropogramada, en el sentido que se discute en la Parte IV.

La salida de ID1 o MIS se introduce en la segunda etapa de la unidad de decodificación, ID2, en un bloque de hasta seis microops a la vez. ID2 encola las microops en el orden original del programa. Llegados a este punto, hay una segunda ocasión para predecir saltos. Si alguna de las microops es de salto, se presenta a la unidad de predicción estática, que a su vez alimenta a la unidad de predicción dinámica de saltos. La predicción de saltos se describe más tarde en esta sección.

Las microops encoladas en ID2 pasan a través de una fase de renombramiento de registros, conocida como «asignador de registros» («register allocator», RAT). El RAT transforma las referencias a los 16 registros de la arquitectura (8 registros de coma flotante, más EAX, EBX, ECX, EDX, ESI, EDI, EBP y ESP) a un conjunto de 40 registros físicos. La etapa elimina dependencias falsas causadas por el limitado número de registros de la arquitectura, preservando las dependencias de datos verdaderas (lecturas después de escrituras). Después, el RAT introduce las microops revisadas al buffer de reordenación («reorder buffer», ROB).

## BUFFER DE REORDENACIÓN

El ROB es un buffer circular que puede contener hasta 40 microops, y que también contiene los 40 registros hardware. Cada entrada del buffer consta de los siguientes campos:

- **Estado:** indica si la microop está lista para ejecutarse, ha sido enviada a su ejecución, o ha terminado de ejecutarse y está lista para su retiro.
- **Dirección de memoria:** dirección de la instrucción del Pentium que generó la microop.
- **Microop:** la operación propiamente dicha.
- **Registro alias:** si la microop referencia uno de los 16 registros de la arquitectura, este campo redirecciona esa referencia a uno de los 40 registros hardware.

Las microops entran al ROB en orden; después, son enviadas desde el ROB a la unidad de envío/ejecución sin orden. El criterio para que una microop sea enviada es que la unidad de ejecución apropiada y todos los datos que requiera la microop se encuentren disponibles. Por último, las microops se retiran del ROB en orden. Para lograr esto, las microops se retiran empezando por la más antigua, después de que cada microop se haya señalado como lista para ser retirada.

## UNIDAD DE ENVÍO/EJECUCIÓN

La Figura 13.9 es un esquema simplificado de la unidad de envío/ejecución del Pentium II. La central de reservas («reservation station», RS) es responsable de recuperar microops del

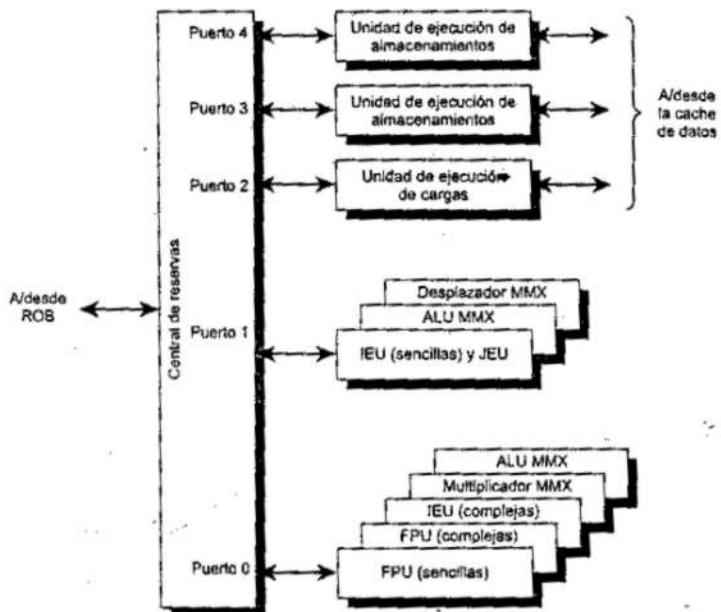


Figura 13.9. Unidad de envío/ejecución del Pentium II.

ROB, enviándolas a su ejecución y guardando los resultados de nuevo en el ROB. La RS busca en el ROB microops cuyo estado indique que la microop dispone de todos sus operandos. Si está disponible la unidad de ejecución que necesita una microop, la RS capta esa microop y la envía a la unidad de ejecución conveniente. Se pueden enviar hasta cinco microops en un ciclo. Si hay más de una microop disponible para una unidad de ejecución dada, la RS las envía secuencialmente desde el ROB. Se trata de un tipo de disciplina FIFO (primero en entrar, primero en salir), que favorece la ejecución en orden, aunque en este momento el flujo de instrucciones se ha reordenado tanto, debido a las dependencias y a los saltos, que se encuentra sustancialmente desordenado.

Hay cinco puertos que unen la RS a las cinco unidades de ejecución. El Puerto 0 se usa para instrucciones con enteros y coma flotante, con la excepción de las operaciones sencillas con enteros y la gestión de las predicciones de salto erróneas, que se asignan al Puerto 1. Además, las unidades de ejecución MMX se asignan a estos dos puertos. Los puertos restantes se utilizan para cargas y almacenamientos en memoria.

Cuando se completa una ejecución, se actualiza la entrada adecuada del ROB, y la unidad de ejecución queda disponible para otra microop.

El flujo sencillo de microops a través de la unidad de envío/ejecución se ve perturbado por las equivocaciones en la predicción de saltos. Si una predicción de salto resulta fallida, habrá microoperaciones en varias etapas de procesamiento que tengan que ser retiradas del cauce. Ello es responsabilidad de la unidad de ejecución de saltos (*jump execution unit*, JEU). Cuando se ejecuta una bifurcación, se compara su resultado con el que pronosticaría el

hardware de predicción. Si los dos no coinciden, la JEU cambia el estado de todas las microops detrás de la bifurcación, para eliminarlas de la reserva de instrucciones. El destino correcto del salto se proporciona entonces a la unidad de predicción de saltos, que regenera el cauce completo a partir de la nueva dirección destino.

## UNIDAD DE RETIRO

La unidad de retiro («retire unit», RU) va procesando el buffer de reordenación para entregar los resultados de la ejecución de instrucciones. En primer lugar, la RU debe tener presentes los fallos en las predicciones de salto, y las microops que se hayan ejecutado pero para las cuales los saltos precedentes no se hayan validado. Una vez que se determina que una microop se ha ejecutado y no es vulnerable a eliminación debido a un fallo de predicción, se marca como lista para ser retirada. Cuando se ha retirado la instrucción del Pentium previa, y todas las microops de la siguiente instrucción se han marcado como listas para ser retiradas, la RU actualiza los registros de la arquitectura afectados por esta instrucción, y quita del ROB la microop.

## PREDICCIÓN DE SALTOS

El Pentium II usa una estrategia de predicción dinámica de saltos basada en la historia de las ejecuciones recientes de instrucciones de bifurcación. Se utiliza un buffer de destino de saltos («branch target buffer», BTB) que guarda información sobre las instrucciones de bifurcación encontradas recientemente. Siempre que aparece una instrucción de bifurcación en el flujo de instrucciones, se comprueba el BTB. Si ya existe una entrada en el BTB, la unidad de instrucciones se guía por la información de historia guardada en esa entrada, para determinar si predice que se producirá el salto. Si se predice un salto, la dirección destino del salto asociada con esta entrada, se utiliza para precaptar la instrucción destino del salto.

Una vez que se ejecuta la instrucción, la parte de historia de la entrada adecuada se actualiza, para que refleje el resultado de la instrucción de bifurcación. Si la instrucción no está representada en el BTB, se carga su dirección en una entrada del BTB; si es preciso, se borra una entrada más antigua.

La descripción de los dos párrafos precedentes se ajusta, en términos generales, a la estrategia de predicción de saltos que se utiliza en el Pentium, además de en el Pentium Pro y el Pentium II. Sin embargo, en el caso del Pentium, se usa un esquema de historia de 2 bits relativamente sencillo. El Pentium Pro y el Pentium II tienen cauces mucho más largos (11 etapas o más comparadas con las 5 del Pentium) y, por tanto, la penalización debida a un error de predicción es mayor. Así pues, el Pentium Pro y el Pentium II emplean un esquema de predicción de saltos más complejo, con más bits de historia para reducir la tasa de fallos de predicción.

El BTB del Pentium II está organizado como una cache asociativa por conjuntos de cuatro vías con 512 líneas. Cada entrada utiliza la dirección de la instrucción de salto como etiqueta. La entrada también incluye la dirección destino del salto de la última vez que se produjo, y un campo de historia de 4 bits. El empleo de cuatro bits de historia contrasta con los 2 bits usados en el Pentium original y en la mayoría de los procesadores superescalares. Con 4 bits, el mecanismo del Pentium II puede tener en cuenta una historia más larga para predecir saltos. Se emplea el algoritmo de Yeh [YEH91]. Los autores de este algoritmo han demostrado que proporciona una reducción significativa de los fallos de predicción, comparado con los algoritmos que utilizan sólo 2 bits de historia [EVER98].

Las bifurcaciones condicionales que no tienen historia en el BTB se predicen usando un algoritmo de predicción estática, de acuerdo con las siguientes reglas:

- En las instrucciones de bifurcación que no son relativas a IP, se predice que se producirá el salto si la bifurcación es un retorno, y que no se saltará en cualquier otro caso.
- En las bifurcaciones condicionales hacia atrás relativas a IP, se predice que se producirá el salto. Esta regla refleja el comportamiento típico en los bucles.
- En las bifurcaciones condicionales hacia delante, se predice que no se producirá el salto.

### T3.4 PowerPC

La arquitectura del PowerPC desciende directamente del IBM 801, el RT PC, y el RS/6000, al que también se alude como una implementación de la arquitectura POWER. Todas estas son máquinas RISC, si bien el primer procesador de la serie que presentó características superescalares fue el RS/6000. La primera implementación de la arquitectura PowerPC, el 601, tiene un diseño superescalar bastante similar al del RS/6000. Los siguientes modelos del PowerPC llevan más lejos el concepto de arquitectura superescalar. En esta sección nos centramos en el 601, que proporciona un buen ejemplo de diseño superescalar basado en un RISC. Al final de la sección, consideraremos brevemente el 620.

#### PowerPC 601

La Figura 13.10 da una visión general de la organización del 601. Como otras máquinas superescalares, el 601 se divide en unidades funcionales independientes, para permitir la ejecución en paralelo. Concretamente, el núcleo del 601 consta de tres unidades de ejecución segmentadas independientes, para enteros, coma flotante y procesamiento de saltos. Juntas, estas unidades pueden ejecutar tres instrucciones al mismo tiempo, ofreciendo un diseño superescalar de grado 3.

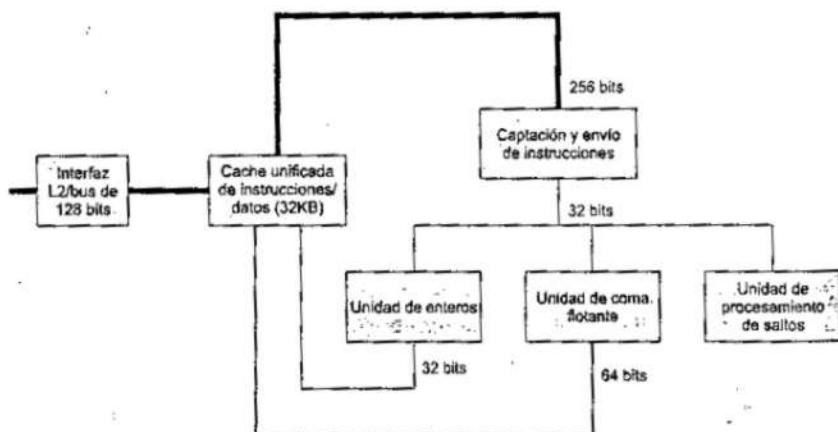


Figura 13.10. Diagrama de bloques del PowerPC 601.

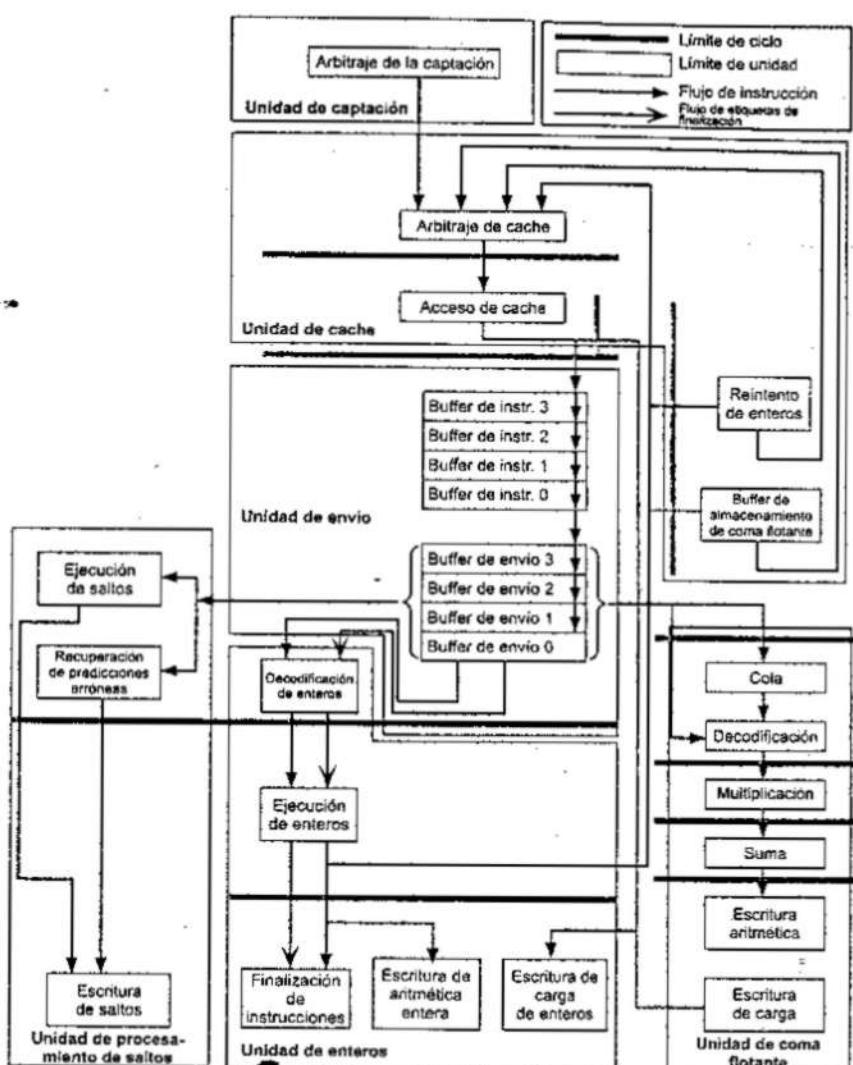


Figura 13.11. Estructura de cauces del PowerPC 601 [POTT94].

La Figura 13.11 muestra un esquema lógico de la arquitectura del 601, destacando el flujo de instrucciones entre las unidades funcionales. La unidad de captación puede precaptar de la cache hasta ocho instrucciones al mismo tiempo. La unidad de cache soporta una cache combinada de instrucciones/datos, y es responsable de suministrar instrucciones a las otras unidades y datos a los registros. La lógica de arbitraje de la cache envía a ésta la dirección del acceso de mayor prioridad.

### Unidad de envío

La unidad de envío toma instrucciones de la cache y las carga en una cola de envío, que puede contener ocho instrucciones a la vez. Procesa este flujo de instrucciones para suministrar un flujo constante de instrucciones a las unidades de procesamiento de saltos, de enteros, y de coma flotante. La mitad superior de la cola actúa, sencillamente, como un buffer que contiene instrucciones hasta que éstas se transfieren a la mitad inferior. Su misión es asegurar que la unidad de envío no se retrase esperando instrucciones provenientes de la cache. En la mitad inferior, las instrucciones se envían según el siguiente esquema:

- **Unidad de procesamiento de saltos:** Se encarga de todas las instrucciones de salto. La instrucción más baja de este tipo en la mitad de abajo de la cola de envío, se emite a la unidad de procesamiento de saltos si ésta puede aceptarla.
- **Unidad de coma flotante:** Se ocupa de todas las instrucciones de coma flotante. La instrucción más baja de este tipo en la mitad de abajo de la cola de envío, se emite a la unidad de coma flotante si el cauce de instrucciones de esta unidad no está lleno.
- **Unidad de enteros:** Se encarga de las instrucciones de enteros, las de carga/almacenamiento entre el banco de registros y la cache, y las instrucciones de comparación de enteros. Una instrucción de enteros se emite sólo después de que se haya filtrado hasta el extremo inferior de la cola de envío.

La posibilidad de emitir instrucciones de salto y de coma flotante desordenadamente desde la cola de envío, ayuda a mantener llenos los cauces de instrucciones de las unidades de procesamiento de saltos y de coma flotante, y a mover instrucciones a través de la cola de envío lo más rápido posible.

La unidad de envío también contiene lógica que le permite calcular la dirección de pre-captación. Continúa captando instrucciones secuencialmente, hasta que una instrucción de salto se transfiere a la mitad inferior de la cola de envío. Cuando la unidad de procesamiento de saltos procesa una instrucción, puede actualizar la dirección de precaptación, de manera que las siguientes instrucciones se capten de la nueva dirección y entren en la cola de envío.

### Cauces de instrucciones

La Figura 13.12 ilustra los cauces de instrucciones de las distintas unidades. Hay un ciclo de captación común a todas las instrucciones: éste tiene lugar antes de que una instrucción se envíe a una unidad concreta. El segundo ciclo comienza con el envío de una instrucción a una unidad particular. Éste se solapa con otras actividades dentro de la unidad. En cada ciclo de reloj, la unidad de envío examina los cuatro elementos inferiores de la cola de instrucciones, y envía hasta tres instrucciones.

En las instrucciones de salto, el segundo ciclo supone la decodificación y ejecución de instrucciones, además de la predicción de saltos. La última actividad se estudia en la siguiente subsección.

La unidad de enteros se ocupa de las instrucciones que causan una operación de carga/almacenamiento de/en memoria (incluyendo carga/almacenamiento de coma flotante), una transferencia registro a registro, o una operación de la ALU. En el caso de carga/almacenamiento, hay un ciclo de generación de dirección, seguido del envío de la dirección resultante a la cache y, si es necesario, un ciclo de escritura. En otras instrucciones, la cache no está involucrada, y hay un ciclo de ejecución seguido de una escritura en un registro.

Las instrucciones de coma flotante siguen un cauce similar, pero con dos ciclos de ejecución, que reflejan la complejidad de las operaciones de coma flotante.

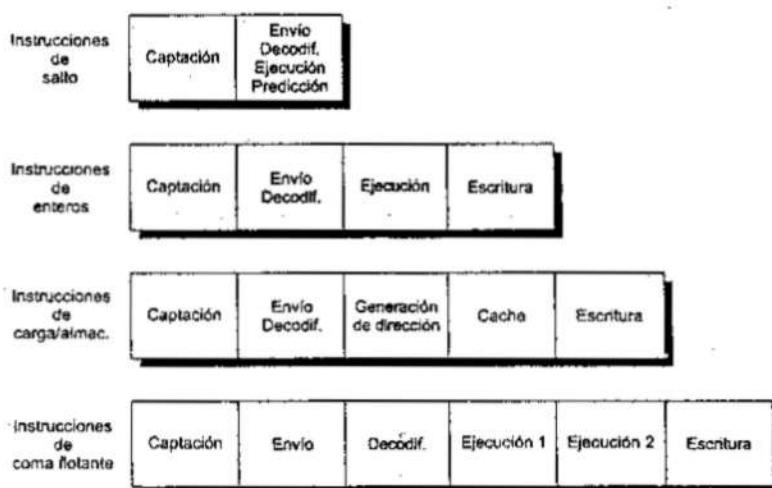


Figura 13.12. Cauce segmentado del PowerPC 601.

Hay otros puntos que son dignos de atención. El registro de condición contiene ocho campos de código de condición de 4 bits independientes. Esto permite guardar los códigos de condición, lo que reduce el interbloqueo o dependencia entre instrucciones. Por ejemplo, el compilador puede transformar la secuencia:

```

comparar
bifurcar
comparar
bifurcar
*
*
*
```

en la secuencia:

```

comparar
comparar
*
*
*
bifurcar
bifurcar
*
*
```

Como cada unidad funcional puede enviar sus códigos de condición a diferentes campos del registro de condición, se pueden evitar los interbloqueos entre instrucciones causados por el hecho de compartir los códigos de condición.

La presencia de los registros salvar y restaurar («save and restore registers», SRR) en el procesador de saltos, le permite gestionar interrupciones sencillas e interrupciones software sin involucrar lógica de otras unidades funcionales. De este modo, los servicios sencillos del sistema operativo se pueden ejecutar rápidamente sin manipulaciones de estados o sincronizaciones complicadas entre las unidades funcionales.

Como el 601 puede emitir instrucciones de salto y de coma flotante desordenadamente, se necesitan controles que aseguren la correcta ejecución. Cuando existe una dependencia (es decir, cuando una instrucción necesita un operando que aún no ha sido calculado por una instrucción previa), el cauce de la unidad correspondiente se detiene.

### PROCESAMIENTO DE SALTOS

La clave de las altas prestaciones de una máquina RISC o superescalar, es su habilidad para optimizar el uso del cauce. Típicamente, el elemento más crítico del diseño es cómo manejar los saltos. En el PowerPC, el procesamiento de saltos es responsabilidad de la unidad de saltos. La unidad está diseñada de manera tal, que en muchos casos los saltos no tengan efecto en el ritmo de ejecución de las otras unidades; a este tipo de saltos se les llama «saltos de cero ciclos». Para conseguir saltos de cero ciclos se emplean las siguientes estrategias:

1. Se utiliza una lógica que examina el buffer de envío en busca de saltos. Se generan direcciones destino de salto, cuando aparece un salto en la mitad inferior de la cola y no hay saltos anteriores pendientes de ejecución.
2. Se intenta determinar el resultado de las bifurcaciones condicionales. Si el código de condición se ha ajustado por adelantado lo suficientemente pronto, este resultado puede determinarse. En todo caso, tan pronto como se encuentre una instrucción de salto, la lógica determina si el salto:
  - a) Se producirá: éste es el caso de los saltos incondicionales y de las bifurcaciones condicionales cuyo código de condición se conoce, e indica que hay que saltar.
  - b) No se producirá: éste es el caso de bifurcaciones condicionales cuyo código de condición se conoce, e indica que no hay salto.
  - c) No se puede determinar todavía: en este caso, se supone que se producirá el salto en las bifurcaciones hacia atrás (típicas en los bucles), y se estima que no se producirá en saltos hacia delante. Las instrucciones secuenciales posteriores a la instrucción de salto se pasan a las unidades de ejecución de manera condicional. Una vez que el código de condición se ajusta en la unidad de ejecución, la unidad de salto, o bien cancela las instrucciones que hay en el cauce y continúa por la instrucción destino captada, si el salto se produce, o bien indica que se ejecuten las instrucciones condicionales. El compilador puede usar un bit del código de la instrucción para invertir este comportamiento implícito.

La incorporación de una estrategia de predicción de saltos, basada en la historia de los saltos, se rechazó, porque los diseñadores pensaron que se conseguía un beneficio mínimo.

Como ejemplo del efecto de la predicción de saltos, considere el programa de la Figura 13.13, y suponga que el procesador de saltos predice que el salto condicional no se producirá (el caso implícito para un salto hacia delante). La Figura 13.14a muestra el efecto en el cauce si, de hecho, el salto no se produce. En el primer ciclo, la coja de envío se carga con ocho instrucciones. Las seis primeras instrucciones son instrucciones con enteros, y se envían,

```

if (a > 0)
else a = a + b + c + d + e;
 a = a - b - c - d - e;

```

(a) Código C

```

#r1 apunta a a,
#r1+4 apunta a b,
#r1+8 apunta a c,
#r1+12 apunta a d,
#r1+16 apunta a e.
lwz r8=a (r1) #carga de a
lwz r12=b (r1, 4) #carga de b
lwz r9=c (r1, 8) #carga de c
lwz r10=d (r1, 12) #carga de d
lwz r11=e (r1, 16) #carga de e
cmpi cr0=r8, 0 #comparación inmediata
bc ELSEZ, cr0/gt=false #bifurcación si el bit indica falso
IF:
 add r12=r8, r12 #suma
 add r12=r12, r9 #suma
 add r12=r12, r10 #suma
 add r4=r12, r11 #suma
 stw a (r1) *r4 #almacenamiento
 b OUT #salto incondicional
ELSE:
 subf r12=r12, r8 #resta
 subf r12=r9, r12 #resta
 subf r12=r10, r12 #resta
 subf r4=r12, r11 #resta
 stw a (r1) *r4 #almacenamiento
OUT:

```

(b) Código ensamblador

Figura 13.13. Ejemplo de código con bifurcación condicional (WEIS94).

una por ciclo, a la unidad de enteros. La instrucción de bifurcación condicional no se puede enviar hasta que llegue a la mitad inferior de la cola de envío, lo que sucede en el ciclo 5. La unidad de saltos predice que este salto no se producirá y, por tanto, la siguiente instrucción secuencial se envía condicionalmente (indicada como D'). El salto no puede resolverse hasta que se ejecute la instrucción de comparación en el ciclo 8. En ese momento, el procesador de saltos confirma que la predicción fue correcta, y la ejecución continúa. No hay retardos, y el cauce se mantiene lleno.

Observe que no se captan instrucciones en los ciclos del 4 al 8. Esto es debido a que la cache está ocupada durante esos ciclos con la etapa de acceso a la cache de las cinco instrucciones de carga. Incluso así, el flujo de instrucciones no se retrasa, porque la cola de envío puede contener ocho instrucciones.

La Figura 13.14b muestra el resultado si la predicción es incorrecta y el salto se produce. En este caso, las tres instrucciones que comienzan en el IF se desechan, y la captación se reanuda en las instrucciones que comienzan en el ELSE. Por consiguiente, la etapa de ejecución del cauce de enteros está desocupada en los ciclos 9 y 10, siendo el resultado la pérdida de dos ciclos por culpa de la predicción incorrecta.

|       |                    | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8  | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|-------|--------------------|---|---|---|---|---|---|---|----|---|----|----|----|----|----|----|
| lwz   | r8=a (r1)          | F | D | S | C | W |   |   |    |   |    |    |    |    |    |    |
| lwz   | r12=b (r1, 4)      | F | * | D | S | C | N |   |    |   |    |    |    |    |    |    |
| lwz   | r9=c (r1, 8)       | F | * | * | D | S | C | W |    |   |    |    |    |    |    |    |
| lwz   | r10=d (r1, 12)     | F | * | * | * | D | S | C | W  |   |    |    |    |    |    |    |
| lwz   | r11=e (r1, 16)     | F | * | * | * | * | D | S | C  | W |    |    |    |    |    |    |
| cmpi  | cc0=r8, 0          | F | * | * | * | * | * | D | S  |   |    |    |    |    |    |    |
| bc    | ELSE, cc0/gt=false | F | * | * | * | S |   |   |    |   |    |    |    |    |    |    |
| IF:   | add r12=r8, r12    | F | * | * | * | * | * | * | D' | S | W  |    |    |    |    |    |
|       | add r12=r12, r9    | F | * | * | * | * | * | * | G  | S | W  |    |    |    |    |    |
|       | add r12=r12, r10   | F | * | * | * | * | * | * | *  | D | S  | W  |    |    |    |    |
|       | add r4=r12, r11    |   |   |   |   |   |   |   | F  | * | D  | S  | W  |    |    |    |
|       | stw a (r1) =r4     |   |   |   |   |   |   |   | G  | * | D  | S  | W  |    |    |    |
| b     | OUT                |   |   |   |   |   |   |   |    |   |    |    |    |    |    |    |
| ELSE: | subf r12=r8, r12   |   |   |   |   |   |   |   |    |   |    |    |    |    |    |    |
|       | subf r12=r12, r9   |   |   |   |   |   |   |   |    |   |    |    |    |    |    |    |
|       | subf r12=r12, r10  |   |   |   |   |   |   |   |    |   |    |    |    |    |    |    |
|       | subf r4=r12, r11   |   |   |   |   |   |   |   |    |   |    |    |    |    |    |    |
|       | stw a (r1) =r4     |   |   |   |   |   |   |   |    |   |    |    |    |    |    |    |
| OUT:  |                    |   |   |   |   |   |   |   |    |   |    |    |    |    |    |    |

(a) Predicción correcta: el salto no se produjo

|       |                    | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8  | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|-------|--------------------|---|---|---|---|---|---|---|----|---|----|----|----|----|----|----|----|
| lwz   | r8=a (r1)          | F | D | S | C | W |   |   |    |   |    |    |    |    |    |    |    |
| lwz   | r12=b (r1, 4)      | F | * | D | S | C | N |   |    |   |    |    |    |    |    |    |    |
| lwz   | r9=c (r1, 8)       | F | * | * | D | S | C | W |    |   |    |    |    |    |    |    |    |
| lwz   | r10=d (r1, 12)     | F | * | * | * | D | S | D | W  |   |    |    |    |    |    |    |    |
| lwz   | r11=e (r1, 16)     | F | * | * | * | * | D | S | C  | W |    |    |    |    |    |    |    |
| cmpi  | cc0=r8, 0          | F | * | * | * | * | * | D | S  |   |    |    |    |    |    |    |    |
| bc    | ELSE, cc0/gt=false | F | * | * | * | S |   |   |    |   |    |    |    |    |    |    |    |
| IF:   | add r12=r8, r12    | F | * | * | * | * | * | * | D' |   |    |    |    |    |    |    |    |
|       | add r12=r12, r9    | F | * | * | * | * | * | * | *  |   |    |    |    |    |    |    |    |
|       | add r12=r12, r10   | F | * | * | * | * | * | * | *  |   |    |    |    |    |    |    |    |
|       | add r4=r12, r11    |   |   |   |   |   |   |   | F  | * | D  | S  | W  |    |    |    |    |
|       | stw a (r1) =r4     |   |   |   |   |   |   |   | G  | * | D  | S  | W  |    |    |    |    |
| b     | OUT                |   |   |   |   |   |   |   |    |   |    |    |    |    |    |    |    |
| ELSE: | subf r12=r8, r12   |   |   |   |   |   |   |   |    |   |    |    |    |    |    |    |    |
|       | subf r12=r12, r9   |   |   |   |   |   |   |   |    |   |    |    |    |    |    |    |    |
|       | subf r12=r12, r10  |   |   |   |   |   |   |   |    |   |    |    |    |    |    |    |    |
|       | subf r4=r12, r11   |   |   |   |   |   |   |   |    |   |    |    |    |    |    |    |    |
|       | stw a (r1) =r4     |   |   |   |   |   |   |   |    |   |    |    |    |    |    |    |    |
| OUT:  |                    |   |   |   |   |   |   |   |    |   |    |    |    |    |    |    |    |

(b) Predicción incorrecta: el salto se produjo

F = Captación

C = Acceso a cache

D = Envío/decodificación

W = Escritura

E = Ejecución/direcciónamiento

S = Envío

Figura 13.14. Predicción de salto: no habrá salto [WEIS94].

### PowerPC 620

El 620 es la primera implementación de 64 bits de la arquitectura PowerPC. En la Figura 4.25 se mostró un diagrama de bloques general de este procesador, que es el mismo que el del más reciente G3. Una característica notable de esta implementación es que incluye seis unidades de ejecución independientes:

- Unidad de instrucciones.
- Tres unidades de enteros.
- Unidad de carga/almacenamiento.
- Unidad de coma flotante.

Esta organización permite al procesador enviar hasta cuatro instrucciones simultáneamente a las tres unidades de enteros y a la de coma flotante.

El 620 emplea una estrategia de predicción de saltos de altas prestaciones, que incluye lógica de predicción, buffers de renombramiento de registros, y centrales de reserva dentro de las unidades de ejecución. Cuando se capta una instrucción, se le asigna un buffer de renombramiento, que contiene temporalmente los resultados de la instrucción, tales como datos a almacenar en registros. Gracias al uso de los buffers de renombramiento, el procesador puede ejecutar *especulativamente* instrucciones basadas en predicción de saltos: si la predicción resulta ser incorrecta, los resultados de las instrucciones especulativas pueden desecharse sin afectar al banco de registros. Una vez confirmado el resultado de un salto, los resultados temporales pueden ser escritos de modo definitivo.

Cada unidad tiene dos o más centrales de reserva, que almacenan instrucciones enviadas, que deben suspenderse en espera de resultados de otras instrucciones. Esta característica posibilita que se retiren estas instrucciones de la unidad de instrucciones, permitiendo a ésta continuar enviando instrucciones a las otras unidades de ejecución.

El 620 puede ejecutar especulativamente hasta cuatro instrucciones de bifurcación no resueltas (frente a una en el 601). La predicción de saltos se basa en el uso de una tabla de historia de saltos con 2.048 entradas. Las simulaciones ejecutadas por los diseñadores del PowerPC muestran que la tasa de aciertos en la predicción de saltos es del 90 % [THOM94].

### 13.5. MIPS R10000

El MIPS R10000, que ha evolucionado a partir del MIPS R4000 y usa el mismo repertorio de instrucciones, es una implementación bastante elegante y sencilla de los principios de diseño superescalar. La Figura 13.15 muestra la estructura global.

La primera etapa en el procesamiento de instrucciones del R10000 es una etapa de predecodificación de instrucciones. La predecodificación es una función que se encuentra en varias máquinas superescalares, incluidos el PowerPC 620 y el UltraSparc. La predecodificación es una ayuda para satisfacer las exigencias de los cauces de ejecución en paralelo. En cualquier máquina superescalar, el camino decodificación-emisión, que incluye un examen de las dependencias y los saltos, es crítico para lograr una gran capacidad de ejecución de instrucciones, porque las instrucciones introducidas por este camino abastecen a múltiples unidades de ejecución. En consecuencia, este camino representa un cuello de botella potencial. Para agilizar el camino decodificación-captación, algunos procesadores escalares llevan a cabo una decodificación parcial de instrucciones, al cargarlas desde una cache secundaria externa a la cache de instrucciones interna al chip. La predecodificación clasifica las instrucciones entrantes

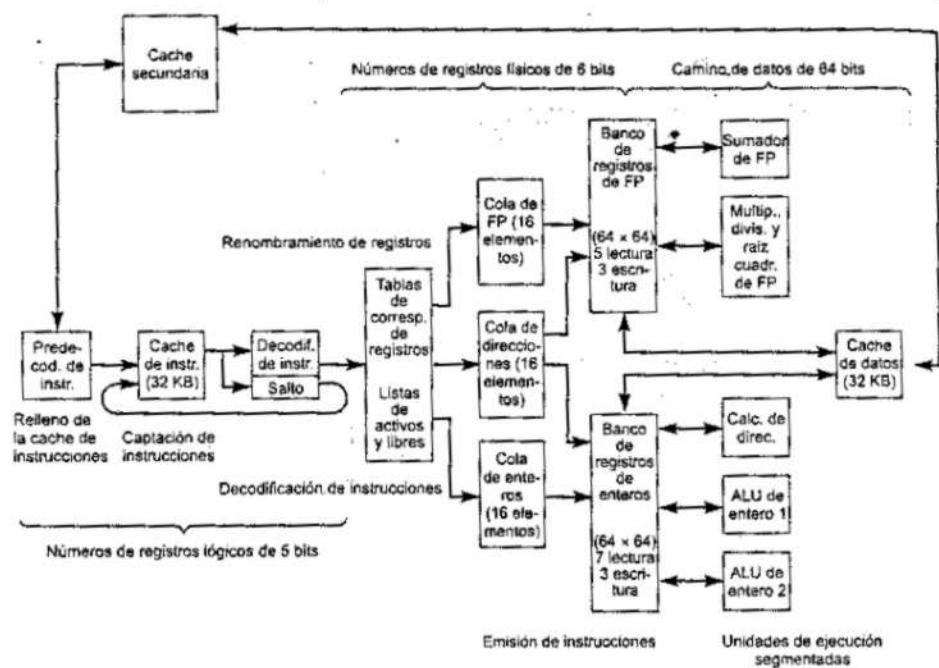


Figura 13.15. Estructura del MIPS R10000.

tes, y añade varios bits de decodificación a cada instrucción para simplificar el procesamiento posterior. En el caso del R10000, se añaden cuatro bits de decodificación a cada instrucción para indicar la unidad de ejecución a la cual será enviada posteriormente.

El R10000 capta cuatro instrucciones al mismo tiempo desde la cache secundaria, las predice y las coloca en la cache de instrucciones. Como en otras máquinas superescalares, el número de instrucciones captadas está pensado para que, como mínimo, iguale la máxima velocidad de ejecución y decodificación. Después, el R10000 decodifica hasta cuatro instrucciones en paralelo. Si aparece un salto incondicional, el destino del salto se proporciona a la cache de instrucciones para que ajuste el puntero de cara a la próxima captación. Si se encuentra una bifurcación condicional, la etapa de decodificación efectúa una predicción de salto usando un esquema de historia de 2 bits; se cargan a continuación las instrucciones de la ramificación predicha.

La siguiente etapa consiste en un renombramiento de registros, que convierte los números de dirección lógica de 5 bits (Figura 12.8) en números de registro físico de 6 bits. La etapa de decodificación guarda la correspondencia de direcciones lógicas a físicas, y una lista de registros en uso y disponibles. El proceso de renombramiento elimina las falsas dependencias, dejando sólo las dependencias de datos verdaderas.

Tras la decodificación, el R10000 coloca cada instrucción en una de las tres colas de instrucciones. Cada cola puede contener hasta 16 instrucciones. Cuando una instrucción se

envía, un bit de reserva se pone a ocupado para cada registro destino físico que se nombre en la instrucción. En las instrucciones de enteros y coma flotante, sus colas respectivas funcionan como centrales de reserva en lugar de las colas FIFO; las instrucciones pueden emitirse en cualquier orden compatible con las dependencias de datos, según lo indicado por los bits de reserva. Cuando todos los bits de reserva de los operandos fuente de una instrucción llegan a ponerse como no ocupados, la instrucción puede emitirse tan pronto como su unidad de ejecución esté lista. La cola de direcciones se utiliza para las instrucciones de carga/almacenamiento, y usa una política de emisión en orden.

Los bancos de registros de enteros y de coma flotante contienen 64 registros físicos cada uno. Las unidades de ejecución leen los operandos directamente desde los bancos de registros y escriben los resultados directamente. Cada banco dispone de múltiples puertos de lectura y escritura para permitir el acceso paralelo.

Hay cinco unidades de ejecución: una calculadora de direcciones, dos ALU para enteros, un sumador de coma flotante, y una unidad de coma flotante para multiplicar, dividir, y efectuar la raíz cuadrada. Las dos ALU para enteros pueden hacer sumas, restas y operaciones lógicas. La ALU 1 realiza desplazamientos y operaciones de condiciones de salto. La ALU 2 efectúa además las operaciones de multiplicación y división de enteros.

**13.6. ULTRASPARC II**

El UltraSPARC II es una máquina superescalar derivada del procesador SPARC; comparte la misma arquitectura del repertorio de instrucciones y las mismas características RISC, tales como el empleo de un gran número de registros y de una ventana de registros, descritos en la Sección 12.7.

## ORGANIZACIÓN INTERNA

La Figura 13.16 muestra la estructura interna de una versión reciente del procesador, conocida como el UltraSPARC III. Una cache L2 externa abastece a dos caches L1 separadas para instrucciones y datos. La unidad de precapitación y envío («prefetch and dispatch units», PDU) capta instrucciones en un buffer que puede contener hasta 12 instrucciones. La PDU es, además, responsable de la predicción de saltos, utilizando para ello una tabla de historia de saltos con un esquema de historia de saltos de 2 bits. Por último, la PDU incluye un módulo lógico de agrupamiento, que intenta organizar las instrucciones entrantes en grupos de hasta cuatro instrucciones para su envío simultáneo. De cada grupo, dos instrucciones pueden ser enteras, y dos de coma flotante/gráficos.

La unidad de ejecución de enteros contiene dos ALU completas y ocho ventanas de registros. Dos instrucciones con enteros se pueden ejecutar en paralelo. La unidad de coma flotante contiene dos ALU de coma flotante y una unidad de gráficos. Esto permite la ejecución de dos instrucciones de coma flotante, o una instrucción de coma flotante y una de gráficos, en paralelo. La unidad de gráficos da soporte al repertorio de instrucciones visuales («visual instruction set», VIS), una ampliación del repertorio de instrucciones del SPARC. El VIS es un repertorio de instrucciones especializadas para procesamiento de gráficos [TREM96], parecido al repertorio de instrucciones MMX del Pentium II.

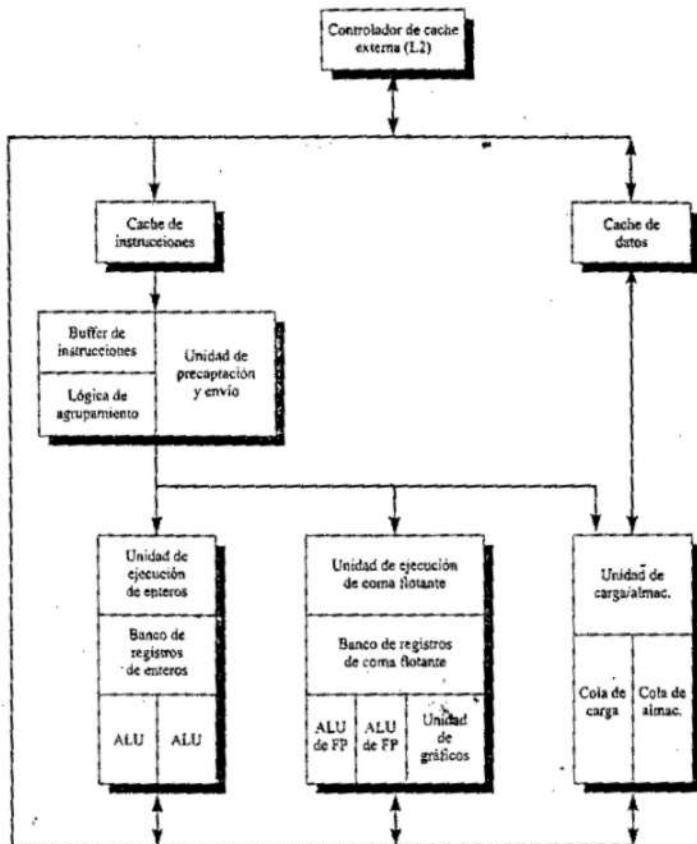


Figura 13.16. Diagrama de bloques del UltraSPARC III.

### CAUCE SEGMENTADO

El UltraSPARC II emplea un cauce de instrucciones de nueve etapas (Figura 13.17), que se divide en dos partes: una para las instrucciones de enteros y otra para las de coma flotante/gráficos. Para una instrucción entera, se utilizan las siguientes etapas:

- **Captación:** Se captan simultáneamente hasta cuatro instrucciones de la cache de instrucciones. En esta etapa también se efectúa la predicción de saltos.
- **Decodificación:** Esta etapa decodifica instrucciones y las coloca en el buffer de instrucciones.
- **Agrupamiento:** Esta etapa selecciona hasta cuatro instrucciones del buffer de instrucciones para su envío simultáneo. De este grupo, dos instrucciones pueden ser de enteros, y dos de coma flotante/gráficos.

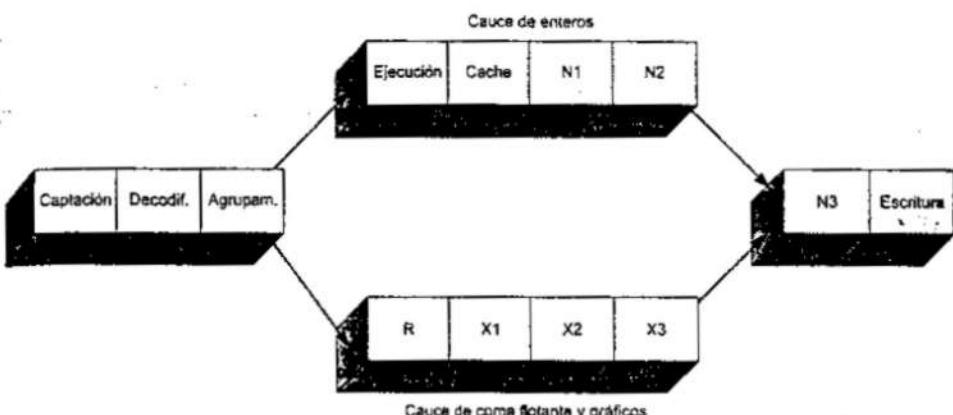


Figura 13.17. Cauce de instrucciones del UltraSPARC II.

- Ejecución: Las dos ALU de enteros acceden al banco de registros de enteros y ejecutan instrucciones de enteros. Esta etapa también calcula la dirección virtual de las instrucciones de carga/almacenamiento.
- Cache: Esta etapa determina si la dirección virtual de la etapa anterior conduce a un acierto o a un fallo de cache. Las operaciones de la ALU de la etapa de ejecución, generan códigos de condición en esta etapa, que se envían a las etapas de precaptación y envío para comprobar las predicciones de saltos previas. Si hay un error en la predicción, las instrucciones anteriores que haya en el cauce se eliminan.
- N1: Si ha ocurrido un fallo de cache, la instrucción entra en la cola de carga o almacenamiento para acceder a la cache L2 y a la memoria principal.
- N2: Se trata sencillamente de una etapa de espera al cauce de coma flotante.
- N3: En esta etapa se resuelven las interceptaciones.
- Escritura: En esta etapa, todos los resultados se escriben en el banco de registros.

El cauce de instrucciones de coma flotante comparte las tres primeras y las dos últimas etapas del cauce de enteros. Las tres etapas exclusivas del cauce de coma flotante son las siguientes:

- R: Durante esta etapa, las instrucciones de coma flotante y de gráficos se terminan de decodificar, y se accede al banco de registros.
- X1: En esta etapa, las instrucciones de coma flotante y de gráficos inician su ejecución.
- X2: Durante esta etapa continúa la ejecución.
- X3: En esta etapa se completa la ejecución.

### 13.7. IA-64/MERCEDES

Con el Pentium II y el Pentium III, la familia de microprocesadores que comenzó con el 8086, y que ha sido la línea de productos informáticos de mayor éxito de todos los tiempos,

parece haber llegado a su fin. Intel se ha asociado con Hewlett-Packard (HP) para desarrollar una nueva arquitectura de 64 bits, llamada IA-64. La arquitectura IA-64 no es una ampliación a 64 bits de la arquitectura ×86 de 32 bits de Intel, ni una adaptación de la arquitectura PA-RISC de 64 bits de Hewlett-Packard. En lugar de eso, la IA-64 es una nueva arquitectura, edificada sobre años de investigación en las dos compañías y en algunas universidades. La arquitectura aprovecha la enorme circuitería y la gran velocidad disponibles en las próximas generaciones de microchips gracias a la utilización sistemática del paralelismo.

Los conceptos básicos en los que se fundamenta la arquitectura IA-64 son los siguientes:

- Paralelismo a nivel de instrucciones, que es explícito en las instrucciones máquina, en lugar de depender del procesador en tiempo de ejecución.
- Palabras de instrucción grandes o muy grandes («long instruction word», LIW; o «very long instruction word», VLIW).
- Ejecución de saltos basada en predicados (no es lo mismo que predicción de saltos).
- Carga especulativa.

Intel y HP hacen referencia a esta combinación de conceptos con el nombre de «computación con instrucciones explícitamente paralelas» (explicitly parallel instruction computing, EPIC). Intel y HP utilizan el término EPIC para referirse a la tecnología, o conjunto de técnicas. La arquitectura IA-64 consiste en un repertorio de instrucciones real, destinado a ser implementado usando la tecnología EPIC. El primer producto de Intel basado en el IA-64 tiene el nombre clave de Merced. Le seguirán otros productos basados en la misma arquitectura IA-64.

La Tabla 13.2 resume las principales diferencias entre la arquitectura IA-64 y la aproximación superescalar tradicional. Estas diferencias se discuten en el resto de la sección.

## MOTIVACIÓN

Para Intel, mudarse a una nueva arquitectura, con un hardware incompatible con la arquitectura de instrucciones ×86, supone una decisión muy crítica. Pero está impulsada por los dictados de la tecnología. Cuando comenzó la familia ×86, allá a finales de los años setenta, el procesador tenía de decenas a miles de transistores, y era un dispositivo esencialmente esca-

Tabla 13.2. Arquitectura superescalar tradicional frente a IA-64

| Superescalar                                                                                            | IA-64                                                                                                                 |
|---------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------|
| Instrucciones de tipo RISC, una por palabra                                                             | Instrucciones de tipo RISC puestas en grupos de tres                                                                  |
| Múltiples unidades de ejecución en paralelo                                                             | Múltiples unidades de ejecución en paralelo                                                                           |
| Reordena y optimiza el flujo de instrucciones en tiempo de ejecución                                    | Reordena y optimiza el flujo de instrucciones en tiempo de compilación                                                |
| Predicción de saltos con ejecución especulativa de un camino                                            | Ejecución especulativa de los dos caminos de una bifurcación                                                          |
| Carga datos desde memoria sólo cuando es necesario, e intenta encontrar los datos primero en las cachés | Carga datos especulativamente antes de que se necesiten, y sigue intentando encontrar los datos primero en las cachés |

lar. Es decir, se procesaba una instrucción en cada instante, con poca o ninguna segmentación. Cuando el número de transistores creció a cientos de miles a mediados de los 80, Intel introdujo la segmentación (como ejemplo, véase Figura 11.18). Mientras tanto, otros fabricantes intentaban sacar partido al creciente número de transistores, aumentando la velocidad por medio del enfoque RISC, que permitía una segmentación más eficiente y, más tarde, por medio de la combinación superescalar/RISC, que traía consigo múltiples unidades de ejecución. Con el Pentium, Intel hizo un intento moderado de utilizar técnicas superescalares, permitiendo que dos instrucciones CISC se ejecutaran al mismo tiempo. Después, el Pentium Pro y el Pentium II incorporaron una traducción de instrucciones CISC a microoperaciones de tipo RISC, y una utilización más agresiva de técnicas superescalares. Este enfoque permitió la utilización eficaz de un chip con millones de transistores. Pero, para el procesador de la siguiente generación, el que viene después de los Pentium II y III, Intel y otros fabricantes se enfrentan a la necesidad de utilizar eficazmente decenas de millones de transistores en un único chip procesador.

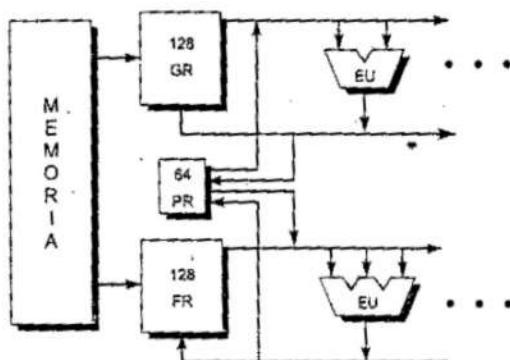
Los diseñadores de procesadores tienen pocas alternativas a la hora de usar esta abundancia de transistores. Una posibilidad es poner esos transistores adicionales en mayores cachés internas. Ello puede mejorar las prestaciones hasta cierto punto, pero llega un momento en el que aumentar la cache se traduce en una mejora insignificante de la tasa de aciertos. Otra alternativa es incrementar el grado de paralelismo superescalar, añadiendo más unidades de ejecución. El problema de esta solución es que los diseñadores están tropezando, de hecho, con un muro de complejidad. Cuantas más y más unidades de ejecución se añaden, haciendo el procesador más «ancho», se necesita más lógica para organizarlas. Ha de mejorarse la predicción de saltos, hay que usar procesamiento sin orden y hay que emplear cauces más largos. La ejecución desordenada requiere un gran número de registros de renombramiento y una compleja circuitería para acabar con las dependencias. Por consiguiente, los mejores procesadores de hoy día pueden retirar como mucho cuatro instrucciones por ciclo, y por lo general menos.

Para solucionar estos problemas, Intel y HP han propuesto un planteamiento de diseño global, que permite la utilización eficaz de un procesador con muchas unidades de ejecución en paralelo. El corazón de este nuevo enfoque es el concepto de paralelismo explícito. En esta aproximación, el compilador planifica estáticamente las instrucciones en tiempo de compilación, en lugar de que lo haga dinámicamente el procesador en tiempo de ejecución. El compilador determina qué instrucciones pueden ejecutarse en paralelo, e incluye esta información en la instrucción máquina. El procesador usa esa información para llevar a cabo la ejecución paralela. Una ventaja de esta aproximación es que el procesador no requiere tanta circuitería compleja como un procesador superescalar capaz de ejecutar instrucciones sin orden. Además, mientras que el procesador tiene que determinar la posibilidad de una potencial ejecución en paralelo en cuestión de nanosegundos, el compilador dispone de un plazo varios órdenes de magnitud mayor para examinar el código con detenimiento, y ve el programa como un todo.

## ORGANIZACIÓN

Como cualquier arquitectura de procesador, la IA-64 puede implementarse con diversas organizaciones. La Figura 13.18 indica, en términos generales, la organización de una máquina IA-64. Sus características más importantes son las siguientes:

- Un gran número de registros. El formato de instrucción de la arquitectura IA-64 supone el empleo de 256 registros de 64 bits, 128 de uso general para datos enteros y lógicos y 128 para uso con coma flotante y gráficos. También hay 64 registros predicado de un bit, usados para la ejecución con predicados como se explica más tarde.



GR = Registro de uso general o de enteros

FR = Registro de coma flotante o de gráficos

PR = Registro predicado de un bit

EU = Unidad de ejecución

Figura 13.18. Organización general de la arquitectura IA-64.

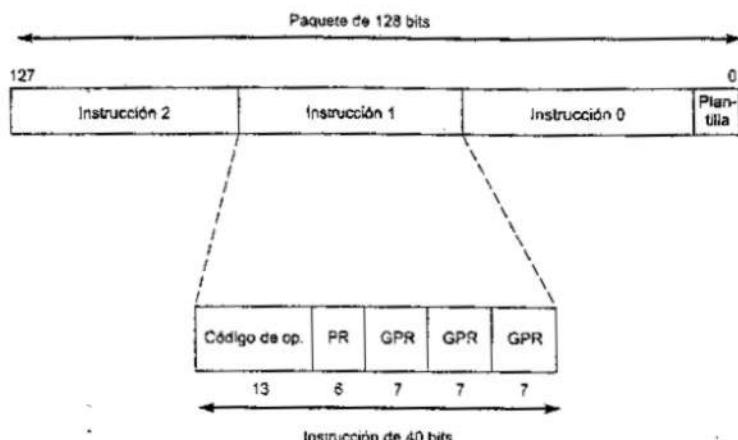
- Múltiples unidades de ejecución. Una máquina superescalar comercial típica puede soportar cuatro cauces paralelos, empleando cuatro unidades de ejecución en paralelo para las dos partes del procesador: enteros y coma flotante. Se espera que la IA-64 se implemente en sistemas con ocho o más unidades paralelas.

El banco de registros es bastante grande, comparado con la mayoría de las máquinas RISC y superescalares. La razón de ello es que se necesita un número de registros grande para permitir un alto grado de paralelismo. En una máquina superescalar tradicional, el lenguaje máquina (y el lenguaje ensamblador) emplean un pequeño número de registros visibles, y el procesador establece una correspondencia con número mayor de registros, usando técnicas de renombramiento y análisis de dependencias. Dado que deseamos tener paralelismo explícito y liberar al procesador de la carga que supone el renombramiento de registros y el análisis de dependencias, necesitamos un gran número de registros explícitos.

El número de unidades de ejecución depende del número de transistores disponibles en una implementación concreta. El procesador explotará el paralelismo hasta el punto que pueda. Por ejemplo, si el flujo de instrucciones en lenguaje máquina indica que se pueden ejecutar ocho instrucciones de enteros en paralelo, un procesador con cuatro cauces de enteros las ejecutará en dos turnos. Un procesador con ocho cauces ejecutará las ocho instrucciones simultáneamente.

### FORMATO DE INSTRUCCIÓN

La arquitectura IA-64 define un paquete de 128 bits, que contiene tres instrucciones y un campo plantilla (Figura 13.19). El procesador puede captar un paquete de instrucciones a la vez, así que cada captación trae tres instrucciones. El campo plantilla contiene información que indica qué instrucciones se pueden ejecutar en paralelo. La interpretación de este campo no se limita a un único paquete. En vez de eso, el procesador puede examinar varios paquetes



PR = Registro predicado

GPR = Registro de uso general (enteros o coma flotante)

Figura 13.19. Formato de instrucción de la arquitectura IA-64.

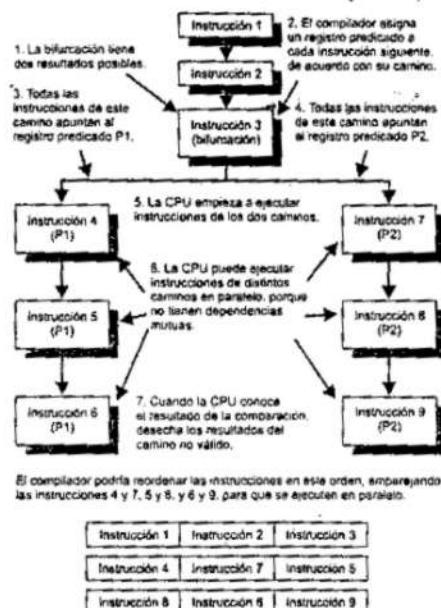
para determinar qué instrucciones pueden ejecutarse en paralelo. Por ejemplo, el flujo de instrucciones puede ser de tal manera, que permita la ejecución en paralelo de ocho instrucciones. El compilador reordenará las instrucciones de manera que estas ocho abarquen paquetes contiguos, y ajustará los bits de la plantilla de forma que el procesador sepa que las ocho instrucciones son independientes.

Las instrucciones agrupadas no tienen que estar en el orden original del programa. Además, debido a la flexibilidad del campo plantilla, el compilador puede mezclar instrucciones dependientes e independientes en el mismo paquete. Al contrario que algunos diseños VLIW previos, la arquitectura IA-64 no necesita insertar instrucciones de no operación (NOP) para completar los paquetes.

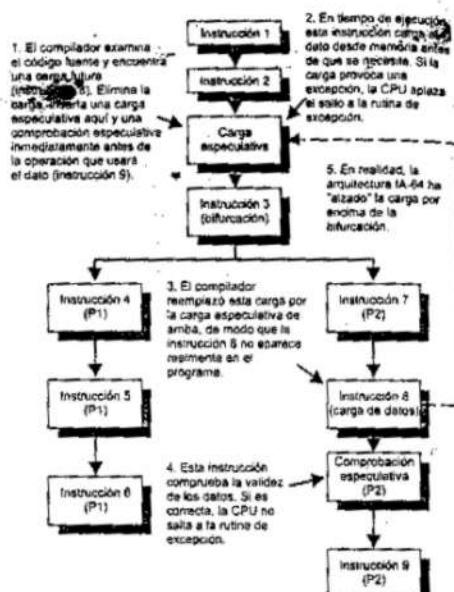
Cada instrucción tiene un formato fijo de 40 bits. Esta longitud es algo más larga que la tradicional de 32 bits de las máquinas RISC y RISC superescalares (aunque es mucho más corta que la microoperación de 118 bits del Pentium II). Hay dos factores que conducen a estos bits añadidos. En primer lugar, la arquitectura IA-64 emplea más registros que una máquina RISC típica: 128 para enteros y 128 para coma flotante. En segundo lugar, para dar cabida a la técnica de ejecución con predicados, una máquina IA-64 incluye 64 registros predicado. Estos últimos son invisibles para el programador, aunque el compilador inserta en las instrucciones una referencia a un registro predicado. Su uso se explica en la siguiente subsección.

## EJECUCIÓN CON PREDICADOS

Las dos innovaciones principales de la IA-64 son: la ejecución con predicados y la carga especulativa. La Figura 13.20, basada en una figura de [HALF97], ilustra las dos técnicas, que se discuten en esta subsección y en la siguiente.



(a) Ejecución con predicados



(b) Carga especulativa

Figura 13.20. Ejecución con predicados y carga especulativa en la arquitectura IA-64.

La ejecución con predicados es una técnica mediante la cual el procesador determina qué instrucciones pueden ejecutarse en paralelo. En este método, el compilador elimina saltos del programa usando ejecución condicional. Un ejemplo típico es la instrucción de un lenguaje de alto nivel `if-then-else`. Un compilador clásico inserta una bifurcación condicional en el punto del `if` de esta construcción. Si la condición tiene cierto resultado lógico, el salto no se produce, y se ejecuta el siguiente bloque de instrucciones, que representa el camino del `then`; al final de este camino hay un salto incondicional que evita el siguiente bloque, que representa el camino del `else`. Si la condición tiene el otro resultado lógico, se produce el salto sobre el bloque de instrucciones del `then`, y la ejecución prosigue por el bloque de instrucciones del `else`. Los dos flujos de instrucciones se juntan tras el final del bloque del `else`. Un compilador de IA-64, en lugar de eso, hace lo siguiente (Figura 13.20a):

- En el punto del programa del `if`, inserta una instrucción de comparación que crea dos predicados. Si la comparación es verdadera, el primer predicho se ajusta a verdadero y el segundo a falso; si la comparación es falsa, el primer predicho se ajusta a falso y el segundo a verdadero.
- Aumenta cada instrucción del camino del `then` con una referencia a un registro predicho que contiene el valor del primer predicho, y aumenta cada instrucción del camino del `else` con una referencia a un registro predicho que contiene el valor del segundo predicho.

3. El procesador ejecuta instrucciones de los dos caminos. Cuando el resultado de la comparación se conoce, el procesador desecha los resultados de un camino, y entrega los resultados del otro camino.

A modo de ejemplo, considere el siguiente código fuente:

```

if (a&&b)
 j = j + 1;
else
 if (c)
 k = k + 1;
 else
 k = k - 1;
 i = i + 1;

```

Código fuente:

Dos sentencias if seleccionan uno de tres posibles caminos de ejecución. Esto puede compilarse en el siguiente código en lenguaje ensamblador, que tiene tres bifurcaciones condicionales y una instrucción de salto incondicional:

```

beq a, 0, L1
beq b, 0, L1
add j, j, 1
jump L3
L1: beq c, 0, L2
add k, k, 1
jump L3
L2: sub k, k, 1
add i, i, 1

```

Código en ensamblador:

La Figura 13.21 muestra un diagrama de flujo de este código en ensamblador. Este diagrama divide el programa en lenguaje ensamblador, en bloques de código separados. El compilador puede asignar un predicado a cada bloque que se ejecuta condicionalmente. Los predicados se indican en la Figura 13.21. Suponiendo que a todos estos predicados se les haya asignado un valor inicial de falso, el código resultante queda como sigue:

|                        |                                                                                                                                                                                          |
|------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Código con predicados: | (1) P1, P2 = cmp (a == 0)<br>(2) <P2> P1, P3 = cmp (b == 0)<br>(3) <P3> add j, j, 1<br>(4) <P1> P4, P5 = cmp (c != 0)<br>(5) <P4> add k, k, 1<br>(6) <P5> sub k, k, 1<br>(7) add i, i, 1 |
|------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

Hemos introducido tres construcciones. Una instrucción de la forma

$\nwarrow$   
<PI> instrucción

va a ejecutarse sólo si el predicado PI es verdadero. En términos del formato de instrucción de la Figura 13.19, el campo PR tendrá el valor I para indicar el registro predicado de un bit I. El procesador captará, decodificará y empezará a ejecutar esta instrucción, pero sólo tomará la decisión de entregar el resultado o no, una vez que determine si el valor del registro predicado I es 1 (VERDADERO) ó 0 (FALSO).

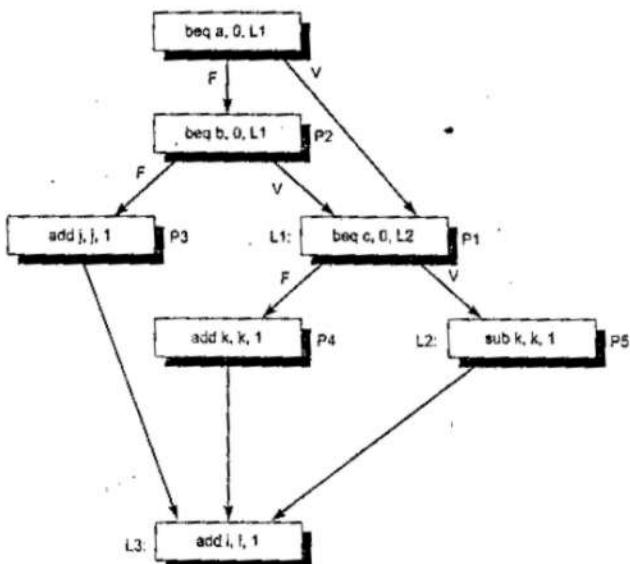


Figura 13.21. Ejemplo de código con predicados.

Una instrucción de la forma:

`PJ, PK = cmp (relación)`

ajustará el valor del predicado J a VERDADERO y K a FALSO si la relación es verdadera y ajustará el valor del predicado J a FALSO y K a VERDADERO si la relación es falsa. El formato de instrucción en este caso no será el que se muestra en la Figura 13.19, que es el de una instrucción registro a registro asociada a un predicado. En lugar de ello, esta instrucción tendrá dos campos de registro predicado. El procesador captará, decodificará y ejecutará esta instrucción, y ajustará el resultado en los registros predicado J y K.

Por último, una instrucción generadora de predicados puede también estar asociada a un predicado:

`<PI> PJ, PK = cmp (relación)`

Los registros predicado J y K se actualizarán según la comparación, si el predicado PI es verdadero. Si el predicado PI es falso, la instrucción no se ejecutará. Esta instrucción requiere tres campos de registro predicado en su formato.

Volviendo a nuestro programa, las dos primeras bifurcaciones condicionales en el código en ensamblador se traducen a dos instrucciones de comparación generadoras de predicados. Si la primera instrucción pone P2 a falso, la segunda instrucción no se ejecutará. P3 es verdadero sólo si la sentencia if externa del código fuente es verdadera. La parte del then de la sentencia if externa se asocia al predicado P3 por esta razón. La instrucción (4) del código con predicados decide si se ejecuta la instrucción de suma o la de resta de la parte del else

externo. Por último, el incremento de *i* se realiza incondicionalmente. Examinando el código fuente, y después el código con predicados, vemos que se va a ejecutar sólo una de las instrucciones (3), (5) y (6). En un procesador superescalar normal, usariamos predicción de saltos para estimar cuál de las tres va a ejecutarse, y seguir ese camino. Si el procesador hace una estimación equivocada, el cauce ha de vaciarse. Un procesador IA-64 puede iniciar la ejecución de estas tres instrucciones y, una vez que los valores de los registros predicho se separan, entregar solamente los resultados de la instrucción válida. Por tanto, utilizamos las unidades de ejecución en paralelo adicionales para evitar los retardos debidos al vaciado del cauce.

Gran parte de la investigación original sobre ejecución con predicados se hizo en la Universidad de Illinois. Los estudios de simulación de los investigadores indican que la utilización de predicados se traduce en una reducción sustancial de saltos dinámicos y fallos en la predicción de saltos, y en una mejora considerable de las prestaciones de procesadores con múltiples cauces paralelos (por ejemplo, [MAHL94] y [MAHL95]).

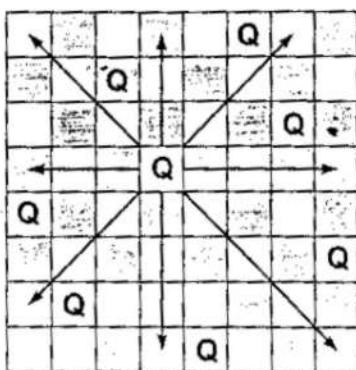
## CARGA ESPECULATIVA

Otra innovación importante de la arquitectura IA-64 es la carga especulativa. Permite al procesador cargar datos de memoria antes de que el programa los necesite, para evitar retardos de memoria. Además, el procesador aplaza el informe de excepciones hasta que sea necesario comunicar la excepción. Se usa el término *alzar* (*hoist*) para hacer referencia al movimiento de una instrucción de carga a un punto anterior en el flujo de instrucciones.

La minimización de los retardos provocados por las cargas es crucial para mejorar las prestaciones. Típicamente, al principio de un bloque de código, hay varias operaciones de carga que llevan datos desde la memoria a los registros. Como la memoria, aunque este aumentada con uno o dos niveles de cache, es lenta comparada con un procesador superescalar, los retardos para obtener datos de la memoria se convierten en un cuello de botella. Para minimizarlo, nos gustaría reordenar el código, de forma que las cargas se hicieran tan pronto como fuera posible. Esto puede hacerse con cualquier compilador, hasta cierto punto. El problema se presenta cuando intentamos mover una carga de una parte a otra en un flujo de control. No se puede mover incondicionalmente la carga más arriba de una bifurcación, porque la carga puede no producirse realmente. Podríamos mover la carga condicionalmente usando predicados, de modo que los datos pudieran recuperarse de la memoria pero no entregarse a un registro de la arquitectura mientras no se conociera el resultado del predicado. El problema de esta estrategia es que la carga puede dilatarse. Podrá generarse una excepción, debida a una dirección no válida o una falta de página. Si esto ocurre, el procesador tendría que ocuparse de la excepción o la falta, causando un retraso.

¿Cómo podemos, entonces, mover la carga más arriba de una bifurcación? La solución indicada en la arquitectura IA-64 es la carga especulativa, que separa el funcionamiento de la carga (entregar un valor) de la excepción (Figura 13.20b). Una instrucción de carga del programa original se reemplaza por dos instrucciones:

- Una carga especulativa (*ld.s*) ejecuta la captación desde memoria, y lleva a cabo la detección de excepciones, pero no lanza la excepción (no llama a la rutina del SO que maneja la excepción). Esta instrucción *ld.s* se alza a un punto anterior del programa que sea adecuado.
- Una instrucción de comprobación (*check.s*) permanece en el lugar de la carga original y lanza las excepciones. Esta instrucción *check.s* puede asociarse a un predicado, de forma que sólo se ejecute si el predicado es verdadero.



Bucle:  
if ((b[j] == true) && (a[i+j] == true) && (c[i-j+7] == true))

Figura 13.22. El problema de las ocho reinas.

Si ld.s detecta una excepción, ajusta un bit de recuerdo asociado con el registro destino. Si la instrucción check.s correspondiente se ejecuta, y el bit de recuerdo está a uno, la instrucción check.s salta a una rutina de servicio de la excepción.

Un ejemplo utilizado por Intel y HP para probar los programas con predicados, y para ilustrar el uso de cargas especulativas, es el problema de las ocho reinas. El objetivo es colocar ocho reinas en un tablero de ajedrez de tal manera que ninguna reina amenace a otra. La Figura 13.22 muestra una solución. La línea principal del código fuente, en un bucle interno, es la siguiente:

```
if ((b[j] == true) && (a[i+j] == true) && (c[i-j+7] == true))
```

El programa completo mueve por columnas, colocando una reina en cada columna, de tal modo que no sea atacada por una reina colocada con anterioridad en su fila o en una de sus diagonales. La matriz B comprueba la fila, y las matrices A y C comprueban las dos diagonales.

Un programa en ensamblador simple incluye tres cargas y tres bifurcaciones:

|                        |                                   |
|------------------------|-----------------------------------|
| (1)                    | R1 = &b[j]                        |
| (2)                    | ld R2, (R1)                       |
| (3)                    | bne R2, 1, L2                     |
| (4)                    | R3 = &a(i + j)                    |
| (5)                    | ld R4, (R3)                       |
| Código en ensamblador: | bne R4, 1, L2                     |
| (6)                    | R5 = &c[i - j + 7]                |
| (7)                    | ld R6, (R5)                       |
| (8)                    | bne R6, 1, L2                     |
| (9)                    | L1: < código del camino del then> |
| (10)                   | L2: < código del camino del else> |

Con cargas especulativas y ejecución con predicados, queda lo siguiente:

Código con  
cargas especulativas  
y predicados:

```

(1) R1 = &b[j]
(2) R3 = &a[i + j]
(3) R5 = &c[i - j + 7]
(4) ld R2, (R1)
(5) ld.s R4, (R3)
(6) ld.s R6, (R5)
(7) P1, P2 = cmp (R2 == 1)
(8) <P2> br L2
(9) chk.s R4
(10) P3, P4 = cmp (R4 == 1)
(11) <P4> br L2
(12) chk.s R6
(13) P5, P6 = cmp (R5 == 1)
(14) <P6> br L2
(15) L1: < código del camino del then >
(16) L2: < código del camino del else >

```

El programa en ensamblador se compone de tres bloques básicos de código, cada uno de los cuales es una carga seguida de una bifurcación condicional. Las instrucciones de ajuste de dirección 4 y 7 del código en ensamblador, son cálculos aritméticos sencillos; pueden hacerse en cualquier momento, así que el compilador los coloca al principio. Después, el compilador afronta los tres bloques simples, cada uno de los cuales consiste en una carga, un cálculo de condición y una bifurcación condicional. Parece haber poca esperanza de hacer algo en paralelo aquí. Además, si suponemos que la carga necesita dos o más ciclos de reloj, perdemos algún tiempo antes de que la bifurcación condicional pueda ejecutarse. Lo que puede hacer el compilador es alzar la segunda y la tercera carga (instrucciones 5 y 8 en el código en ensamblador) por encima de todos los saltos. Esto se hace poniendo una carga especulativa al principio (instrucciones 5 y 6), y dejando una comprobación en el bloque de código original (instrucciones 9 y 12).

Esta transformación hace posible ejecutar las tres cargas en paralelo, y empezar pronto las cargas, a fin de minimizar o evitar los retrasos debido al tiempo de carga. El compilador puede ir más lejos con un uso más agresivo de los predicados, y eliminar dos de los tres saltos:

Código con  
cargas especulativas  
y predicados revisado:

```

(1) R1 = &b[j]
(2) R3 = &a[i + j]
(3) R5 = &c[i - j + 7]
(4) ld R2, (R1)
(5) ld.s R4, (R3)
(6) ld.s R6, (R5)
(7) P1, P2 = cmp (R2 == 1)
(8) <P1> chk.s R4
(9) <P1> P3, P4 = cmp (R4 == 1)
(10) <P3> chk.s R6
(11) <P3> P5, P6 = cmp (R5 == 1)
(12) <P6> br L2
(13) L1: < código del camino del then >
(14) L2: < código del camino del else >

```

Ya teníamos una comparación que generaba dos predicados. En el código revisado, en lugar de saltar en el predicado falso, el compilador habilita la ejecución de la comprobación y la

siguiente comparación en el predicado verdadero. Eliminar los dos saltos significa eliminar dos errores de predicción potenciales, de manera que el ahorro es mayor que sólo dos instrucciones.

### 13.8. LECTURAS Y SITIOS WEB RECOMENDADOS

[JOHN91] sigue siendo un libro apropiado y excelente sobre diseño superescalar. [SMIT9 y [SIMA97] son artículos generales sobre la materia dignos de consideración. [JOUP89] examina el paralelismo a nivel de instrucciones, examina varias técnicas para maximizar el paralelismo, y compara las aproximaciones superescalar y supersegmentada, usando simulaciones.

[POPE91] proporciona un estudio detallado de una máquina superescalar propuesta. También ofrece una excelente revisión de los temas de diseño relacionados con políticas de ejecución desordenada de instrucciones. En [KUGA91] se encuentra otro estudio de un sistema propuesto; este artículo plantea y estudia la mayoría de los temas de diseño importantes de las implementaciones superescalares. [LEE91] examina las técnicas software que se pueden usar para aumentar las prestaciones superescalares. [WALL91] es un interesante estudio del grado hasta el cual puede explotarse el paralelismo a nivel de instrucciones en un procesador superescalar.

[SHAN98] contiene una descripción muy detallada de la segmentación de instrucciones del Pentium II. En [PAPW96] se encuentra una buena discusión sobre diseño, que cubre el Pentium Pro y analiza algunos de los compromisos de diseño de la organización del procesador. El Pentium Pro tiene la misma organización superescalar que los Pentium II y III, pero sin la componente MMX.

[POTT94] incluye una revisión detallada de la segmentación de instrucciones en el PowerPC 601. [SHAN95] también ofrece una buena cobertura.

El mejor informe sobre el MIPS R10000 está en [YEAG96]. En [NORM98] se encuentra una buena discusión sobre el UltraSPARC II. [DUL098] ofrece una visión de conjunto de la arquitectura IA-64, y [HWU98] aporta una breve introducción a la ejecución con predicados.

DUL098 Dulong, C. «The IA-64 Architecture at Work.» *Computer*, July, 1998.

HWU98 Hwu, W. «Introduction to Predicated Execution.» *Computer*, January, 1998.

JOHN91 Johnson, M. *Superscalar Microprocessor Design*. Englewood Cliffs, NJ: Prentice Hall, 1991.

JOUP89a Jouppi, N., y Wall, D. «Available Instruction-Level Parallelism for Superscalar and Superpipelined Machines.» *Proceedings, Third International Conference on Architectural Support for Programming Languages and Operating Systems*, April, 1989.

KUGA91 Kuga, M.; Murakami, K.; y Tomita, S. «DSNS (Dynamically-Hazard Resolved, Static Code-Scheduled, Nonuniform Superscalar): Yet Another Superscalar Processor Architecture.» *Computer Architecture News*, June, 1991.

LEE91 Lee, R.; Kwok, A.; y Briggs, F. «The Floating Point Performance of a Superscalar SPARC Processor.» *Proceedings, Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, April, 1991.

NORM98 Normoyle, K., et al. «UltraSPARC-III: Expanding the Boundaries of a System on a Chip.» *IEEE Micro*, March/April, 1998.

- PAPW96 Papworth, D. «Tuning the Pentium Pro Microarchitecture.» *IEEE Micro*, April, 1996.
- POPE91 Popescu, V., et al. The Metalfow Architecture.» *IEEE Micro*, June, 1991.
- POTT94 Potter, T., et al. «Resolution of Data and Control-Flow Dependencies in the PowerPC 601.» *IEEE Micro*, October, 1994.
- SHAN95 Shanley, T. *PowerPC System Architecture*. Reading, MA: Addison-Wesley, 1995.
- SHAN98 Shanley, T. *Pentium Pro and Pentium II System Architecture*. Reading, MA: Addison-Wesley, 1998.
- SIMA97 Sima, D. «Superscalar Instruction Issue.» *IEEE Micro*, September/October, 1997.
- SMIT95 Smith, J., y Sohi, G. «The Microarchitecture of Superscalar Processors.» *Proceedings of the IEEE*, December, 1995.
- WALL91 Wall, D. «Limits of Instruction-Level Parallelism.» *Proceedings, Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, April, 1991.
- YEAG96 Yeager, K. «The MIPS R10000 Superscalar Microprocessor.» *IEEE Micro*, April, 1996.



#### SITIOS WEB RECOMENDADOS:

- Merced/IA-64: sitio de Intel con la última información sobre el IA-64 y el procesador Merced.
- IMPACT: éste es un sitio de la Universidad de Illinois, donde se ha hecho gran parte de la investigación sobre ejecución con predicados. Hay varios artículos disponibles sobre la materia.

#### 13.9. PROBLEMAS

- 13.1. Cuando se usa finalización desordenada en un procesador superescalar, la reanudación de la ejecución después del procesamiento de una interrupción es complicada, porque la condición de excepción puede haberse detectado en una instrucción que produjo su resultado fuera de orden. El programa no puede reanudarse en la instrucción siguiente a la instrucción de la excepción, porque las siguientes instrucciones ya han finalizado, y hacerlo así provocaría que esas instrucciones se ejecutaran dos veces. Sugiera un mecanismo, o varios, para tratar esta situación.
- 13.2. Considere la siguiente secuencia de instrucciones, donde la sintaxis consta de un código de operación seguido de un registro destino, seguido a su vez por uno o dos registros fuente:

|   |      |            |
|---|------|------------|
| 0 | ADD  | R3, R1, R2 |
| 1 | LOAD | R6, [R3]   |
| 2 | AND  | R7, R5, 3  |
| 3 | ADD  | R1, R6, R0 |

|    |      |             |
|----|------|-------------|
| 4  | SRL  | R7, R0, 8   |
| 5  | OR   | R2, R4, R7  |
| 6  | SUB  | R5, R3, R4  |
| 7  | ADD  | R0, R1, R10 |
| 8  | LOAD | R6, [R5]    |
| 9  | SUB  | R2, R1, R6  |
| 10 | AND  | R3, R7, 15  |

Suponga el uso de un cauce de cuatro etapas: captación, decodificación/emisión, ejecución y escritura. Suponga que todas las etapas del cauce tardan un ciclo de reloj excepto la etapa de ejecución. En las instrucciones aritméticas y lógicas con enteros sencillas, la etapa de ejecución necesita un ciclo, pero un LOAD desde memoria consume cinco ciclos en la etapa de ejecución.

Si tenemos un cauce escalar sencillo, pero que permite ejecución desordenada, podemos construir la siguiente tabla para la ejecución de las siete primeras instrucciones:

| Instrucción | Captación | Decodificación | Ejecución | Escritura |
|-------------|-----------|----------------|-----------|-----------|
| 0           | 0         | 1              | 2         | 3         |
| 1           | 1         | 2              | 4         | 9         |
| 2           | 2         | 3              | 5         | 6         |
| 3           | 3         | 4              | 10        | 11        |
| 4           | 4         | 5              | 6         | 7         |
| 5           | 5         | 6              | 8         | 10        |
| 6           | 6         | 7              | 9         | 12        |

Los elementos bajo las cuatro etapas del cauce, indican el ciclo de reloj en el que cada instrucción inicia cada fase. En este programa, la segunda instrucción ADD (instrucción 3) depende de la instrucción LOAD (instrucción 1) en uno de sus operandos, r6. Como la instrucción LOAD tarda cinco ciclos de reloj, y la lógica de emisión encuentra la instrucción dependiente ADD después de dos ciclos, la lógica de emisión tiene que retrasar la instrucción ADD tres ciclos de reloj. Con una capacidad de ejecución desordenada, el procesador puede detener la instrucción 3 en el ciclo de reloj 4, y pasar a emitir las siguientes tres instrucciones independientes, que entran en ejecución en los ciclos 6, 8 y 9. La instrucción LOAD termina su ejecución en el ciclo 9, y entonces la instrucción dependiente ADD puede comenzar su ejecución en el ciclo 10.

- a) Complete la tabla anterior.
  - b) Rehaga la tabla suponiendo que no se tiene la capacidad de ejecución desordenada. ¿Cuál es el ahorro usando esa capacidad?
  - c) Rehaga la tabla suponiendo una implementación superescalar que pueda manejar dos instrucciones a la vez en cada etapa.
- 13.3. En la cola de instrucciones de la unidad de envío del PowerPC 601, las instrucciones pueden enviarse desordenadamente a las unidades de procesamiento de saltos y de coma flotante, pero las instrucciones dirigidas a la unidad de enteros tienen que ser enviadas sólo desde el elemento inferior de la cola. ¿Por qué existe esta limitación?
- 13.4. Produzca una figura similar a la Figura 13.14 para los siguientes casos:
- a) Predicción de salto: saltar; predicción correcta: se produce el salto.
  - b) Predicción de salto: saltar; predicción incorrecta: el salto no se produce.

- 13.5. Considere el siguiente programa en lenguaje ensamblador:

|                    |                                |
|--------------------|--------------------------------|
| I1: Move R3, R7    | /R3 $\leftarrow$ (R7)/         |
| I2: Load R8, (R3)  | /R8 $\leftarrow$ Memoria (R3)/ |
| I3: Add R3, R3, 4  | /R3 $\leftarrow$ (R3) + 4/     |
| I4: Load R9, (R3)  | /R9 $\leftarrow$ Memoria (R3)/ |
| I5: BLE R8, R9, L3 | /Bifurcar si (R9) > (R8)/      |

Este programa incluye dependencias escritura-escritura, lectura-escritura, y escritura-lectura. Muéstrelas.

- 13.6. La Figura 13.23 muestra un ejemplo de organización superescalar de un procesador. El procesador puede emitir dos instrucciones por ciclo si no hay conflicto por los recursos y ningún problema de dependencias de datos. Hay básicamente dos cauces, con cuatro etapas de procesamiento (captación, decodificación, ejecución y almacenamiento). Cada cauce tiene su propia unidad de captación, decodificación y almacenamiento. Hay disponibles cuatro unidades funcionales (multiplicador, sumador, unidad lógica y unidad de carga) para la etapa de ejecución, que son compartidas por los dos cauces de forma dinámica. Los dos cauces pueden usar dinámicamente las dos unidades de almacenamiento, dependiendo de su disponibilidad en un ciclo concreto. Hay una ventana de anticipación para la emisión desordenada de instrucciones.

Considere el siguiente programa que va a ejecutarse en este procesador:

|                |                               |
|----------------|-------------------------------|
| I1: Load R1, A | /R1 $\leftarrow$ Memoria (A)/ |
| I2: Add R2, R1 | /R2 $\leftarrow$ (R2) + (R1)/ |
| I3: Add R3, R4 | /R3 $\leftarrow$ (R3) + (R4)/ |
| I4: Mul R4, R5 | /R4 $\leftarrow$ (R4) + (R5)/ |
| I5: Comp R6    | /R6 $\leftarrow$ (R6)/        |
| I6: Mul R6, R7 | /R3 $\leftarrow$ (R3) + (R4)/ |

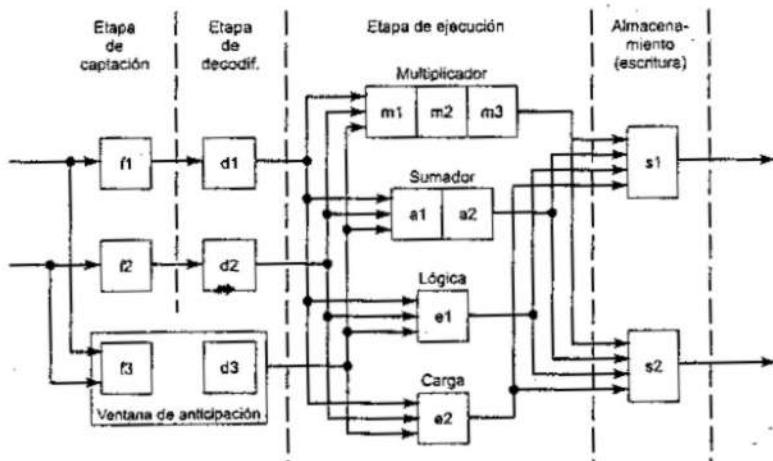


Figura 13.23. Un procesador superescalar con un cauce doble.

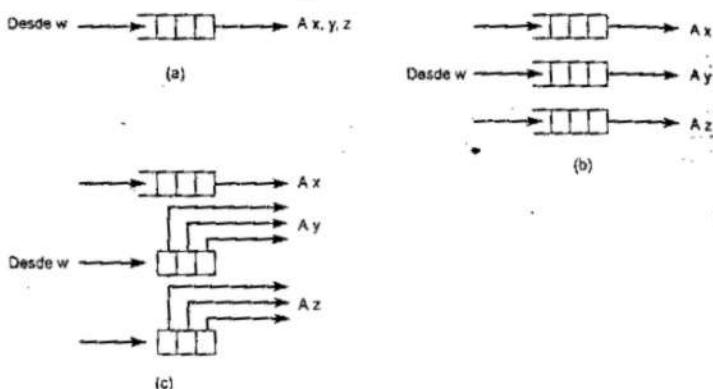


Figura 13.24. Figura del Problema 13.7.

- a) ¿Qué dependencias hay en este programa?
- b) Muestre la actividad del cauce para este programa en el procesador de la Figura 13.23, con políticas de emisión en orden y finalización en orden, usando una presentación similar a la de la Figura 13.2.
- c) Repita el apartado anterior para emisión en orden y finalización desordenada.
- d) Repita el apartado anterior para emisión desordenada y finalización desordenada.
- 13.7. La Figura 13.24 está tomada de un artículo sobre diseño superescalar. Explique las tres partes de la figura, y defina w, x, y y z.
- 13.8. En la Sección 13.7, introdujimos las siguientes construcciones para la ejecución con predicados:

$PJ, PK = \text{cmp (relación)}$   
 $\langle PI \rangle PJ, PK = \text{cmp (relación)}$

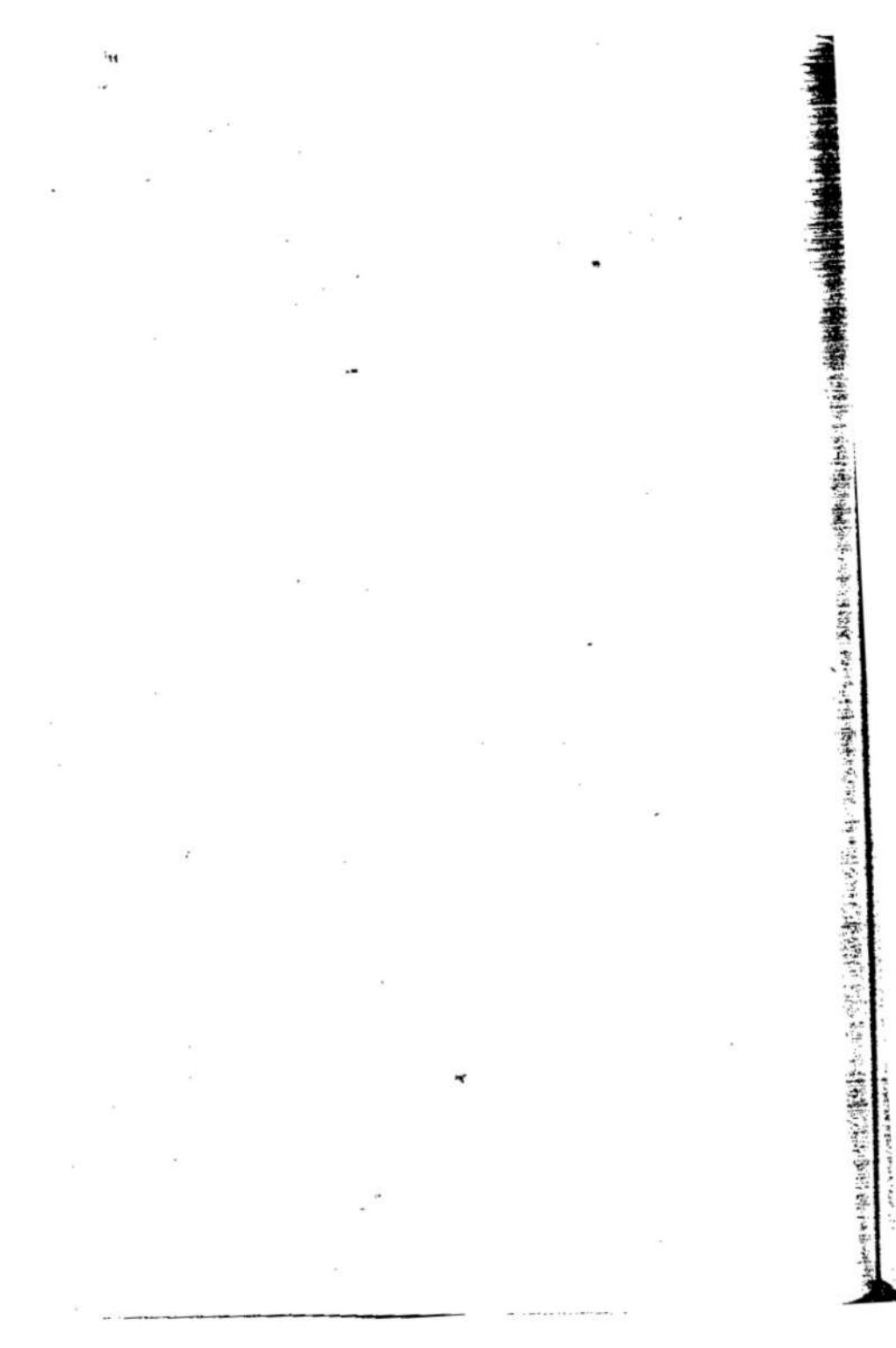
Complete la siguiente tabla de verdad:

| PI          | Comparación | PJ | PK |
|-------------|-------------|----|----|
| No presente | 0           |    |    |
| No presente | 1           |    |    |
| 0           | 0           |    |    |
| 0           | 1           |    |    |
| 1           | 0           |    |    |
| 1           | 1           |    |    |

- 13.9 Para el programa con predicados de la Sección 13.7, que implementa el diagrama de flujo de la Figura 13.21, indique:
- Las instrucciones que pueden ejecutarse en paralelo.
  - Las instrucciones que pueden agruparse en el mismo paquete de instrucciones de la arquitectura IA-64.
- 13.10. Considere el siguiente segmento de código fuente:

```
for (i = 0; i<100; i++)
 if (A[i]<50)
 j = j + 1;
 else
 k = k + 1;
```

- Escriba el correspondiente segmento de código en ensamblador.
- Vuelva a escribir el segmento de código en ensamblador, usando técnicas de ejecución con predicados.



## PARTE IV

# LA UNIDAD DE CONTROL

### CUESTIONES A TRATAR EN LA CUARTA PARTE

En la Parte III, nos concentraremos en las instrucciones máquina y las operaciones que lleva a cabo el procesador para ejecutar cada instrucción. Lo que quedó fuera de la discusión es qué se hace exactamente para que tenga lugar cada operación individual. Esa es la tarea de la unidad de control.

La unidad de control es la parte del procesador que realmente hace que ocurra todo. La unidad de control emite señales de control externas al procesador, para producir el intercambio de datos con la memoria y los módulos de E/S. También emite señales de control internas al procesador, para transferir datos entre registros, hacer que la ALU ejecute una función concreta, y regular otras operaciones internas. La entrada a la unidad de control está compuesta por el registro de instrucción, los indicadores, y ciertas señales de control de fuentes externas (por ejemplo, señales de interrupción).

### ESQUEMA DE LA CUARTA PARTE

#### CAPÍTULO 14. FUNCIONAMIENTO DE LA UNIDAD DE CONTROL

El Capítulo 14 examina el funcionamiento de la unidad de control, explicando en términos funcionales lo que ésta hace. Se entiende que la responsabilidad fundamental de la unidad de control es hacer que se produzca una secuencia de operaciones elementales, llamadas *microoperaciones*, durante el desarrollo de un ciclo de instrucción.

#### CAPÍTULO 15. EL CONTROL MICROPROGRAMADO

En el Capítulo 15, veremos cómo el concepto de microoperación conduce a un método elegante y potente para implementar la unidad de control, conocido como «microprogramación». Fundamentalmente, se desarrolla un lenguaje de programación de un nivel más bajo.

Cada instrucción del lenguaje máquina del procesador se traduce a una secuencia de instrucciones de la unidad de control. Estas instrucciones de más bajo nivel se llaman «microinstrucciones», y el proceso de traducción es conocido como «micropogramación».

## CAPÍTULO 14

---

# Funcionamiento de la unidad de control

### 14.1. Microoperaciones

- El ciclo de captación
- El ciclo indirecto
- El ciclo de interrupción
- El ciclo de ejecución
- El ciclo de instrucción

### 14.2. Control del procesador

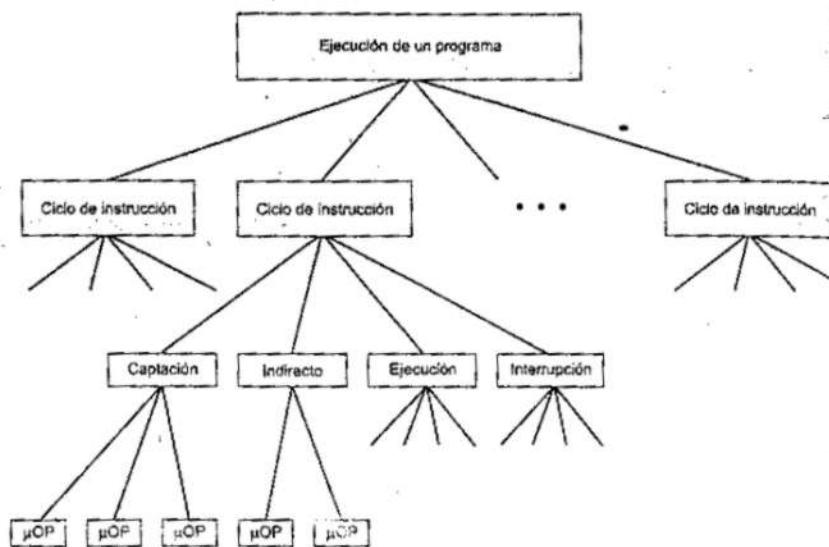
- Requisitos funcionales
- Señales de control
- Un ejemplo de señales de control
- Organización interna del procesador
- El Intel 8085

### 14.3. Implementación cableada

- Entradas de la unidad de control
- Lógica de la unidad de control

### 14.4. Lecturas recomendadas

### 14.5. Problemas



- La ejecución de una instrucción implica la ejecución de una secuencia de pasos más pequeños, normalmente llamados «ciclos». Por ejemplo, una ejecución puede de constar de ciclos de captación, acceso indirecto a memoria, ejecución e interrupción. Además, cada ciclo se compone de una serie de operaciones más elementales, llamadas «microoperaciones». Una única microoperación implica, por lo general, una transferencia entre registros, una transferencia entre un registro y un bus externo, o una sencilla operación de la ALU.
- La unidad de control de un procesador realiza dos tareas: (1) hace que el procesador ejecute las microoperaciones en la secuencia correcta, determinada por el programa que se está ejecutando, y (2) genera las señales de control que provocan la ejecución de cada microoperación.
- Las señales de control generadas por la unidad de control causan la apertura y el cierre de ciertas puertas lógicas, lo que da como resultado una transferencia de datos hacia, o desde, los registros, y una operación de la ALU.
- Una técnica para construir la unidad de control es la implementación cableada, en la cual ésta es un circuito combinacional. Sus señales lógicas de entrada, gobernadas por la instrucción máquina en curso, se transforman en un conjunto de señales de control de salida.

**E**n el Capítulo 9, señalamos que el repertorio de instrucciones máquina contribuye en gran medida a definir el procesador. Si conocemos el repertorio de instrucciones máquina, lo que incluye una comprensión del efecto de cada código de operación y de los modos de direccionamiento, y si conocemos el conjunto de registros visibles por el usuario, entonces conocemos las funciones que puede realizar el procesador. Ésta no es una descripción completa. Necesitamos conocer las interfaces externas, por lo general accesibles a través de un bus, y saber cómo se manejan las interrupciones. Siguiendo esta línea de razonamiento, surge la siguiente lista de conceptos, necesarios para especificar la funcionalidad de un procesador:

1. Operaciones (códigos de operación)
2. Modos de direccionamiento
3. Registros
4. Interfaz con el módulo de E/S
5. Interfaz con el módulo de memoria
6. Estructura del procesamiento de interrupciones

Esta lista, aunque general, es bastante completa. Los puntos del 1 al 3 quedan definidos por el repertorio de instrucciones. Los puntos 4 y 5 vienen determinados típicamente por el bus del sistema. El punto 6 está definido parcialmente por el bus del sistema, y parcialmente por el tipo de apoyo que ofrece el procesador al sistema operativo.

Los seis puntos de esta lista podrían denominarse «requisitos funcionales de un procesador». Ellos determinan lo que debe hacer el procesador. Nos ocupamos de esto en las Partes II y III. Ahora nos vamos a centrar en la cuestión de cómo se realizan esas funciones, o, más específicamente, cómo se controlan los diversos elementos del procesador para proporcionar esas funciones. Por tanto, vamos a estudiar la unidad de control, que controla el funcionamiento del procesador.

#### 14.1. MICROOPERACIONES

Hemos visto que el funcionamiento de un computador, cuando ejecuta un programa, consiste en una secuencia de ciclos de instrucción, con una instrucción máquina por ciclo. Naturalmente, debemos recordar que esta secuencia de ciclos de instrucción no es necesariamente la misma que la *secuencia escrita* de instrucciones que constituye un programa, debido a la existencia de instrucciones de salto. A lo que nos referimos aquí es a la *secuencia temporal* de ejecución de instrucciones.

Hemos visto, además, que cada ciclo de instrucción puede considerarse compuesto por varias unidades más pequeñas. Una subdivisión práctica es: captación, ciclo indirecto, ejecución, e interrupción, si bien sólo aparecen siempre, los ciclos de captación y de ejecución.

Para diseñar una unidad de control, sin embargo, necesitamos desglosar más esta descripción. En nuestra discusión sobre la segmentación en el Capítulo 11, comenzamos a ver que es posible una mayor descomposición. En realidad, veremos que cada uno de los ciclos más pequeños implica una serie de pasos, cada uno de los cuales involucra ciertos registros del procesador. Nos referiremos a estos pasos como *microoperaciones*. El prefijo *micro* se refiere al hecho de que cada paso es muy sencillo y hace muy poco. La Figura 14.1 representa la relación entre los distintos conceptos de los que hemos hablado. Resumiendo, la ejecución de un programa consiste en la ejecución secuencial de instrucciones. Cada instrucción se ejecuta durante un ciclo de instrucción, compuesto por subciclos más cortos (por ejemplo: subciclo de captación, indirecto, de ejecución, de interrupción ...). La ejecución de cada subciclo involucra una o más operaciones más breves, es decir, microoperaciones.

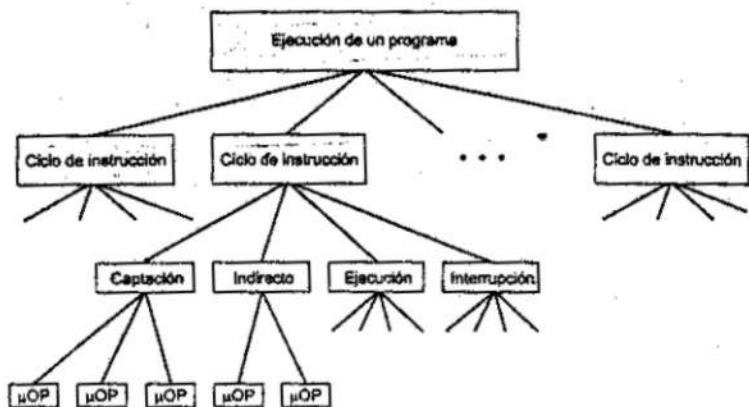


Figura 14.1. Elementos que constituyen la ejecución de un programa.

Las microoperaciones son las operaciones funcionales, o atómicas, de un procesador. En esta sección, examinaremos las microoperaciones, para llegar a comprender cómo los eventos de cualquier instrucción se pueden describir como una secuencia de tales microoperaciones. Usaremos un ejemplo sencillo. En el resto de este capítulo, mostraremos cómo el concepto de microoperaciones sirve como guía para el diseño de la unidad de control.

## EL CICLO DE CAPTACIÓN

Comenzamos examinando el ciclo de captación, que tiene lugar al principio de cada ciclo de instrucción, y hace que una instrucción sea captada de la memoria. Para el fin de este estudio, suponemos la organización representada en la Figura 11.7. Hay cuatro registros implicados:

- Registro de dirección de memoria (*memory address register*, MAR): Está conectado a las líneas de dirección del bus del sistema. Especifica la dirección de memoria de una operación de lectura o de escritura.
- Registro intermedio de memoria (*memory buffer register*, MBR): Está conectado a las líneas de datos del bus del sistema. Contiene el valor a almacenar en memoria o el último valor leído de memoria.
- Contador de programa (*program counter*, PC): Contiene la dirección de la siguiente instrucción a captar.
- Registro de instrucción (*instruction register*, IR): Contiene la última instrucción captada.

Consideremos la secuencia de eventos del ciclo de captación desde el punto de vista de su efecto sobre los registros del procesador. En la Figura 14.2 se muestra un ejemplo. Al comienzo del ciclo de captación, la dirección de la siguiente instrucción a ejecutar está en el contador de programa (PC); en este caso, la dirección es 1100100. El primer paso es llevar esa dirección al registro de dirección de memoria (MAR), ya que éste es el único registro conectado a las líneas de dirección del bus del sistema. El segundo paso es traer la instrucción. La dirección deseada (en MAR) se coloca en el bus de direcciones; la unidad de control emite una orden READ por el bus de control; el resultado aparece en el bus de datos y se copia en

|     |                  |
|-----|------------------|
| MAR |                  |
| MBR |                  |
| PC  | 0000000001100100 |
| IR  |                  |
| AC  |                  |

(a) Comienzo

|     |                  |
|-----|------------------|
| MAR | 0000000001100100 |
| MBR | 0001000000100000 |
| PC  | 0000000001100101 |
| IR  |                  |
| AC  |                  |

(c) Segundo paso

|     |                  |
|-----|------------------|
| MAR | 0000000001100100 |
| MBR |                  |
| PC  | 0000000001100100 |
| IR  |                  |
| AC  |                  |

(b) Primer paso

|     |                  |
|-----|------------------|
| MAR | 0000000001100100 |
| MBR | 0001000000100000 |
| PC  | 0000000001100101 |
| IR  | 0001000000100000 |
| AC  |                  |

(d) Tercer paso

Figura 14.2. Secuencia de eventos del ciclo de captación.

el registro intermedio de memoria (MBR). Es necesario, además, incrementar PC en  $I$  (longitud de la instrucción) para que esté preparado para la siguiente instrucción. Como estas dos acciones (leer una palabra de memoria, sumar  $I$  a PC) no interfieren entre sí, podemos hacerlas simultáneamente para ahorrar tiempo. El tercer paso es transferir el contenido de MBR al registro de instrucción (IR). Esto libera MBR para su uso durante un posible ciclo indirecto.

De este modo, el sencillo ciclo de captación consta de tres pasos y cuatro microoperaciones. Cada microoperación implica la transferencia de datos hacia dentro o hacia fuera de un registro. Con tal de que estas transferencias no interfieran entre sí, varias de ellas pueden tener lugar durante un paso, ahorrando tiempo. Simbólicamente, podemos escribir esta secuencia de eventos como sigue:

$$\begin{aligned} t_1: & \text{ MAR} \leftarrow (\text{PC}) \\ t_2: & \text{ MBR} \leftarrow \text{Memoria} \\ & \text{PC} \leftarrow (\text{PC}) + I \\ t_3: & \text{ IR} \leftarrow (\text{MBR}) \end{aligned}$$

donde  $I$  es la longitud de la instrucción. Tenemos que hacer varios comentarios sobre esta secuencia. Suponemos que se dispone de un reloj a efectos de temporización, y que éste emite pulsos de reloj espaciados regularmente. Cada pulso de reloj define una unidad de tiempo. Así, todas las unidades de tiempo tienen la misma duración. Cada microoperación puede llevarse a cabo dentro de una única unidad de tiempo. La notación  $(t_1, t_2, t_3)$  representa las sucesivas unidades de tiempo. En palabras, tenemos:

- Primera unidad de tiempo: Transferir el contenido de PC a MAR.
- Segunda unidad de tiempo: Transferir los contenidos de la posición de memoria especificada por MAR a MBR. Incrementar en  $I$  el contenido de PC.
- Tercera unidad de tiempo: Transferir el contenido de MBR a IR.

Observe que las microoperaciones segunda y tercera tienen lugar durante la segunda unidad de tiempo. La tercera microoperación podría haberse agrupado con la cuarta sin afectar a la operación de captación:

- $t_1: MAR \leftarrow (PC)$   
 $t_2: MBR \leftarrow \text{Memoria}$   
 $t_3: PC \leftarrow (PC) + I$   
 $IR \leftarrow (MBR)$

Los agrupamientos de microoperaciones deben cumplir dos sencillas reglas:

1. Debe seguirse la secuencia correcta de eventos. Así ( $MAR \leftarrow (PC)$ ) debe preceder ( $MBR \leftarrow \text{Memoria}$ ), ya que la operación de lectura de memoria hace uso de la dirección almacenada en MAR.
2. Deben evitarse los conflictos. No se debe intentar leer y escribir en el mismo registro en una unidad de tiempo, ya que los resultados serían imprevisibles. Por ejemplo, microoperaciones ( $MBR \leftarrow \text{Memoria}$ ) e ( $IR \leftarrow MBR$ ) no deberían tener lugar en misma unidad de tiempo.

Un punto final digno de atención es que una de las microoperaciones incluye una suma. Para evitar la duplicación de circuitería, la suma podría realizarse en la ALU. El uso de la ALU puede implicar microoperaciones adicionales, dependiendo de la funcionalidad de ALU y de la organización del procesador. Aplazamos la discusión de este punto hasta más adelante en este capítulo.

Es útil comparar los eventos descritos en ésta y en las siguientes subsecciones con la Figura 3.5. Mientras las microoperaciones se ignoraban en aquella figura, la presente discusión muestra las microoperaciones necesarias para llevar a cabo los subciclos del ciclo de instrucción.

### EL CICLO INDIRECTO

Una vez que se capta una instrucción, el siguiente paso es captar los operandos fuente. Siguiendo con nuestro sencillo ejemplo, supongamos un formato de instrucción de una dirección, que permite direccionamiento directo e indirecto. Si la instrucción especifica una dirección indirecta, un ciclo indirecto ha de preceder al ciclo de ejecución. El flujo de datos, que se indicó en la Figura 11.8, incluye las siguientes microoperaciones:

- $t_1: MAR \leftarrow (IR(\text{Dirección}))$   
 $t_2: MBR \leftarrow \text{Memoria}$   
 $t_3: IR(\text{Dirección}) \leftarrow (MBR/\text{Dirección})$

El campo de dirección en la instrucción se transfiere a MAR. Éste se usa después para captar la dirección del operando. Por último, el campo de dirección de IR se actualiza con el contenido de MBR, de modo que contenga una dirección directa en lugar de una indirecta.

IR tiene ahora el mismo estado que si no se hubiera usado direccionamiento indirecto, y está listo para el ciclo de ejecución. De momento pasamos por alto ese ciclo, para considerar el ciclo de interrupción.

### EL CICLO DE INTERRUPCIÓN

Cuando termina el ciclo de ejecución, se realiza una comprobación para determinar si ha ocurrido alguna interrupción habilitada. Si es así, tiene lugar un ciclo de interrupción. La naturaleza de este ciclo varía mucho de una máquina a otra. Aquí presentamos una secuencia muy simple de eventos, ilustrados en la Figura 11.9. Tenemos:

- $t_1: MAR \leftarrow (PC)$
- $t_2: MAR \leftarrow$  Dirección de salvaguardia  
 $PC \leftarrow$  Dirección de la rutina
- $t_3: Memoria \leftarrow (MBR)$

En el primer paso, el contenido de PC se transfiere a MBR, de modo que pueda guardarse para el retorno de la interrupción. Entonces, MAR se carga con la dirección en la cual va a guardarse el contenido de PC, y PC se carga con la dirección de comienzo de la rutina de procesamiento de la interrupción. Cada una de estas dos acciones puede ser una única microoperación. Sin embargo, ya que la mayoría de los procesadores tienen múltiples tipos y niveles de interrupciones, podrían hacer falta una o más microoperaciones adicionales para obtener la dirección de salvaguardia y la dirección de la rutina, antes de que puedan transferirse a MAR y a PC, respectivamente. En todo caso, una vez hecho esto, el paso final es almacinar MBR, que contiene el antiguo valor de PC, en la memoria. El procesador quedará entonces preparado para iniciar el siguiente ciclo de instrucción.

## EL CICLO DE EJECUCIÓN

Los ciclos de captación, indirecto y de interrupción, son sencillos y predecibles. Cada uno implica una secuencia pequeña y fija de microoperaciones y, en todos los casos, se repiten las mismas microoperaciones para cada ejecución de una instrucción.

Esto no ocurre así en el ciclo de ejecución. En una máquina con  $N$  códigos de operación diferentes, pueden ocurrir  $N$  secuencias diferentes de microoperaciones. Consideremos varios ejemplos hipotéticos.

En primer lugar, consideremos una instrucción de suma:

ADD R1, X

que suma el contenido de la posición X al registro R1. Puede suceder la siguiente secuencia de microoperaciones:

- $t_1: MAR \leftarrow (IR(\text{Dirección}))$
- $t_2: MBR \leftarrow \text{Memoria}$
- $t_3: R1 \leftarrow (R1) + (MBR)$

En un principio, IR contiene la instrucción ADD. En el primer paso, la parte de dirección de IR se carga en MAR. Después, se lee la posición de memoria referenciada. Por último, la ALU suma los contenidos de R1 y MBR. En realidad, éste es un ejemplo simplificado. Pueden necesitarse operaciones adicionales para extraer la referencia a registro desde IR y, tal vez, para poner las entradas o salidas de la ALU en algunos registros intermedios.

Consideremos dos ejemplos más complejos. Una instrucción frecuente es «incrementar y saltar si cero» (Increment and Skip if Zero):

ISZ X

El contenido de la posición X se incrementa en 1. Si el resultado es 0, la siguiente instrucción se salta. Una posible secuencia de microoperaciones es:

- $t_1: MAR \leftarrow (IR(\text{Dirección}))$
- $t_2: MBR \leftarrow \text{Memoria}$

- $t_3: MBR \leftarrow (MBR) + 1$   
 $t_4: Memoria \leftarrow (MBR)$   
 Si  $((MBR) = 0)$  entonces  $(PC \leftarrow (PC) + I)$

La nueva característica introducida aquí, es la actuación condicional. PC se incrementa :  $(MBR) = 0$ . Esta comprobación y actuación puede implementarse como una microoperación. Observe que esta microoperación puede ejecutarse durante la misma unidad de tiempo en la cual el valor actualizado de MBR se almacena en memoria.

Por último, examinemos una instrucción de llamada a subrutina. Como ejemplo, consideremos la instrucción «saltar y guardar la dirección» (Branch and Save Address):

#### BSA X

La dirección de la instrucción que viene a continuación de la instrucción BSA, se guarda en la posición X, y la ejecución continúa en la posición  $X + I$ . La dirección guardada se usará más adelante en el retorno. Ésta es una técnica sencilla para proporcionar llamadas a subrutas. Son suficientes las siguientes microoperaciones:

- $t_1: MAR \leftarrow (IR(\text{Dirección}))$   
 $MBR \leftarrow (PC)$   
 $t_2: PC \leftarrow (IR(\text{Dirección}))$   
 $Memoria \leftarrow (MBR)$   
 $t_3: PC \leftarrow (PC) + I$

La dirección que hay en PC al comienzo de la instrucción es la dirección de la siguiente instrucción secuencial. Ésta se guarda en la dirección señalada por IR. Esta última dirección también se incrementa, para obtener la dirección de la instrucción correspondiente al siguiente ciclo de instrucción.

### EL CICLO DE INSTRUCCIÓN

Hemos visto que cada fase del ciclo de instrucción puede descomponerse en una secuencia de microoperaciones elementales. En nuestro ejemplo, hay una secuencia para cada uno de los ciclos de captación, indirecto y de interrupción, y para el ciclo de ejecución existe una secuencia de microoperaciones para cada código de operación.

Para completar la descripción, tenemos que unir las secuencias de microoperaciones, como se ha hecho en la Figura 14.3. Suponemos que hay un nuevo registro de 2 bits llamado *código de ciclo de instrucción* (instruction cycle code, ICC). El ICC designa el estado del procesador en términos de en qué parte del ciclo se encuentra éste:

- 00: Captación
- 01: Indirecto
- 10: Ejecución
- 11: Interrupción

Al final de cada uno de los cuatro ciclos, el ICC se actualiza convenientemente. El ciclo indirecto siempre viene seguido del ciclo de ejecución. El ciclo de interrupción siempre es seguido por el ciclo de captación (ver Figura 11.5). En el caso de los ciclos de ejecución y captación, el siguiente ciclo depende del estado del sistema.

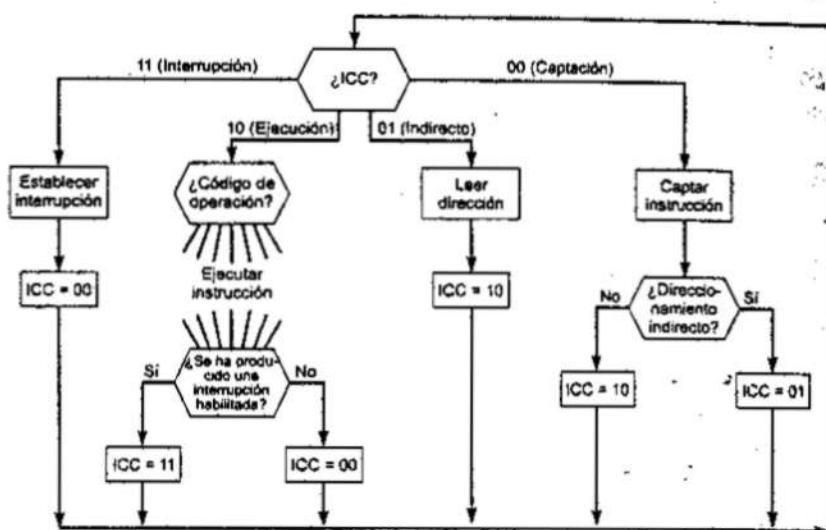


Figura 14.3. Diagrama de flujo del ciclo de instrucción.

De este modo, el diagrama de flujo del la Figura 14.3 define la secuencia completa de microoperaciones, que dependen sólo de la secuencia de instrucciones y del patrón de interrupciones. Naturalmente, éste es un ejemplo simplificado. El diagrama de flujo de un procesador real sería más complejo. En todo caso, hemos llegado al punto de nuestra discusión en el que el funcionamiento del procesador se define como la ejecución de una secuencia de microoperaciones. Podemos considerar ahora cómo origina esta secuencia el procesador.

## T4.2. CONTROL DEL PROCESADOR

### REQUISITOS FUNCIONALES

Como consecuencia de nuestro análisis de la sección precedente, hemos descompuesto el comportamiento o funcionamiento del procesador en operaciones elementales, llamadas «microoperaciones». Reduciendo el funcionamiento del procesador a su nivel más básico, podemos definir exactamente qué es lo que la unidad de control tiene que hacer que ocurra. Así, podemos definir los *requisitos funcionales* de la unidad de control como aquellas funciones que debe llevar a cabo. Una definición de estos requisitos funcionales es la base del diseño e implementación de la unidad de control.

Con la información a mano, el siguiente proceso de tres pasos lleva a la caracterización de la unidad de control:

1. Definir los elementos básicos del procesador.
2. Describir las microoperaciones que ejecuta el procesador.

3. Determinar las funciones que debe realizar la unidad de control para hacer que se ejecuten las microoperaciones.

Ya hemos presentado los pasos 1 y 2. Resumamos los resultados. En primer lugar, los elementos funcionales básicos del procesador son los siguientes:

- ALU
- Registros
- Caminos de datos internos
- Caminos de datos externos
- Unidad de control

Algunas consideraciones deberían convencerle de que esta lista está completa. La ALU es la esencia funcional del computador. Los registros se usan para almacenar datos internos del procesador. Algunos registros contienen información de estado necesaria para gestionar el secuenciamiento de las instrucciones (por ejemplo, la palabra de estado del programa). Otros contienen datos que van hacia, o vienen de, la ALU, la memoria, y los módulos de E/S. Los caminos de datos internos se usan para transferir datos entre los registros y entre éstos y la ALU. Los caminos de datos externos unen los registros a la memoria y a los módulos de E/S, a menudo por medio de un bus del sistema. La unidad de control hace que se produzcan operaciones dentro del procesador.

La ejecución de un programa consta de operaciones que involucran estos elementos del procesador. Como hemos visto, estas operaciones consisten en una secuencia de microoperaciones. Tras la revisión de la Sección 14.1, el lector debería observar que todas las microoperaciones se pueden clasificar en una de las siguientes categorías:

- Transferir datos de un registro a otro.
- Transferir datos de un registro a una interfaz externa (por ejemplo, al bus del sistema).
- Transferir datos de una interfaz externa a un registro.
- Realizar una operación aritmética o lógica, usando registros para entrada y salida.

Todas las microoperaciones necesarias para realizar un ciclo de instrucción, incluyendo todas las microoperaciones necesarias para ejecutar cada instrucción del repertorio, están incluidas en una de estas categorías.

Podemos ser ahora algo más explícitos acerca de la forma en que funciona la unidad de control. La unidad de control realiza dos tareas básicas:

- **Secuenciamiento:** La unidad de control hace que el procesador avance a través de una serie de microoperaciones en la secuencia oportuna, basada en el programa que se está ejecutando.
- **Ejecución:** La unidad de control hace que se ejecute cada microoperación.

Lo que precede es una descripción funcional de lo que hace la unidad de control. La clave de cómo funciona la unidad de control es la utilización de señales de control.

## SEÑALES DE CONTROL

Hemos definido los elementos que componen el procesador (ALU, registros y caminos de datos), y las microoperaciones que se llevan a cabo. Para que la unidad de control realice su función, debe tener entradas que le permitan determinar el estado del sistema, y salidas que le permitan controlar el comportamiento del mismo. Éstas son las especificaciones externas de

la unidad de control. Internamente, la unidad de control ha de tener la lógica necesaria para realizar sus funciones de secuenciamiento y ejecución. Apilaremos el estudio del funcionamiento interno de la unidad de control hasta la Sección 14.3 y el Capítulo 15. El resto de esta sección se ocupa de la interacción entre la unidad de control y otros elementos del procesador.

La Figura 14.4 es un modelo general de la unidad de control, que muestra todas sus entradas y salidas. Las entradas son las siguientes:

- Reloj: Es el encargado de «mantener la hora exacta» en el procesador. La unidad de control hace que se ejecute una microoperación (o un conjunto de microoperaciones simultáneas) en cada pulso de reloj. Éste, a menudo, es referenciado como «tiempo de ciclo del procesador», o «período de reloj».
- Registro de instrucción: El código de operación de la instrucción en curso se usa para determinar qué microoperaciones hay que realizar durante el ciclo de ejecución.
- Indicadores: Los necesita la unidad de control para determinar el estado del procesador y el resultado de anteriores operaciones de la ALU. Por ejemplo, para la instrucción incrementar y saltar si cero (ISZ), la unidad de control incrementará PC si el indicador de cero está a uno.
- Señales de control del bus de control: La parte de control del bus del sistema suministra señales a la unidad de control, tales como señales de interrupción y de reconocimiento.

Las salidas son las siguientes:

- Señales de control internas al procesador: Son de dos tipos: las que hacen que los datos se transfieran de un registro a otro, y las que activan funciones específicas de la ALU.
- Señales de control hacia el bus de control: También las hay de dos tipos: señales de control de la memoria, y señales de control de los módulos de E/S.

El nuevo concepto introducido en esta figura es el de señal de control. Se usan tres tipos de señales de control: las que activan una función de la ALU, las que activan un camino de datos, y las que son señales del bus del sistema externo u otra interfaz externa. Todas estas señales se aplican al final, como entradas binarias, directamente a puertas lógicas individuales.

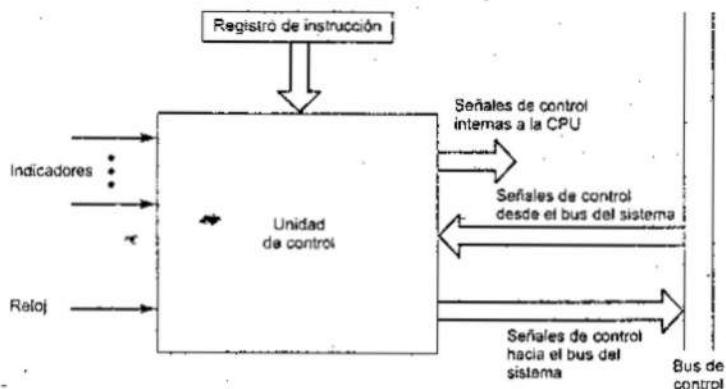


Figura 14.4. Modelo de la unidad de control.

Consideremos nuevamente el ciclo de captación, para entender cómo mantiene el control la unidad de control. La unidad de control se mantiene al tanto de dónde está dentro el ciclo de instrucción. En un punto determinado, sabe que inmediatamente después se va a realizar un ciclo de captación. El primer paso es transferir el contenido de PC a MAR. La unidad de control hace esto activando la señal de control que abre las puertas entre los bits PC y los bits de MAR. El siguiente paso es leer una palabra desde memoria a MBR e iniciar PC. La unidad de control hace esto enviando las siguientes señales de control simultáneamente:

- Una señal de control, que abre las puertas que permiten que el contenido de MAR aparezca en el bus de direcciones.
- Una señal de control de lectura de memoria, en el bus de control.
- Una señal de control, que abre las puertas que permiten almacenar el contenido del bit de datos en MBR.
- Señales de control de la lógica, que suman 1 al contenido de PC y almacenan el resultado de nuevo en PC.

Después de esto, la unidad de control envía una señal de control, que abre las puertas adecuadas entre MBR e IR.

Esto completa el ciclo de captación, exceptuando un detalle: la unidad de control debe decidir si ejecuta a continuación un ciclo indirecto o un ciclo de ejecución. Para decidir esto examina IR, viendo si se hace una referencia indirecta a memoria.

Los ciclos indirecto y de interrupción funcionan de un modo parecido. En el ciclo de ejecución, la unidad de control comienza examinando el código de operación y, en función de él decide qué secuencia de microoperaciones deben realizarse para ejecutar el ciclo.

## UN EJEMPLO DE SEÑALES DE CONTROL

Para ilustrar el funcionamiento de la unidad de control, examinemos un ejemplo sencillo. La Figura 14.5 ilustra el ejemplo. Se trata de un procesador sencillo, con un único acumulador. Se indican los caminos de datos entre los distintos elementos. Los caminos de control de las señales que proceden de la unidad de control no se muestran, pero las terminaciones de las señales de control están designadas como  $C_i$ , y se indican mediante un círculo. La unidad de control recibe entradas del reloj, del registro de instrucción, y de los indicadores. En cada ciclo de reloj, la unidad de control lee todas sus entradas y emite un conjunto de señales de control. Las señales de control van hacia tres destinos distintos:

- **Caminos de datos:** La unidad de control dirige el flujo interno de datos. Por ejemplo, en la captación de instrucción, el contenido del registro intermedio de memoria se transfiere al registro de instrucción. Por cada camino a controlar, hay una puerta (indicada mediante un círculo en la figura). Una señal de control de la unidad de control abre temporalmente la puerta para dejar pasar los datos.
- **ALU:** La unidad de control gobierna el funcionamiento de la ALU mediante un conjunto de señales de control. Estas señales activan diversos dispositivos y puertas dentro de la ALU.
- **Bus del sistema:** La unidad de control envía señales de control a las líneas de control del bus del sistema (por ejemplo, lectura de la memoria).

La unidad de control debe conocer en todo momento dónde está dentro del ciclo de instrucción. Usando ese conocimiento, y leyendo todas sus entradas, la unidad de control emite

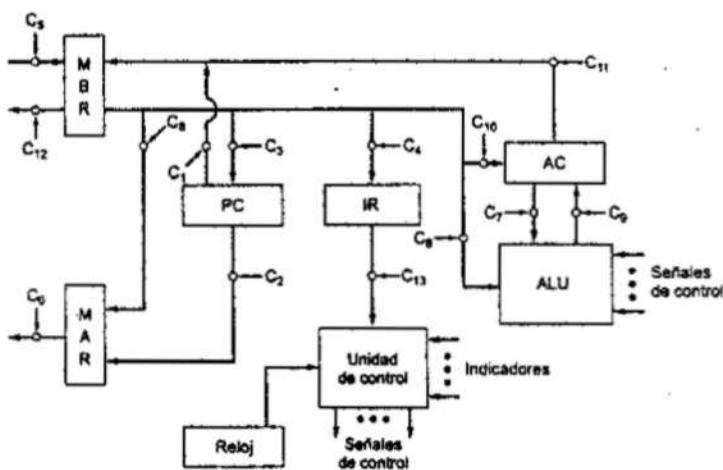


Figura 14.5. Caminos de datos y señales de control.

una serie de señales de control, que provocan las microoperaciones. La unidad de control usa los pulsos de reloj para temporizar la secuencia de eventos, dejando tiempo entre eventos para que los niveles de las señales se estabilicen. La Tabla 14.1 indica las señales de control necesarias para realizar algunas de las secuencias de microoperaciones descritas con anterioridad. Por simplicidad, no se han mostrado los caminos de datos y de control necesarios para incrementar PC y para cargar direcciones fijas en PC y MAR.

El carácter mínimo de la unidad de control merece ser considerado. La unidad de control es el motor que mueve todo el computador, y lo hace basándose sólo en el conocimiento de las instrucciones que tiene que ejecutar y en la naturaleza de los resultados de las operaciones aritméticas y lógicas (por ejemplo, resultado positivo, desbordamiento, etc.). Nunca

Tabla 14.1. Microoperaciones y señales de control

| Microoperaciones | Temporización                                                                                                                                             | Señales de control activas                                          |
|------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------|
| Captación:       | t1: MAR $\leftarrow$ (PC)<br>t2: MBR $\rightarrow$ Memoria<br>PC $\leftarrow$ (PC) + 1<br>t3: IR $\leftarrow$ (MBR)                                       | C <sub>2</sub><br>C <sub>3</sub> , C <sub>8</sub><br>C <sub>6</sub> |
| Indirecto:       | t1: MAR $\leftarrow$ (IR(Dirección))<br>t2: MBR $\leftarrow$ Memoria<br>t3: IR(Dirección) $\leftarrow$ (MBR(Dirección))                                   | C <sub>3</sub><br>C <sub>4</sub> , C <sub>8</sub><br>C <sub>5</sub> |
| Interrupción:    | t1: NBR $\leftarrow$ (PC)<br>t2: MAR $\rightarrow$ Dirección de salvaguardia<br>PC $\rightarrow$ Dirección de la rutina<br>t3: Memoria $\leftarrow$ (MBR) | C <sub>1</sub><br>C <sub>12</sub> , C <sub>W</sub>                  |

C<sub>n</sub> = Señal de control de lectura (<read>) hacia el bus del sistema

C<sub>w</sub> = Señal de control de escritura (<write>) hacia el bus del sistema

llega a ver los datos que se procesan o los resultados reales producidos. Y controla todo con unas pocas señales de control que van a ciertos puntos dentro del procesador, y unas pocas señales que van hacia el bus del sistema.

### ORGANIZACIÓN INTERNA DEL PROCESADOR

La Figura 14.5 indica el uso de diversos caminos de datos. La complejidad de este tipo de organización debería estar clara. Es más normal usar algún tipo de configuración de bus interno, como se sugirió en la Figura 11.2.

Usando un bus interno al procesador, la Figura 14.5 puede disponerse del modo que muestra la Figura 14.6. La ALU y todos los registros del procesador están conectados por un único bus interno. Las puertas y las señales de control se proporcionan para realizar transferencias de datos entre el bus y cada registro. Otras señales de control dirigen las transferencias de datos hacia y desde el bus (externo) del sistema y el funcionamiento de la ALU.

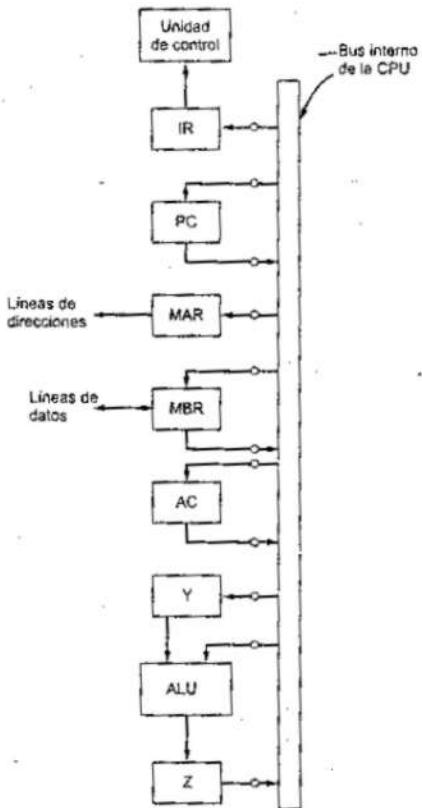


Figura 14.6. Procesador con bus interno.

Se han añadido dos nuevos registros, referenciados como Y y Z, a la organización. Son necesarios para el correcto funcionamiento de la ALU. Cuando se realiza una operación que involucra dos operandos, uno se puede obtener desde el bus interno, pero el otro ha de obtenerse de otra fuente. El AC podría usarse para este propósito, pero ello limita la flexibilidad del sistema, y no funcionaría en un procesador con múltiples registros de uso general. El registro Y proporciona un almacenamiento temporal de la otra entrada. La ALU es un circuito combinacional (véase Apéndice A) sin almacenamiento interno. De este modo, cuando las señales de control activan una función de la ALU, a partir de la entrada de ésta se obtiene una salida. Debido a ello, la salida de la ALU no se puede conectar directamente al bus, ya que realimentaría la entrada. El registro Z proporciona el almacenamiento temporal de salida. Con esta configuración, una operación de suma de un valor de la memoria al AC tendría los siguientes pasos:

- $t_1: MAR \leftarrow (IR \text{ Dirección})$
- $t_2: MBR \leftarrow \text{Memoria}$
- $t_3: Y \leftarrow (MBR)$
- $t_4: Z \leftarrow (AC) + (Y)$
- $t_5: AC \leftarrow (Z)$

Son posibles otras organizaciones, pero, en general, se usa algún tipo de bus interno o conjunto de buses internos. El uso de caminos de datos comunes simplifica el trazado de las interconexiones y el control del procesador. Otra razón práctica para usar un bus interno es ahorrar espacio. El espacio ocupado por las conexiones entre registros tiene que minimizarse, especialmente en los microprocesadores, que sólo pueden ocupar un trozo cuadrado de silicio de 1/4 de pulgada.

## EL INTEL 8085

Para ilustrar algunos de los conceptos introducidos hasta aquí en este capítulo, consideremos el Intel 8085. Su organización se muestra en la Figura 14.7. Hay varios componentes clave que pueden no explicarse por sí mismos:

- **Latch (cerrojo) incrementador/decrementador de direcciones:** Lógica que puede sumar 1 o restar 1 al contenido del puntero de pila o al contador de programa. Esto ahorra tiempo, ya que se evita usar la ALU para este fin.
- **Control de interrupciones:** Este módulo maneja múltiples niveles de señales de interrupción.
- **Control de E/S serie:** Este módulo se conecta a dispositivos capaces de transferir 1 bit cada vez.

La Tabla 14.2 describe las señales externas que entran y salen del 8085. Éstas se unen al bus externo del sistema. Estas señales son la interfaz entre el procesador 8085 y el resto del sistema (Figura 14.8).

La unidad de control se identifica como los dos componentes rotulados: (1) decodificador de instrucciones y codificación del ciclo máquina, y (2) temporización y control. Se aplaza la discusión del primer componente hasta la siguiente sección. La parte fundamental de la unidad de control es el módulo de temporización y control. Este módulo incluye un reloj, y acepta como entradas la instrucción en curso y algunas señales de control externas. Su salida consiste en señales de control hacia los otros componentes del procesador, más señales de control hacia el bus externo del sistema.

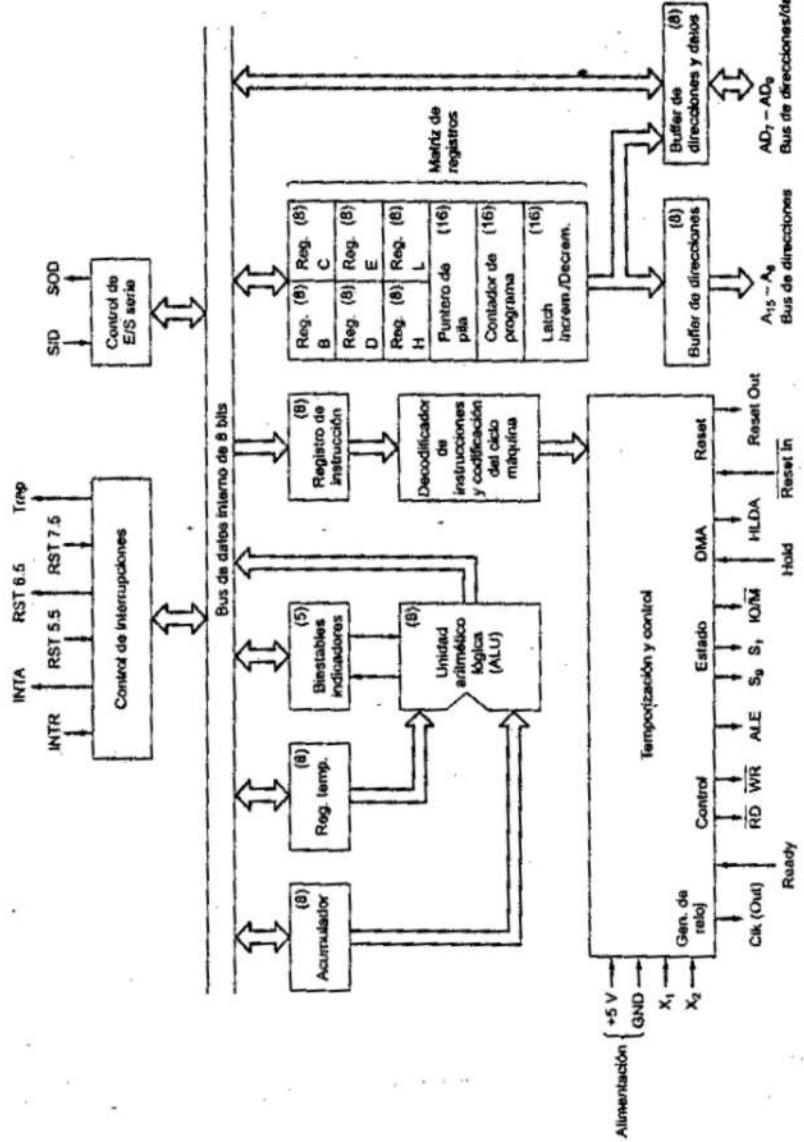


Figura 14.7. Diagrama de bloques del procesador Intel 8086.

Tabla 14.2. Señales externas del Intel 8085

| Señales de datos y direcciones                                                                                                                                                                                                                                                                                                                                                                             |  |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|
| <b>Direcciones altas (A<sub>15</sub>-A<sub>0</sub>)</b><br>Los 8 bits de orden alto de una dirección de 16 bits.                                                                                                                                                                                                                                                                                           |  |
| <b>Direcciones/datos (AD<sub>7</sub>-AD<sub>0</sub>)</b><br>Los 8 bits de orden bajo de una dirección de 16 bits, o bien 8 bits de datos. Esta multiplexación ahorra patillas.                                                                                                                                                                                                                             |  |
| <b>Dato de entrada serie (Serial Input Data, SID)</b><br>Una entrada de un único bit para adaptarse a dispositivos que transmitan de forma serie (un bit cada vez).                                                                                                                                                                                                                                        |  |
| <b>Dato de salida serie (Serial Output Data, SOD)</b><br>Una salida de un único bit para adaptarse a dispositivos que reciben de forma serie.                                                                                                                                                                                                                                                              |  |
| Señales de temporización y control                                                                                                                                                                                                                                                                                                                                                                         |  |
| <b>CLK (Out)</b><br>El reloj del sistema. Cada ciclo representa un estado T. La señal CLK va hacia circuitos periféricos y sincroniza su temporización.                                                                                                                                                                                                                                                    |  |
| <b>X<sub>1</sub>, X<sub>2</sub></b><br>Estas señales provienen de un cristal externo u otro dispositivo, para controlar si generador de reloj interno.                                                                                                                                                                                                                                                     |  |
| <b>Habilitación del latch de direcciones (Address Latch Enable, ALE)</b><br>Tiene lugar durante el primer estado de un ciclo máquina, y hace que los circuitos periféricos almacenen las líneas de dirección. Esto permite a un módulo (por ejemplo, memoria E/S) reconocer que está siendo direccionado.                                                                                                  |  |
| <b>Estado (S<sub>0</sub>, S<sub>1</sub>)</b><br>Señales de control usadas para indicar si está teniendo lugar una operación de lectura o escritura.                                                                                                                                                                                                                                                        |  |
| <b>IO/M</b><br>Usada para controlar operaciones de lectura o escritura de los módulos de E/S o de memoria.                                                                                                                                                                                                                                                                                                 |  |
| <b>Control de lectura (RD)</b><br>Indica que el módulo de memoria o de E/S seleccionado va a leerse, y que el bus de datos está disponible para una transferencia de datos.                                                                                                                                                                                                                                |  |
| <b>Control de escritura (WR)</b><br>Indica que el dato en el bus de datos es para escribirlo en la posición de memoria E/S seleccionada.                                                                                                                                                                                                                                                                   |  |
| Señales originadas en memoria y E/S                                                                                                                                                                                                                                                                                                                                                                        |  |
| <b>Hold</b><br>Pide al procesador que abandone el control y el uso del bus del sistema externo. El procesador completará la ejecución de la instrucción que hay en el momento presente en el IR, y entrará en un estado de desconexión, durante el cual no pone señales en los buses de control, de direcciones y de datos. Durante el estado de desconexión, el bus puede usarse para operaciones de DMA. |  |
| <b>Reconocimiento de Hold (Hold Acknowledge, HLDA)</b><br>Esta señal de salida de la unidad de control reconoce la señal HOLD, e indica que el bus está disponible.                                                                                                                                                                                                                                        |  |
| <b>Ready</b><br>Usado para sincronizar el procesador con una memoria más lenta o con dispositivos de E/S. Cuando un dispositivo seleccionado mantiene Ready a uno, el procesador puede continuar con una operación de entrada (D8IN) o salida (WR). En caso contrario, el procesador entra en un estado de espera hasta que el dispositivo esté listo.                                                     |  |
| Señales relacionadas con interrupciones                                                                                                                                                                                                                                                                                                                                                                    |  |
| <b>Trap</b><br>Interrupciones de reanudación (RST 7.5, 6.5, 5.5)                                                                                                                                                                                                                                                                                                                                           |  |
| <b>Petición de interrupción (INTR)</b><br>Un dispositivo externo usa estas cinco líneas para interrumpir al procesador. Éste no aceptará una petición si se encuentra en el estado de desconexión, o si la interrupción está inhabilitada. Una interrupción se acepta sólo al final de una instrucción. Estas interrupciones se indican en orden de prioridad descendente.                                 |  |
| <b>Reconocimiento de Interrupción</b><br>Reconoce una interrupción.                                                                                                                                                                                                                                                                                                                                        |  |
| Reinicio del procesador                                                                                                                                                                                                                                                                                                                                                                                    |  |
| <b>Reset In</b><br>Provoca la puesta a cero del contenido de PC. El procesador reanuda la ejecución en la posición cero.                                                                                                                                                                                                                                                                                   |  |
| <b>Reset Out</b><br>Reconoce que el procesador ha sido reiniciado. Esta señal puede usarse para reiniciar el resto del sistema.                                                                                                                                                                                                                                                                            |  |
| Alimentación y tierra                                                                                                                                                                                                                                                                                                                                                                                      |  |
| <b>V<sub>cc</sub></b><br>Alimentación a +5 voltios.                                                                                                                                                                                                                                                                                                                                                        |  |
| <b>V<sub>ss</sub></b><br>Tierra.                                                                                                                                                                                                                                                                                                                                                                           |  |

|                 |    |    |                 |
|-----------------|----|----|-----------------|
| X <sub>1</sub>  | 1  | 40 | V <sub>CC</sub> |
| X <sub>2</sub>  | 2  | 39 | Hold            |
| Reset Out       | 3  | 38 | HLDA            |
| SOD             | 4  | 37 | Clk (Out)       |
| SID             | 5  | 36 | Reset In        |
| Trap            | 6  | 35 | Ready           |
| RST 7.5         | 7  | 34 | IO/M            |
| RST 6.5         | 8  | 33 | S <sub>1</sub>  |
| RST 5.5         | 9  | 32 | RD              |
| INTR            | 10 | 31 | WR              |
| INTA            | 11 | 30 | ALE             |
| AD <sub>0</sub> | 12 | 29 | S <sub>0</sub>  |
| AD <sub>1</sub> | 13 | 28 | A <sub>15</sub> |
| AD <sub>2</sub> | 14 | 27 | A <sub>14</sub> |
| AD <sub>3</sub> | 15 | 26 | A <sub>13</sub> |
| AD <sub>4</sub> | 16 | 25 | A <sub>12</sub> |
| AD <sub>5</sub> | 17 | 24 | A <sub>11</sub> |
| AD <sub>6</sub> | 18 | 23 | A <sub>10</sub> |
| AD <sub>7</sub> | 19 | 22 | A <sub>9</sub>  |
| V <sub>SS</sub> | 20 | 21 | A <sub>8</sub>  |

Figura 14.8. Configuración de patillas del Intel 8085.

La temporización de las operaciones del procesador está sincronizada por el reloj, y está controlada por la unidad de control por medio de señales de control. Cada ciclo de instrucción se divide en *ciclos máquina*, de uno a cinco; cada ciclo máquina se divide a su vez en *estados*, de tres a cinco. Cada estado dura un ciclo de reloj. Durante un estado, el procesador ejecuta una o un conjunto de microoperaciones simultáneas, determinadas por las señales de control.

El número de ciclos máquina es fijo para cada instrucción, pero varía de una instrucción a otra. Los ciclos máquina se definen como equivalentes a los accesos al bus. De este modo, el número de ciclos máquina de una instrucción depende del número de veces que el procesador debe comunicarse con los dispositivos externos. Por ejemplo, si una instrucción se compone de dos partes de 8 bits, se necesitan dos ciclos máquina para captar la instrucción. Si la instrucción implica una operación de 1 byte con memoria o E/S, será necesario un tercer ciclo máquina para su ejecución.

La Figura 14.9 ofrece un ejemplo de temporización del 8085 que muestra el valor de las señales de control externas. Naturalmente, al mismo tiempo, la unidad de control está generando señales de control internas para controlar transferencias de datos en el interior del procesador. El diagrama muestra el ciclo de instrucción de la instrucción OUT. Se necesitan tres ciclos máquina ( $M_1$ ,  $M_2$ ,  $M_3$ ). Durante el primero, se capta la instrucción OUT. El segundo ciclo máquina capta la segunda mitad de la instrucción, que contiene el número del dispositivo de E/S seleccionado para salida. Durante el tercer ciclo, el contenido del AC se escribe, a través del bus de datos, en el dispositivo seleccionado.

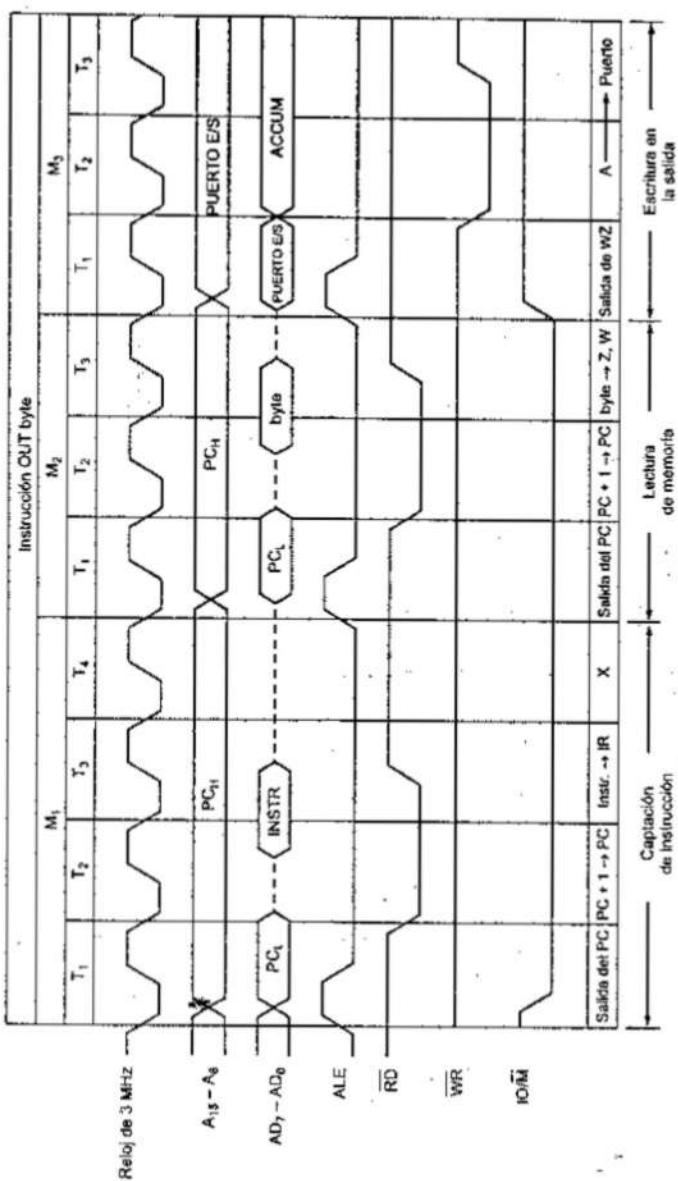


Figura 14.9. Diagrama de tiempos de la instrucción OUT del Intel 8085.

El comienzo de cada ciclo máquina viene determinado por el pulso de habilitación del latch (cerrojo) de direcciones (Address Latch Enable, ALE), emitido por la unidad de control. Durante el estado de temporización  $T_1$  del ciclo máquina  $M_1$ , la unidad de control ajusta la señal IO/M para indicar una operación con memoria. Además, la unidad de control hace que el contenido de PC se sitúe en el bus de direcciones ( $A_{15} - A_0$ ) y el bus de direcciones/datos ( $AD_7 - AD_0$ ). En el flanco de bajada del pulso ALE, los otros módulos conecta dos al bus y almacenan la dirección.

Durante el estado de temporización  $T_2$ , el módulo de memoria direccionado pone el contenido de la posición de memoria en el bus de direcciones/datos. La unidad de control activa la señal de control de lectura ( $\overline{RD}$ ) para indicar una lectura, pero espera hasta  $T_3$  para captar los datos del bus. Esto deja tiempo al módulo de memoria para poner los datos en el bus, y para que se estabilicen los niveles de las señales. El estado final,  $T_4$ , es un estado de *bus desocupado*, durante el cual el procesador decodifica la instrucción. Los restantes ciclos máquina actúan de forma parecida.

#### 14.3. IMPLEMENTACIÓN CABLEADA:

Hemos estudiado la unidad de control en lo referente a entradas, salidas y funciones. Ahora es el momento de volver al tema de la implementación de la unidad de control. Se ha usado una gran variedad de técnicas. La mayoría de ellas se pueden clasificar en dos categorías:

- Implementación cableada.
- Implementación microprogramada.

En una implementación cableada, la unidad de control es esencialmente un circuito combinacional. Sus señales lógicas de entrada se transforman en un conjunto de señales de salida, que son las señales de control. Este enfoque se examina en esta sección. La implementación microprogramada es el tema del que trata el Capítulo 15.

#### ENTRADAS DE LA UNIDAD DE CONTROL

La Figura 14.4 representa la unidad de control como la hemos estudiado hasta aquí. Las entradas principales son el registro de instrucción, el reloj, los indicadores y las señales de control del bus. En el caso de los indicadores y de las señales de control del bus, cada bit individual tiene normalmente un significado determinado (por ejemplo, desbordamiento). Las otras dos entradas, sin embargo, directamente no son útiles a la unidad de control tal como entran.

Consideremos en primer lugar el registro de instrucción. La unidad de control hace uso del código de operación, y realiza acciones diferentes (emite una combinación diferente de señales de control) para cada instrucción. Para simplificar la lógica de la unidad de control, debería existir una entrada lógica única para cada código de operación. Esta función la puede realizar un *decodificador*, que toma una entrada codificada y produce una salida única. En general, un decodificador tendrá  $n$  entradas binarias y  $2^n$  salidas binarias. Cada uno de los  $2^n$  patrones de entrada distintos activará una única salida. La Tabla 14.3 es un ejemplo. El decodificador de una unidad de control normalmente tendrá que ser más complejo que el de la tabla, para representar códigos de operación de longitud variable. En el Apéndice A se presenta un ejemplo de la lógica digital usada para realizar un decodificador.

El reloj de la unidad de control emite una secuencia repetitiva de pulsos. Esto es útil para delimitar la duración de las microoperaciones. Esencialmente, el período de los pulsos de reloj ha de ser suficientemente largo para permitir la propagación de las señales a lo largo de

Tabla 14.3. Un decodificador con cuatro entradas y diecisésis salidas

| I1 | I2 | I3 | I4 | O1 | O2 | O3 | O4 | O5 | O6 | O7 | O8 | O9 | O10 | O11 | O12 | O13 | O14 | O15 | O16 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|-----|-----|-----|-----|-----|-----|
| 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0   | 0   | 0   | 0   | 1   |     |
| 0  | 0  | 0  | 1  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0   | 0   | 0   | 1   | 0   |     |
| 0  | 0  | 1  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0   | 0   | 1   | 0   | 0   |     |
| 0  | 0  | 1  | 1  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0   | 1   | 0   | 0   | 0   |     |
| 0  | 1  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0   | 1   | 0   | 0   | 0   |     |
| 0  | 1  | 0  | 1  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 1   | 0   | 0   | 0   | 0   |     |
| 0  | 1  | 1  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 1   | 0   | 0   | 0   | 0   | 0   |     |
| 0  | 1  | 1  | 1  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 1  | 0   | 0   | 0   | 0   | 0   | 0   |     |
| 1  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 1  | 0  | 0   | 0   | 0   | 0   | 0   | 0   |     |
| 1  | 0  | 0  | 1  | 0  | 0  | 0  | 0  | 0  | 0  | 1  | 0  | 0  | 0   | 0   | 0   | 0   | 0   | 0   |     |
| 1  | 0  | 1  | 0  | 0  | 0  | 0  | 0  | 0  | 1  | 0  | 0  | 0  | 0   | 0   | 0   | 0   | 0   | 0   |     |
| 1  | 0  | 1  | 1  | 0  | 0  | 0  | 0  | 1  | 0  | 0  | 0  | 0  | 0   | 0   | 0   | 0   | 0   | 0   |     |
| 1  | 1  | 0  | 0  | 0  | 0  | 0  | 1  | 0  | 0  | 0  | 0  | 0  | 0   | 0   | 0   | 0   | 0   | 0   |     |
| 1  | 1  | 0  | 1  | 0  | 0  | 1  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0   | 0   | 0   | 0   | 0   |     |
| 1  | 1  | 1  | 0  | 0  | 1  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0   | 0   | 0   | 0   | 0   |     |
| 1  | 1  | 1  | 1  | 1  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0   | 0   | 0   | 0   | 0   |     |

los caminos de datos, y a través de la circuitería del procesador. Sin embargo, como hemos visto, la unidad de control emite señales de control diferentes en unidades de tiempo diferentes, dentro de un único ciclo de instrucción. Por tanto, podríamos tener un contador como entrada a la unidad de control, con una señal de control diferente para  $T_1$ ,  $T_2$ , etc. Al final de un ciclo de instrucción, la unidad de control deberá realimentar el contador para reiniciarlo a  $T_1$ .

Con estos dos refinamientos, la unidad de control se puede representar como en la Figura 14.10.

## LÓGICA DE LA UNIDAD DE CONTROL

Para definir la implementación

Esencialmente, lo que tenemos que hacer es obtener, para cada señal de control, su expresión booleana como una función de las entradas. Esto se explica mejor con un ejemplo. Consideremos otra vez nuestro sencillo ejemplo, ilustrado en la Figura 14.5. Vimos en la Tabla 14.1 las secuencias de microoperaciones y de señales de control necesarias para controlar tres de las cuatro fases del ciclo de instrucción.

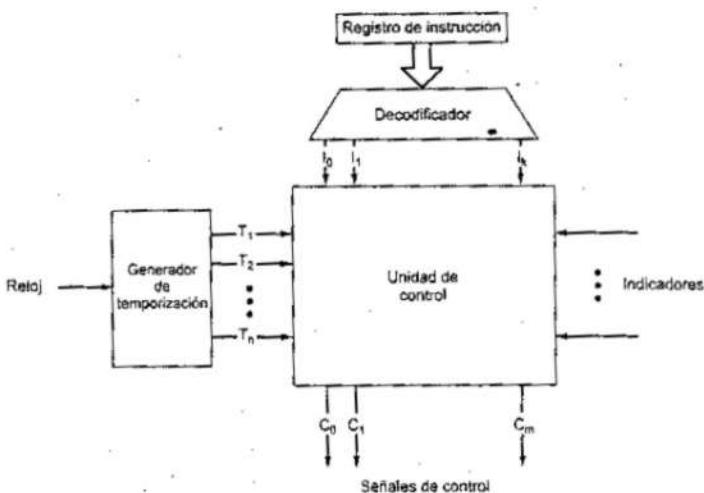


Figura 14.10. Unidad de control con entradas decodificadas.

Consideremos una única señal de control,  $C_5$ . Esta señal hace que se lean datos del bus de datos externo en MBR. Podemos ver que se usa dos veces en la Tabla 14.1. Definamos dos nuevas señales de control, P y Q, que tengan la siguiente interpretación:

|           |                       |
|-----------|-----------------------|
| $PQ = 00$ | Ciclo de captación    |
| $PQ = 01$ | Ciclo indirecto       |
| $PQ = 10$ | Ciclo de ejecución    |
| $PQ = 11$ | Ciclo de interrupción |

La siguiente expresión booleana define a  $C_5$ :

$$C_5 = \bar{P} \cdot \bar{Q} \cdot T_2 + \bar{P} \cdot Q \cdot T_2$$

Es decir, la señal de control,  $C_5$ , se pondrá a uno durante la segunda unidad de tiempo de los ciclos de captación e indirecto.

Esta expresión no está completa.  $C_5$  se necesita también durante el ciclo de ejecución. Para nuestro sencillo ejemplo, supongamos que hay sólo tres instrucciones que leen de la memoria: LDA, ADD y AND. Ahora podemos definir  $C_5$  como:

$$C_5 = \bar{P} \cdot \bar{Q} \cdot T_2 + \bar{P} \cdot Q \cdot T_2 + P \cdot \bar{Q} \cdot (LDA + ADD + AND) \cdot T_2$$

Este mismo proceso podría repetirse para cada señal de control generada por el procesador. El resultado sería un conjunto de ecuaciones booleanas que definiría el comportamiento de la unidad de control y, por tanto, del procesador.

Juntando todo, la unidad de control debe controlar el estado del ciclo de instrucción. Como se mencionó, al final de cada subciclo (captación, indirecto, ejecución, interrupción), la unidad de control emite una señal, que hace que el generador de temporización se reinicie y

emita  $T_1$ . Además, la unidad de control ha de establecer los valores adecuados de P y Q para definir el siguiente subciclo a ejecutar.

El lector debe comprender que en un complejo procesador moderno, el número de ecuaciones booleanas necesarias para definir la unidad de control es muy grande. La tarea de implementar un circuito combinacional que satisfaga todas esas ecuaciones, llega a ser sumamente difícil. El resultado es que, por regla general, se usa una aproximación mucho más sencilla, conocida como *microporgramación*. De este tema se ocupa el próximo capítulo.

#### 14.4. ESTRUCTURAS RECOMENDADAS

Varios libros de texto tratan los principios básicos del funcionamiento de la unidad de control, entre los que se incluyen [WARD90] y [MANO97].

MANO97 Mano, M. *Logic and Computer Design Fundamentals*. Upper Saddle River, NJ: Prentice Hall, 1997.

WARD90 Ward, S. y Halstead, R. *Computation Structures*. Cambridge, MA: MIT Press, 1990.

#### 14.5. PROBLEMAS

- 14.1. Dispone de una ALU que puede sumar sus dos registros de entrada y puede hacer la negación lógica de los bits de cada registro de entrada, pero no puede restar. Los números se van a almacenar en complemento a dos. Enumere las microoperaciones que debe realizar la ALU para hacer una resta.
- 14.2. Muestre las microoperaciones y señales de control, tal como lo hace la Tabla 14.1 para el procesador de la Figura 14.5, para las siguientes instrucciones:
  - Cargar el acumulador
  - Almacenar el acumulador
  - Sumar al acumulador
  - AND con el acumulador
  - Saltar
  - Saltar si AC = 0
  - Complementar el acumulador
- 14.3. Suponga que los retardos de propagación a lo largo del bus, y a través de la ALU de la Figura 14.6, son 20 y 100 ns, respectivamente. El tiempo necesario para la carga de datos en un registro desde el bus es 10 ns. ¿Cuál es el tiempo que debe asignarse a
  - a) transferir datos de un registro a otro?
  - b) incrementar el contador de programa?

- 14.4. Escriba la secuencia de microoperaciones necesaria en la estructura de bus de la Figura 14.6, para sumar un número al AC si el número es:
- un operando inmediato
  - un operando direccionado directamente
  - un operando direccionado indirectamente
- 14.5. Una pila está organizada como se muestra en la Figura 9.14. Muestre la secuencia de microoperaciones para:
- desapilar
  - apilar

## CAPÍTULO 15

# Control microprogramado

### 15.1. Conceptos básicos

Microinstrucciones  
Unidad de control microprogramada  
Control de Wilkes  
Ventajas e inconvenientes

### 15.2. Secuenciamiento de microinstrucciones

Consideraciones respecto al diseño  
Técnicas de secuenciamiento  
Generación de direcciones  
Secuenciamiento de microinstrucciones en el LSI-11

### 15.3. Ejecución de microinstrucciones

Una taxonomía de las microinstrucciones  
Codificación de las microinstrucciones  
Ejecución de microinstrucciones en el LSI-11  
Ejecución de microinstrucciones en el IBM 3033

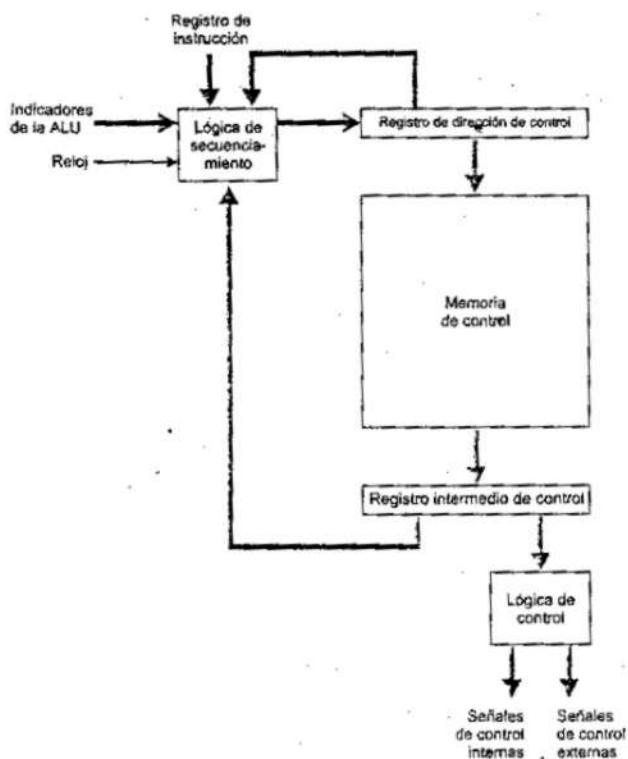
### 15.4. TI 8800

Formato de microinstrucción  
Microsecuenciador  
ALU con registros

### 15.5. Aplicaciones de la micropogramación

### 15.6. Lecturas recomendadas

### 15.7. Problemas



- Una alternativa a la unidad de control cableada es la unidad de control microprogramada, en la cual la lógica de la unidad de control viene dada por un micropograma. Un micropograma consiste en una secuencia de instrucciones en un lenguaje de microprogramación. Se trata de instrucciones muy elementales, que especifican microoperaciones.
- Una unidad de control microprogramada es un circuito lógico relativamente sencillo que es capaz de (1) realizar el secuenciamiento de las microinstrucciones y (2) generar las señales de control para ejecutar cada microinstrucción.
- Como en la unidad de control cableada, las señales de control generadas por una microinstrucción se usan para producir transferencias entre registros y operaciones de la ALU.

**E**l término *microprograma* lo acuñó M. V. Wilkes a principios de los años 50 [WILK51]. Wilkes propuso una aproximación al diseño de la unidad de control que era ordenada y sistemática, y evitaba la complejidad de la implementación cableada. La idea despertó la curiosidad de muchos investigadores, pero se mostraba irrealizable, porque necesitaba una memoria de control que fuera rápida y relativamente barata.

El número de febrero de 1964 de *Datamation* revisaba el estado del arte de la microprogramación. No había ningún sistema microprogramado de uso generalizado en aquella época, y uno de los artículos, [HILL64], resumía la opinión popular del momento de que el futuro de la microprogramación «es un tanto turbio. Ninguno de los grandes fabricantes ha mostrado interés en la técnica, a pesar de que, probablemente, todos la hayan analizado.»

La situación cambió completamente en muy pocos meses. El System/360 de IBM se anunció en abril, y todos los modelos, excepto los más grandes, eran microprogramados. Aunque la serie 360 precedió a la disponibilidad de ROM de semiconductores, las ventajas de la microprogramación fueron lo bastante convincentes para IBM, como para introducir este cambio. Desde entonces, la microprogramación se ha convertido en un instrumento popular, usado en diversas aplicaciones, una de las cuales es la implementación de la unidad de control de un procesador. Esta aplicación se examina en este capítulo.

## 15.1. CONCEPTOS BÁSICOS

### MICROINSTRUCCIONES

La unidad de control parece un dispositivo bastante sencillo. Sin embargo, implementar una unidad de control como una interconexión de elementos lógicos básicos no es una tarea fácil. El diseño debe incluir lógica para realizar el secuenciamiento de las microoperaciones, para ejecutar las microoperaciones, para interpretar los códigos de operación, y para tomar decisiones basadas en los indicadores de la ALU. Todo este hardware es difícil de diseñar y de verificar. Además, el diseño es relativamente inflexible. Por ejemplo, es difícil cambiar el diseño si se desea añadir una nueva instrucción máquina.

Existe una alternativa, bastante común en los procesadores CISC de hoy día, que es realizar una unidad de control microprogramada.

Consideremos de nuevo la Tabla 14.1. Además de indicar las señales de control, cada microoperación está escrita en notación simbólica. Esta notación tiene toda la apariencia de un lenguaje de programación. De hecho es un lenguaje, conocido como *lenguaje de microprogramación*. Cada línea describe un conjunto de microoperaciones que suceden a la vez, y se conoce como *microinstrucción*. Una secuencia de instrucciones se conoce como *microprograma*, o *firmware*. Este último término refleja el hecho de que un microprograma está a mitad de camino entre hardware y software. Es más fácil diseñar en firmware que en hardware, pero es más difícil escribir un programa firmware que un programa software.

¿Cómo se puede usar el concepto de microprogramación para implementar una unidad de control? Consideremos que, para cada microoperación, todo lo que la unidad de control puede hacer es generar un conjunto de señales de control, de modo que, para cualquier microoperación, cada línea de control procedente de la unidad de control está activa o inactiva. Esta condición puede, naturalmente, representarse con un dígito binario para cada línea de control. De este modo, podríamos construir una *palabra de control* en la que cada bit representaría una línea de control. Entonces, cada microoperación se representaría mediante un patrón diferente de 1s y 0s en la palabra de control.

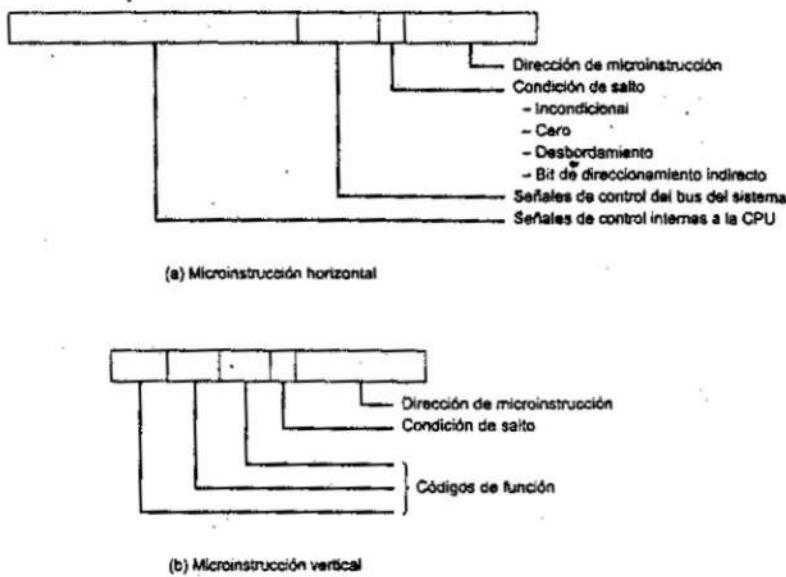


Figura 15.1. Formatos de microinstrucción típicos.

Supongamos que se ensarta una secuencia de palabras de control para representar la secuencia de microoperaciones ejecutadas por la unidad de control. A continuación, hemos de admitir que la secuencia de microoperaciones no es fija. Algunas veces tenemos un ciclo indirecto; otras no. Por tanto, coloquemos nuestras palabras de control en una memoria, cada palabra en una dirección única. Añadimos ahora un campo de dirección a cada palabra de control, indicando la posición de la siguiente palabra de control a ejecutar si una determinada condición es cierta (por ejemplo, que el bit de direccionamiento indirecto de una instrucción que referencia la memoria sea 1). Y añadimos además algunos bits para especificar la condición.

El resultado se conoce como *microinstrucción horizontal*; se muestra un ejemplo en la Figura 15.1a. El formato de la microinstrucción o palabra de control, es el siguiente. Hay un bit para cada línea de control interna al procesador, y un bit para cada línea de control del sistema. Hay un campo de condición que indica la condición bajo la cual debe producirse un salto, y hay un campo con la dirección de la microinstrucción a ejecutar cuando el salto se produzca. Esta microinstrucción se interpreta como sigue:

1. Para ejecutar la microinstrucción, se activan todas las líneas de control con el bit correspondiente a 1, y se dejan inactivas todas las líneas de control indicadas con un bit a 0. Las señales de control resultantes harán que se ejecuten una o más microoperaciones.
2. Si la condición indicada por los bits de condición es falsa, se ejecuta la siguiente microinstrucción secuencial.
3. Si la condición indicada por los bits de condición es cierta, la siguiente microinstrucción a ejecutar se indica en el campo de dirección.



Figura 15.2. Organización de la memoria de control.

La Figura 15.2 muestra cómo se pueden organizar estas palabras de control o microinstrucciones en una *memoria de control*. Las microinstrucciones de cada rutina se ejecutarán secuencialmente. Cada rutina termina con una instrucción de bifurcación o salto, indicando a dónde ir a continuación. Hay una sección especial del ciclo de ejecución cuyo único objetivo es indicar cuál de las rutinas de las instrucciones máquina (AND, ADD, etc.) se va a ejecutar a continuación, en función del código de operación actual.

La memoria de control de la Figura 15.2 es una descripción concisa de todo lo que hace la unidad de control. Define la secuencia de microoperaciones a realizar en cada ciclo (captación, indirecto, ejecución e interrupción), y especifica el secuenciamiento de estos ciclos. Si sólo fuera eso, esta notación sería un recurso útil para documentar el funcionamiento de una unidad de control para un computador particular. Pero es más que eso: es también una forma de implementar la unidad de control.

## UNIDAD DE CONTROL MICROPROGRAMADA

La memoria de control de la Figura 15.2 contiene un programa que describe el funcionamiento de la unidad de control. Resulta que podemos implementar la unidad de control sencillamente, suministrando los mecanismos necesarios para la ejecución de ese programa.

La Figura 15.3 muestra los elementos más importantes de esta implementación. El conjunto de microinstrucciones se almacena en la *memoria de control*. El *registro de dirección de control* (control address register, CAR) contiene la dirección de la siguiente microinstrucción a leer. Cuando se lee una microinstrucción de la memoria de control, se transfiere al

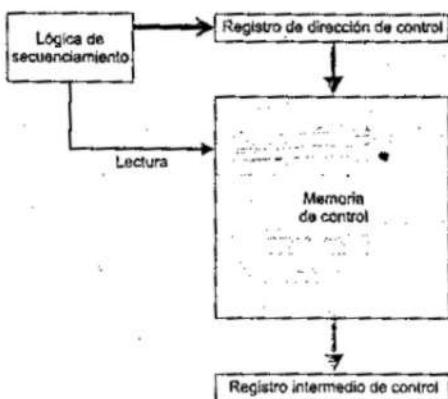


Figura 15.3. Microarquitectura de una unidad de control.

*registro intermedio de control* (control buffer register, CBR). La parte izquierda de ese registro (véase Figura 15.1a) se conecta a las líneas de control que salen de la unidad de control. De este modo, leer una microinstrucción de la memoria de control es lo mismo que ejecutar la microinstrucción. El tercer elemento que muestra la figura es una unidad de secuenciamento, que carga el registro de dirección de control y emite una orden de lectura.

Examinemos esta estructura con mayor detalle, como se representa en la Figura 15.4. Comparándola con la Figura 14.4, vemos que la unidad de control sigue teniendo las mismas entradas (IR, indicadores de la ALU y reloj) y salidas (señales de control). La unidad de control funciona como sigue:

1. Para ejecutar una instrucción, la unidad lógica de secuenciamento emite una orden de lectura a la memoria de control.
2. La palabra cuya dirección se especifica en el registro de dirección de control, se lee en el registro intermedio de control.
3. El registro intermedio de control genera las señales de control y contiene, además, la información de dirección siguiente para la unidad lógica de secuenciamento.
4. La unidad lógica de secuenciamento carga en el registro de dirección de control una nueva dirección, basada en la información de dirección siguiente del registro intermedio de control y en los indicadores de la ALU.

Todo esto sucede durante un pulso de reloj.

El último paso recién mencionado requiere cierta elaboración. Al final de la ejecución de cada microinstrucción, la unidad lógica de secuenciamento carga una nueva dirección en el registro de dirección de control. Dependiendo del valor de los indicadores de la ALU y del registro intermedio de control, se toma una de las tres siguientes decisiones:

- Captar la microinstrucción siguiente: Se suma 1 al registro de dirección de control.
- Saltar a una nueva rutina según indica una microinstrucción de salto: El campo de dirección del registro intermedio de control se carga en el registro de dirección de control.

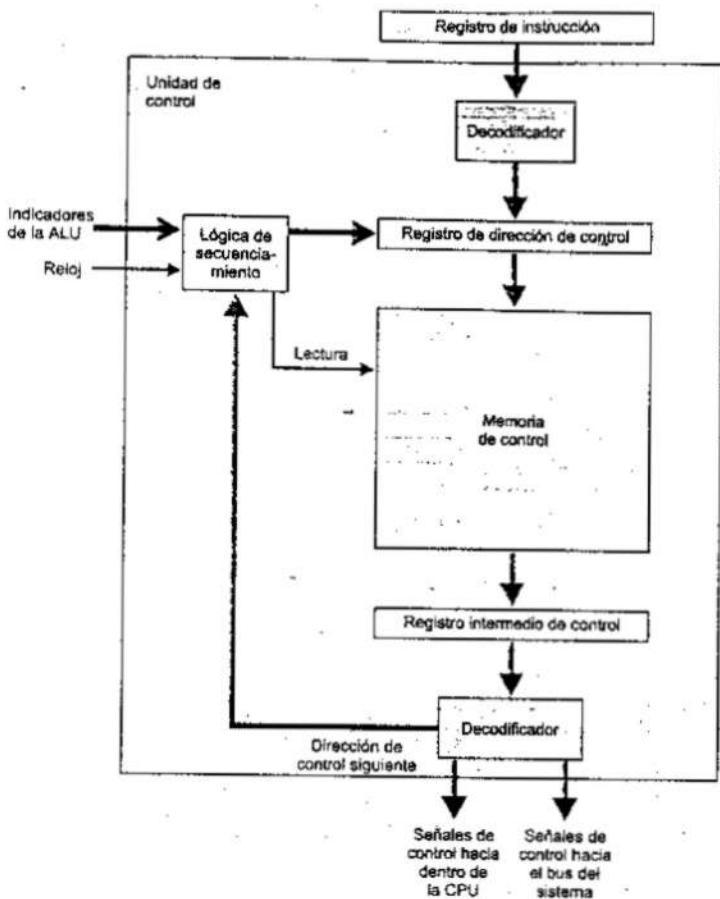


Figura 15.4. Funcionamiento de una unidad de control microprogramada.

- Saltar a la rutina de una instrucción máquina: Se carga el registro de dirección de control en función del código de operación almacenado en IR.

La Figura 15.4 muestra dos módulos, designados como *decodificadores*. El decodificador de arriba traduce el código de operación de IR en una dirección de memoria de control. El decodificador de abajo no se usa con microinstrucciones horizontales pero si con *microinstrucciones verticales* (Figura 15.1b). Como se mencionó, en una microinstrucción horizontal cada bit del campo de control corresponde a una línea de control. En una microinstrucción vertical, se usa un código para cada acción a realizar —por ejemplo,  $MAR \leftarrow (PC)$ —, y el decodificador traduce este código a señales de control individuales. La ventaja de las microinstrucciones verticales es que son más compactas (ocupan menos bits) que las microinstrucciones horizontales, a costa de añadir una pequeña lógica y cierto retardo de tiempo.

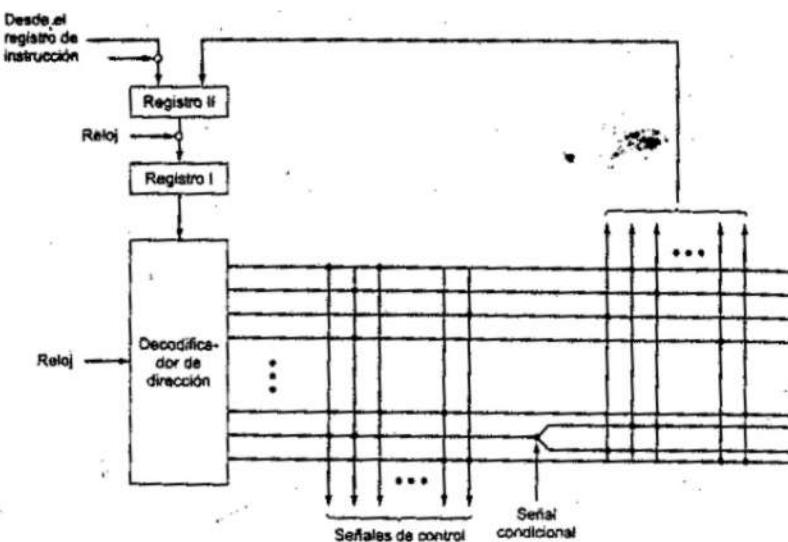


Figura 15.5. Unidad de control microprogramada de Wilkes.

## CONTROL DE WILKES

Como se mencionó, Wilkes fue el primero que propuso la utilización de una unidad de control microprogramada en 1951 [WILK51]. Más tarde, elaboró su propuesta en un diseño más detallado [WILK53]. Es instructivo examinar esta propuesta inicial.

Wilkes estaba interesado en el desarrollo de una aproximación sistemática al diseño de una unidad de control. La configuración que propuso se representa en la Figura 15.5. El núcleo del sistema es una matriz parcialmente llena de diodos. Durante un ciclo máquina, se activa una fila de la matriz mediante un pulso. Esto produce señales en aquellos puntos en los que un diodo está presente (indicados mediante un punto en el diagrama). La primera parte de la fila genera las señales de control que gobiernan el funcionamiento del procesador. La segunda parte genera la dirección de la fila que será seleccionada mediante un pulso en el siguiente ciclo máquina. Por tanto, cada fila de la matriz es una microrinstrucción, y el trazado de la matriz es la memoria de control.

Al comienzo de un ciclo, la dirección de la fila a seleccionar está almacenada en el registro I. Esta dirección es la entrada del decodificador, el cual, cuando se activa mediante un pulso de reloj, selecciona una fila de la matriz. Dependiendo de las señales de control, durante el ciclo, se pasa al registro II el código de operación almacenado en el registro de instrucción, o bien la segunda parte de la fila activada. El contenido del registro II se lleva entonces al registro I mediante un pulso de reloj. Se usan pulsos de reloj alternos para activar una fila de la matriz y para transferir el contenido del registro II al registro I. La configuración con dos registros es necesaria, ya que el decodificador es sencillamente un circuito combinacional; con un único registro, la salida se convertiría en entrada dentro del mismo ciclo, originando un estado inestable.

Este esquema es muy similar al enfoque de microprogramación horizontal descrito anteriormente (Figura 15.1a). La principal diferencia es ésta: en aquella descripción, el registro de dirección de control podía incrementarse en 1 para acceder a la siguiente dirección. En el esquema de Wilkes, la dirección siguiente está contenida en la microinstrucción. Para permitir bifurcaciones, una fila debe contener dos partes de direcciones, controladas por una señal condicional (por ejemplo, un indicador), como muestra la figura.

Después de proponer este esquema, Wilkes proporciona un ejemplo de su utilización para implementar la unidad de control de una máquina sencilla. Este ejemplo, el primer diseño conocido de un procesador microprogramado, merece repetirse aquí, porque ilustra muchos de los principios actuales de la microprogramación.

El procesador de la máquina hipotética incluye los siguientes registros:

- A Multiplicando.
- B Acumulador (mitad menos significativa).
- C Acumulador (mitad más significativa).
- D Registro de desplazamiento.

Además, hay tres registros y dos indicadores de 1 bit, accesibles sólo por la unidad de control. Los registros son los siguientes:

- E Sirve como registro de dirección de memoria (MAR) y como almacenamiento temporal.
- F Contador de programa.
- G Otro registro temporal, usado en cálculos.

La Tabla 15.1 muestra una relación del repertorio de instrucciones máquina para este ejemplo. La Tabla 15.2 contiene el repertorio completo de microinstrucciones, expresadas de forma simbólica, que implementa la unidad de control. Se necesita solamente un total de 38 microinstrucciones para definir completamente el sistema.

La primera columna contiene la dirección (número de fila) de cada microinstrucción. Las direcciones referentes a códigos de operación van precedidas de la letra correspondiente a la instrucción. De este modo, cuando se encuentra el código de operación de la instrucción de

Tabla 15.1. Repertorio de instrucciones máquina del ejemplo de Wilkes

| Orden | Efecto de la orden                                                                                               |
|-------|------------------------------------------------------------------------------------------------------------------|
| A n   | $C(Ac) + C(n)$ al $Ac$ ,                                                                                         |
| S n   | $C(Ac) - C(n)$ al $Ac$ ,                                                                                         |
| H n   | $C(n)$ al $Ac_1$                                                                                                 |
| V n   | $C(Ac_1) \times C(n)$ al $Ac_1$ , donde $C(n) \geq 0$                                                            |
| T n   | $C(Ac_1)$ a $n$ , 0 al $Ac$                                                                                      |
| U n   | $C(Ac_1)$ a $n$                                                                                                  |
| R n   | $C(Ac) \times 2^{-n+1}$ al $Ac$                                                                                  |
| L n   | $C(Ac) \times 2^{n+1}$ al $Ac$ <del>+</del>                                                                      |
| G n   | Si $C(Ac) < 0$ , transferir el control a $n$ ; si $C(Ac) \geq 0$ , ignorar (es decir, continuar secuencialmente) |
| I n   | Leer el siguiente carácter del mecanismo de entrada en $n$                                                       |
| O n   | Enviar $C(n)$ al mecanismo de salida                                                                             |

Notación:  $Ac$  = acumulador

$Ac_1$  = mitad más significativa del acumulador

$Ac_2$  = mitad menos significativa del acumulador

$n$  = posición de memoria  $n$

$C(X)$  = contenido de  $X$  ( $X$  = registro o posición de almacenamiento)

Tabla 15.2. Microinstrucciones del ejemplo de Wilkes

Notación: Las letras A, B, C, etc., representan los diversos registros de las unidades aritmética y de registros de control. «C a D» indica que los circuitos de conmutación conectan la salida del circuito C a la entrada del registro D; «(D + A) a C» indica que la salida del registro A se conecta a una entrada de la unidad de suma (la salida de D está conectada permanentemente a la otra entrada); y la salida del sumador al registro C. Un símbolo numérico n entre comillas (por ejemplo, '1') representa la fuente, cuya salida es el número n en unidades del dígito menos significativo.

|      | Unidad aritmética | Unidad de registros de control | Biestable condicional | Microinstrucción siguiente |
|------|-------------------|--------------------------------|-----------------------|----------------------------|
|      |                   | Ajuste                         | Uao                   | 0 1                        |
| 0    |                   | F a G y E                      |                       | 1                          |
| 1    |                   | (G + '1') a F                  |                       | 2                          |
| 2    |                   | Memoria a G                    |                       | 3                          |
| 3    |                   | G a E                          |                       | 4                          |
| 4    |                   | E a decodificador              |                       | —                          |
| A 5  | C a D             |                                |                       | 16                         |
| S 6  | C a D             |                                |                       | 17                         |
| H 7  | Memoria a B       |                                |                       | 0                          |
| V 8  | Memoria a A       |                                |                       | 27                         |
| T 9  | C a Memoria       |                                |                       | 25                         |
| U 10 | C a Memoria       |                                |                       | 0                          |
| R 11 | B a D             | E a G                          |                       | 19                         |
| L 12 | C a D             | E a G                          |                       | 22                         |
| G 13 |                   | E a G                          | (1)C,                 | 18                         |
| I 14 | Entrada a Memoria |                                |                       | 0                          |
| O 15 | Memoria a salida  |                                |                       | 0                          |
| 16   | (D + Memoria) a C |                                |                       | 0                          |
| 17   | (D - Memoria) a C |                                |                       | 0                          |
| 18   |                   |                                |                       | 1 0 1                      |
| 19   | D a B (R)*        | (G - '1') a E                  |                       | 20                         |
| 20   | C a D             |                                | (1)E,                 | 21                         |
| 21   | D a C (R)         |                                |                       | 1 11 0                     |
| 22   | D a C (L)*        | (G - '1') a E                  |                       | 23                         |
| 23   | B a D             |                                | (1)E,                 | 24                         |
| 24   | D a B (L)         |                                |                       | 1 12 0                     |
| 25   | '0' a B           |                                |                       | 26                         |
| 26   | B a C             |                                |                       | 0                          |
| 27   | '0' a C           | '18' a E                       |                       | 28                         |

Tabla 15.2. Microinstrucciones del ejemplo de Wilkes (continuación)

|    | Unidad aritmética       | Unidad de registros de control | Biestable condicional | Microinstrucción siguiente |
|----|-------------------------|--------------------------------|-----------------------|----------------------------|
| 28 | $B \rightarrow D$       | $E \rightarrow G$              | (1)B <sub>1</sub>     | 29                         |
| 29 | $D \rightarrow B(R)$    | $(G - '1') \rightarrow E$      |                       | 30                         |
| 30 | $C \rightarrow D(R)$    |                                | (2)E <sub>1</sub>     | 31 32                      |
| 31 | $D \rightarrow C$       |                                |                       | 2 28 33                    |
| 32 | $(D + A) \rightarrow C$ |                                |                       | 2 28 33                    |
| 33 | $B \rightarrow D$       |                                | (1)B <sub>1</sub>     | 34                         |
| 34 | $D \rightarrow B(R)$    |                                |                       | 35                         |
| 35 | $C \rightarrow D(R)$    |                                |                       | 1 36 37                    |
| 36 | $D \rightarrow C$       |                                |                       | 0                          |
| 37 | $(D - A) \rightarrow C$ |                                |                       | 0                          |

\* Desplazamiento a la derecha («Right shift»). Los circuitos de conmutación de la unidad aritmética están organizados de tal forma que el dígito menos significativo del registro C se lleva a la posición más significativa del registro B durante las microoperaciones de desplazamiento a la derecha, y el dígito más significativo del registro C (dígito de signo) se repite (realizándose, por tanto, la corrección para números negativos).

\* Desplazamiento a la izquierda («Left shift»). Los circuitos de conmutación están dispuestos de manera similar, para pasar el dígito más significativo del registro B a la posición menos significativa del registro C durante las microoperaciones de desplazamiento a la izquierda.

suma (A), se ejecuta la microinstrucción de la posición 5. Las columnas 2 y 3 expresan las acciones a realizar por la ALU y la unidad de control, respectivamente. Cada expresión simbólica ha de traducirse en un conjunto de señales de control (bits de la microinstrucción). Las columnas 4 y 5 tienen que ver con la modificación y el uso de los dos indicadores (biestables). La columna 4 especifica la señal que ajusta el indicador. Por ejemplo, (1)C, significa que el indicador número 1 se ajusta según el bit de signo del número contenido en el registro C. Si la columna 5 contiene un identificador de indicador, entonces las columnas 6 y 7 contienen las dos direcciones de microinstrucción alternativas; si no, la columna 6 especifica la dirección de la siguiente microinstrucción a captar.

Las instrucciones de la 0 a la 4 constituyen el ciclo de captación. La microinstrucción 4 presenta el código de operación a un decodificador, que genera la dirección de la microinstrucción a captar correspondiente a la instrucción máquina en curso. El lector debería ser capaz de deducir el funcionamiento completo de la unidad de control a partir de un cuidadoso estudio de la Tabla 15.2.

## VENTAJAS E INCONVENIENTES

La ventaja principal que aporta el uso de la microprogramación para implementar una unidad de control, es que simplifica el diseño de ésta, resultando así su implementación más barata y menos propensa a errores. Una unidad de control *cableada* contendrá lógica compleja para hacer el secuenciamiento a través de las muchas microoperaciones del ciclo de instrucción. Por otra parte, los decodificadores y la unidad lógica de secuenciamiento de una unidad de control microprogramada son elementos lógicos muy sencillos.

El principal inconveniente de una unidad microprogramada es que será algo más lenta que una unidad cableada de tecnología comparable. A pesar de ello, la microprogramación es la técnica dominante para implementar unidades de control en los CISC actuales, debido a su facilidad de implementación. Los procesadores RISC, dado que tienen un formato de instrucción más sencillo, emplean normalmente unidades de control cableadas. Examinaremos ahora con más detalle la aproximación microprogramada.

## 15.2. SECUENCIAMIENTO DE MICROINSTRUCCIONES

Las dos tareas básicas realizadas por una unidad de control microprogramada son las siguientes:

- **Secuenciamiento de microinstrucciones:** Obtener la siguiente microinstrucción de la memoria de control.
- **Ejecución de microinstrucciones:** Generar las señales de control necesarias para ejecutar la microinstrucción.

Al diseñar una unidad de control, las dos tareas deben considerarse a la vez, ya que ambas afectan al formato de la microinstrucción y a la temporización de la unidad de control. En esta sección, nos centraremos en el secuenciamiento, y hablaremos lo mínimo posible sobre los temas de formato y temporización. Estos temas se examinarán más detalladamente en la sección siguiente.

### CONSIDERACIONES RESPECTO AL DISEÑO

Hay dos cuestiones involucradas en el diseño de una técnica de secuenciamiento de microinstrucciones: el tamaño de la microinstrucción, y el tiempo de generación de la dirección. El primer asunto es evidente: minimizar el tamaño de la memoria de control reduce su coste. El segundo asunto es sencillamente un deseo de ejecutar las microinstrucciones tan rápido como sea posible.

Cuando se ejecuta un microprograma, la dirección de la siguiente microinstrucción a ejecutar está en una de estas situaciones:

- Viene determinada por el registro de instrucción.
- Es la siguiente dirección secuencial.
- Es el destino de un salto.

La primera situación tiene lugar sólo una vez por ciclo de instrucción, justo tras la captación de la instrucción. La segunda situación es la más común en la mayoría de los diseños. No obstante, el diseño no se puede optimizar sólo para los accesos secuenciales. Los saltos, tanto condicionales como incondicionales, son una parte necesaria del microprograma. Además, las secuencias de microinstrucciones tienden a ser cortas; una de cada tres o cuatro microinstrucciones podría ser un salto [SIEW82]. Por consiguiente, es importante diseñar técnicas compactas y eficientes en cuanto al tiempo, para los saltos a microinstrucciones.

### TÉCNICAS DE SECUENCIAMIENTO

A partir de la microinstrucción en curso, de los indicadores de condición y del contenido del registro de instrucción, hay que generar una dirección de la memoria de control para la

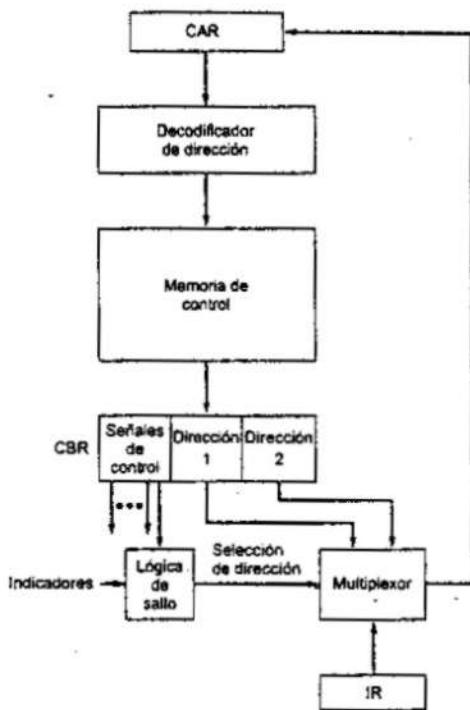


Figura 15.6. Lógica de control de salto; dos campos de dirección.

siguiente microinstrucción. Se han usado una gran variedad de técnicas. Podemos agruparlas en tres categorías, como ilustran las Figuras 15.6 a 15.8. Estas categorías se basan en el formato de la información de dirección de la microinstrucción:

- Dos campos de dirección
- Un único campo de dirección
- Formato variable

La técnica más sencilla es tener dos campos de dirección en cada microinstrucción. La Figura 15.6 indica cómo se va a usar esta información. Se tiene un multiplexor, que sirve de destino de los dos campos de dirección y del registro de instrucción. Basándose en la entrada de selección de dirección, el multiplexor transmite el código de operación, o una de las dos direcciones, al registro de dirección de control (CAR). El CAR se decodifica a continuación para producir la dirección de la siguiente microinstrucción. Las señales de selección de dirección son suministradas por un módulo de lógica de salto, cuyas entradas son los indicadores de la unidad de control y ciertos bits de la parte de control de la microinstrucción.

Aunque el método de dos direcciones es sencillo, necesita más bits por microinstrucción que las otras técnicas. Con alguna lógica adicional, se puede conseguir algún ahorro.

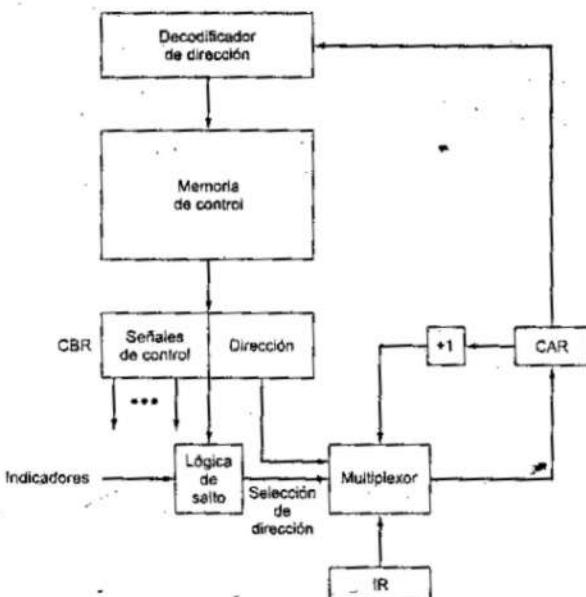


Figura 15.7. Lógica de control de salto; un único campo de dirección.

Una aproximación frecuente es tener un único campo de dirección (Figura 15.7). Con este enfoque, las opciones para la dirección siguiente son:

- Campo de dirección.
- Código del registro de instrucción.
- Siguiente dirección secuencial.

Las señales de selección de dirección determinan qué opción se escoge. Esta técnica reduce el número de campos de dirección a uno. Observemos, sin embargo, que el campo de dirección a menudo no se usa. Por tanto, hay alguna ineficiencia en este esquema de codificación.

Otro método es proporcionar dos formatos de microinstrucción totalmente diferentes (Figura 15.8). Un bit designa qué formato se utilizará. En uno de los dos formatos, los demás bits se usan para activar señales de control. En el otro formato, algunos bits controlan el módulo de lógica de salto, y los bits restantes suministran la dirección. En el primer formato, la dirección siguiente es la siguiente dirección secuencial, o una dirección derivada del registrador de instrucción. En el segundo formato, se especifica un salto condicional o incondicional. Un inconveniente de esta aproximación, tal como se ha descrito, es que se consume un ciclo completo por cada microinstrucción de salto. Con las otras técnicas, la generación de la dirección sucede como parte del mismo ciclo en el que se generan las señales de control, con lo que se minimizan los accesos a la memoria de control.

Las aproximaciones que se acaban de describir son generales. Las implementaciones específicas usarán con frecuencia una variación o una combinación de estas técnicas.

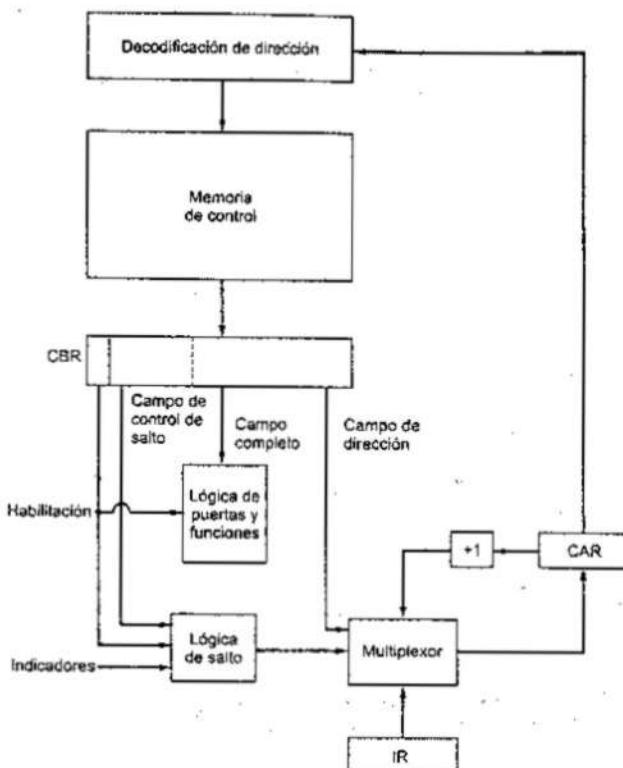


Figura 15.8. Lógica de control de salto; formato variable.

## GENERACIÓN DE DIRECCIONES

Hemos enfocado el problema del secuenciamiento desde el punto de vista de las consideraciones sobre el formato y de los requisitos de lógica en general. Otro punto de vista es considerar las diversas formas de obtener o calcular la siguiente dirección.

La Tabla 15.3 relaciona las diversas técnicas de generación de la dirección. Éstas se pueden dividir en técnicas explícitas, en las que la dirección aparece explícitamente en la microinstrucción, y técnicas implícitas, que requieren lógica adicional para generar la dirección.

Tabla 15.3. Técnicas de generación de direcciones de microinstrucción

| Explícitas                                             | Implícitas                                |
|--------------------------------------------------------|-------------------------------------------|
| Dos campos<br>Salto incondicional<br>Salto condicional | Traducción<br>Adición<br>Control residual |

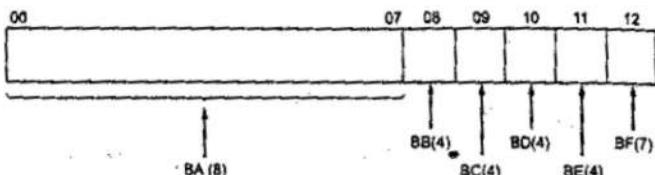


Figura 15.9. Registro de dirección de control del IBM 3033.

Nos hemos ocupado esencialmente de las técnicas explícitas. Con un enfoque de dos pos. hay dos direcciones alternativas disponibles en cada microinstrucción. Usando un único campo de dirección o un formato variable, se pueden implementar varias instrucciones salto. Una instrucción de bifurcación condicional depende de los siguientes tipos de información:

- Indicadores de la ALU.
- Parte del código de operación o campos de modo de direccionamiento de la instrucción máquina.
- Partes de un registro seleccionado, tales como el bit de signo.
- Bits de estado dentro de la unidad de control.

También se usan frecuentemente algunas técnicas implícitas. Una de ellas, la traducción, se necesita en casi todos los diseños. La parte de código de operación de una instrucción máquina se traduce a una dirección de microinstrucción. Esto ocurre sólo una vez por ciclo de instrucción.

Una técnica implícita habitual consiste en combinar o sumar dos partes de una dirección para formar la dirección completa. Este procedimiento fue adoptado por la familia IBM S/360 [TUCK67], y usado por muchos de los modelos S/370. Usaremos el IBM 3033 como ejemplo.

El registro de dirección de control del IBM 3033, de 13 bits, se ilustra en la Figura 15.9. Se pueden distinguir dos partes en la dirección. Los 8 bits más significativos (00-07) no cambian, normalmente, de un ciclo de microinstrucción al siguiente. Durante la ejecución de una microinstrucción, estos 8 bits se copian directamente desde un campo de 8 bits de la microinstrucción (el campo BA), en los 8 bits más significativos del registro de dirección de control. Ello define un bloque de 32 microinstrucciones en la memoria de control. Los otros 5 bits del registro de dirección de control se ajustan para especificar la dirección concreta de la microinstrucción a captar a continuación. Cada uno de estos bits viene determinado por un campo de 4 bits (excepto uno por un campo de 7 bits) de la microinstrucción en curso; cada campo especifica la condición para ajustar el bit correspondiente. Por ejemplo, un bit del registro de dirección de control puede ponerse a 1 o a 0, dependiendo de si se produjo acarreo en la última operación de la ALU.

La última técnica de la lista de la Tabla 15.3 se denomina *control residual*. Esta aproximación implica el uso de una dirección de microinstrucción guardada previamente en un almacenamiento temporal dentro de la unidad de control. Por ejemplo, algunos repertorios de microinstrucciones están dotados de la posibilidad de hacer llamadas a subrutinas. Un registro interno o una pila de registros se usan para guardar las direcciones de retorno. Un ejemplo de esta técnica aparece en el LSI-11, que examinamos ahora.

## SECUENCIAMIENTO DE MICROINSTRUCCIONES EN EL LSI-11

El microcomputador LSI-11 es una versión del PDP-11, con los componentes principales de sistema en una misma tarjeta. El LSI-11 está implementado usando una unidad de control microprogramada [SEBE76].

El LSI-11 utiliza microinstrucciones de 22 bits, y una memoria de control de 2K palabras de 22 bits. La dirección de la siguiente microinstrucción se determina de una de estas cinco formas:

- Dirección secuencial siguiente: En ausencia de otras instrucciones, el registro de dirección control de la unidad de control se incrementa en 1.
- Traducción del código de operación: Al comienzo de cada ciclo de instrucción, la dirección de la siguiente microinstrucción viene determinada por el código de operación.
- Llamada/retorno de subrutina: Explicada más abajo.
- Comprobación de interrupciones: Ciertas microinstrucciones especifican una comprobación de interrupciones. Si ha ocurrido una interrupción, esto determina la dirección de la siguiente microinstrucción.
- Salto: Se usan microinstrucciones de salto condicional e incondicional.

Se proporciona la posibilidad de subrutinas de un nivel. Un bit de cada microinstrucción se dedica a esta tarea. Cuando el bit está a uno, un registro de retorno de 11 bits se carga con el contenido actualizado del registro de dirección de control. Una instrucción posterior que especifique un retorno, hará que se cargue el registro de dirección de control con el contenido del registro de retorno.

El retorno es una forma de microinstrucción de salto incondicional. Otra forma de salto incondicional hace que se carguen los bits del registro de dirección de control con 11 bits de la microinstrucción. La microinstrucción de salto condicional hace uso de un código de comprobación de 4 bits dentro de la microinstrucción. Este código especifica la comprobación de diversos códigos de condición de la ALU, para determinar la decisión de salto. Si la condición no es cierta, se escoge la siguiente dirección secuencial. Si es cierta, los 8 bits menos significativos del registro de dirección de control se cargan con 8 bits de la microinstrucción. Esto permite el salto dentro de una página de memoria de 256 palabras.

Como puede observarse, el LSI-11 incluye un potente secuenciamiento de direcciones dentro de la unidad de control. Ello da al microprogramador una flexibilidad apreciable, y puede facilitar la tarea de la micropogramación. Por otra parte, esta aproximación requiere más lógica en la unidad de control que otra con menores capacidades.

### 15.3. EJECUCIÓN DE MICROINSTRUCCIONES

El ciclo de microinstrucción es el evento básico de un procesador microprogramado. Cada ciclo de compone de dos partes: captación y ejecución. La parte de captación depende de la generación de una dirección de microinstrucción, que fue tratada en la sección precedente. Esta sección se ocupa de la ejecución de una microinstrucción.

Recordemos que el resultado de la ejecución de una microinstrucción es la generación de señales de control. Algunas de estas señales controlan puntos internos del procesador. Las demás señales van al bus de control externo o a otras interfaces externas. Como una función accesoria, se determina la dirección de la siguiente microinstrucción.

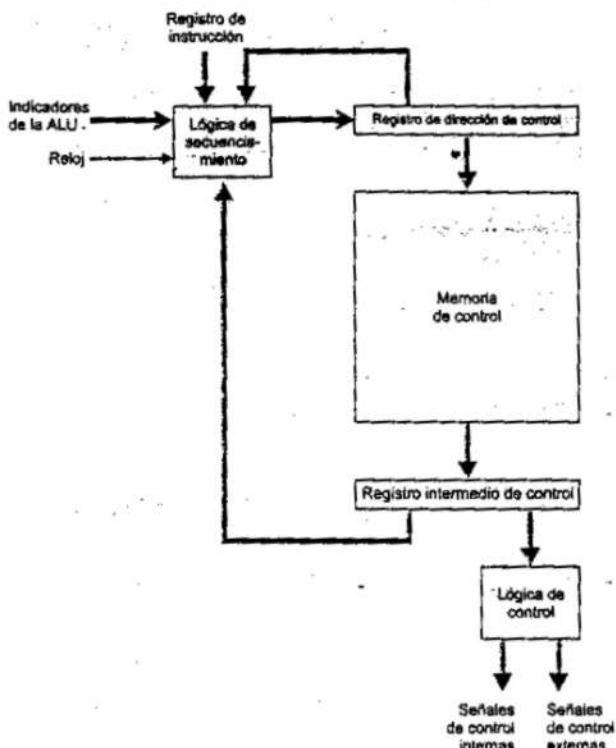


Figura 15.10. Organización de la unidad de control.

La descripción precedente sugiere la organización de una unidad de control, que se muestra en la Figura 15.10. Esta versión ligeramente revisada de la Figura 15.4, subraya el centro de atención de esta sección. Los principales módulos de este diagrama ya deben estar claros. El módulo de lógica de secuenciamiento contiene la lógica que realiza las funciones estudiadas en la sección anterior. Genera la dirección de la siguiente microinstrucción, usando como entradas el registro de instrucción, los indicadores de la ALU, el registro de dirección de control (para incrementarlo), y el registro intermedio de control. El último puede proporcionar una dirección real, bits de control, o ambos. Este módulo está controlado por un reloj, que determina la temporización del ciclo de instrucción.

El módulo de lógica de control genera las señales de control en función de algunos de los bits de la microinstrucción. Debería quedar claro que el formato y contenido de la microinstrucción determinará la complejidad del módulo de lógica de control.

### UNA TAXONOMÍA DE LAS MICROINSTRUCCIONES

Las microinstrucciones se pueden clasificar de varias formas. Las distinciones que generalmente se hacen en la bibliografía incluyen:

- Vertical/horizontal
- Empaquetada/no empaquetada
- Microprogramación hard/soft
- Codificación directa/indirecta

Todas ellas se refieren al formato de la microinstrucción. Ninguno de estos términos se ha usado de una manera coherente y precisa en la bibliografía. No obstante, un examen de estas parejas de cualidades sirve para aclarar las diferentes alternativas en el diseño de las microinstrucciones. En los siguientes párrafos, consideraremos primeramente el principal asunto, referénte al diseño, que subyace en todas estas parejas de características alternativas, y después consideraremos los conceptos que cada pareja de alternativas sugiere.

En la propuesta original de Wilkes [WILK51], cada bit de una microinstrucción producía directamente una señal de control o un bit de la dirección siguiente. Hemos visto, en la sección precedente, que son posibles esquemas de secuenciamiento de direcciones más complejos, que usan menos bits por microinstrucción. Tales esquemas requieren un módulo de lógica de secuenciamiento más complejo. Existe un tipo de compromiso semejante para la parte de la microinstrucción que atañe a las señales de control. Se pueden ahorrar bits de la palabra de control codificando la información de control, y decodificándola más tarde para producir las señales de control.

¿Cómo puede hacerse esta codificación? Para responder a esta pregunta, consideremos que hay un total de  $K$  señales de control internas y externas diferentes que tiene que generar la unidad de control. En el esquema de Wilkes se dedicarían a este propósito  $K$  bits de la microinstrucción. Esto permite que se puedan generar las  $2^K$  combinaciones posibles de señales de control durante cualquier ciclo de instrucción. Pero somos capaces de hacerlo mejor, si observamos que no todas las combinaciones posibles se usarán. Veamos algunos ejemplos:

- Dos fuentes no se pueden llevar al mismo destino (por ejemplo,  $C_2$  y  $C_3$  en la Figura 14.5).
- Un registro no puede ser a la vez fuente y destino (por ejemplo,  $C_5$  y  $C_{12}$  en la Figura 14.5).
- Sólo un patrón de señales de control se puede presentar a la ALU cada vez.
- Sólo un patrón de señales de control se puede presentar al bus de control externo cada vez.

De este modo, para un procesador dado, puede hacerse una lista con todas las posibles combinaciones de señales de control admisibles, obteniendo un número de posibilidades  $Q < 2^K$ , que podrían codificarse con  $\log_2 Q$  bits, siendo  $(\log_2 Q) < K$ . Esta sería la forma más estricta posible de codificación que preserva todas las combinaciones permisibles de señales de control. En la práctica, este sistema de codificación no se usa, por dos razones:

- Es tan difícil de programar como un esquema decodificado puro (como el de Wilkes). Este punto se discutirá más a fondo dentro de poco.
- Requiere un módulo de lógica de control complejo y, por consiguiente, lento.

En lugar de eso, se adoptan algunos compromisos. Los hay de dos tipos:

- Se usan más bits de los estrictamente necesarios para codificar las posibles combinaciones.
- Algunas combinaciones que son físicamente permisibles no se pueden codificar.

El último tipo de compromiso tiene el efecto de reducir el número de bits. El resultado neto, sin embargo, es que se usan más bits que  $\log_2 Q$ .

En la siguiente subsección, discutiremos técnicas de codificación específicas. El resto de esta subsección se ocupa de los efectos de la codificación y de los diversos términos usado para describir esa codificación.

Basándonos en lo anterior, podemos ver que el campo de señales de control del formato de microinstrucción se encuadra dentro de un espectro. En un extremo, hay un bit para cada señal de control; en el otro extremo, se usa un formato muy codificado. La Tabla 15.4 muestra otras características de una unidad de control microprogramada que también se encuantran dentro de espectros de posibilidades y que, por lo general, dependen del espectro de grado de codificación.

La segunda pareja de características de la tabla es bastante evidente. El esquema puro de Wilkes es el que requiere más bits. También debería estar claro que este extremo ofrece la visión más detallada del hardware. El microprogramador maneja individualmente cada señal de control. La codificación se hace de tal modo, que se agrupan funciones o recursos, de manera que el microprogramador ve el procesador a un nivel más alto, menos detallado. Además, la codificación se diseña para facilitar la microprogramación. Por otra parte, debería quedar claro que la labor de comprender y orquestar el uso de todas las señales de control es difícil. Como se mencionó, una de las consecuencias típicas de la codificación es que impide el uso de ciertas combinaciones que serían permisibles si no existiera ésta.

El párrafo precedente discute el diseño de la microinstrucción desde el punto de vista del programador. Pero el grado de codificación también puede considerarse, viendo sus efectos sobre el hardware. Con un formato puro no codificado, se necesita poca o ninguna lógica de decodificación; cada bit genera una señal de control individual. Conforme se usan esquemas de codificación más compactos y globales, se necesita una lógica de decodificación más compleja. Esto puede afectar proporcionalmente a las prestaciones. Se necesita más tiempo para propagar las señales a través de las puertas de una lógica de control más compleja. Por tanto, la ejecución de microinstrucciones codificadas tarda más tiempo que la ejecución de las no codificadas.

De este modo, todas las características relacionadas en la Tabla 15.4 se distribuyen a lo largo de un espectro de estrategias de diseño. En general, un diseño que cae hacia el extremo izquierdo del espectro, se propone optimizar las prestaciones de la unidad de control. Los diseños del extremo derecho están más interesados en optimizar el proceso de microprogramación. En efecto, los repertorios de microinstrucciones cercanos al extremo derecho del espectro, se parecen mucho a los repertorios de instrucciones máquina. Un buen ejemplo de ello es el diseño del LSI-11, descrito más adelante en esta sección. Típicamente, cuando el objetivo es sencillamente implementar una unidad de control, el diseño estará cerca del extremo izquierdo del espectro. El diseño del IBM 3033, que se estudiará luego, está en esta categoría. Como veremos más adelante, algunos sistemas permiten que diferentes usuarios construyan microprogramas diferentes, usando el mismo tipo de microinstrucción. En este segundo caso, el diseño caerá probablemente cerca del extremo derecho del espectro.

Podemos ocuparnos ahora de alguna terminología introducida anteriormente. La Tabla 15.4 indica cómo tres de estas parejas de términos se relacionan con el espectro de las microinstrucciones. Fundamentalmente, cualquiera de estas parejas describe la misma cosa, pero da importancia a diferentes características de diseño.

El grado de empaquetamiento se relaciona con el grado de identificación entre una tarea de control y algunos bits específicos de la microinstrucción. Cuanto más empaquetados están los bits, un número dado de bits contiene más información. Por tanto, el empaquetamiento se relaciona con la codificación. Los términos *horizontal* y *vertical* se relacionan con el ancho relativo de la microinstrucción. [SIEW82] indica, a modo de regla empírica, que las microinstrucciones verticales tienen longitudes en el rango de 16 a 40 bits, y las microinstruccio-

Tabla 15.4. El espectro de las microinstrucciones

| Características                                                                                                                                                                                                                            |                                                                                                                                                                                                                                   |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Microinstrucción no codificada<br>Muchos bits<br>Visión detallada del hardware<br>Difícil de programar<br>Concurrencia explotada completamente<br>Poca o ninguna lógica de control<br>Ejecución rápida<br>Optimización de las prestaciones | Microinstrucción muy codificada.<br>Pocos bits<br>Visión global del hardware<br>Fácil de programar<br>Concurrencia no explotada completamente<br>Lógica de control compleja<br>Ejecución lenta<br>Optimización de la programación |
| Terminología                                                                                                                                                                                                                               |                                                                                                                                                                                                                                   |
| No empaquetada<br>Horizontal<br>Hard                                                                                                                                                                                                       | Empaquetada<br>Vertical<br>Soft                                                                                                                                                                                                   |

nes horizontales longitudes de 40 a 100 bits. Los términos microprogramación *hard* y *soft* se utilizan para indicar el grado de proximidad a las señales de control subyacentes y a la configuración del hardware. Los microprogramas «hard» generalmente son fijos y se sitúan en memoria de sólo lectura. Los microprogramas «soft» son más cambiables, y sugieren una programación por parte del usuario.

La otra pareja de términos mencionados al comienzo de esta subsección, se refiere a codificación directa frente a indirecta, un asunto al que volvemos ahora.

## CODIFICACIÓN DE LAS MICROINSTRUCCIONES

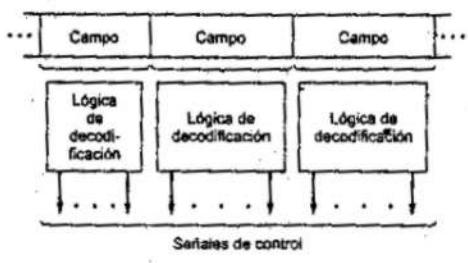
En la práctica, las unidades de control microprogramadas no se diseñan utilizando un formato de microinstrucción puro no codificado u horizontal. Se hace uso al menos de algún grado de codificación, para reducir el ancho de la memoria de control y simplificar la tarea de la microprogramación.

La técnica básica de codificación se ilustra en la Figura 15.11a. La microinstrucción se organiza como un conjunto de campos. Cada campo contiene un código que, tras la decodificación, activa una o más señales de control.

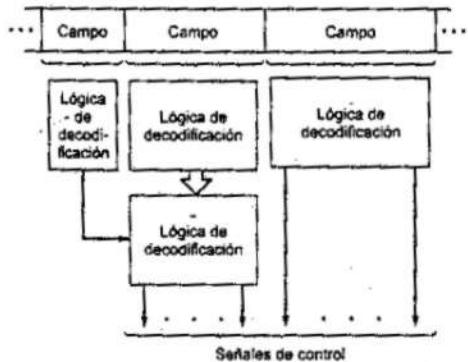
Consideremos las implicaciones de este esquema. Cuando la microinstrucción se ejecuta, cada campo se decodifica y genera señales de control. Así, con  $N$  campos, se especifican  $N$  acciones simultáneas. Cada acción se traduce en la activación de una o más señales de control. Generalmente, aunque no siempre, querremos diseñar el formato de modo que cada señal de control no pueda ser activada por más de un campo. Claramente, no obstante, debe ser posible activar cada señal de control al menos por un campo.

Consideremos ahora un campo individual. Un campo que conste de  $L$  bits puede contener uno de los  $2^L$  códigos posibles, cada uno de los cuales puede codificar un patrón diferente de señales de control. Como sólo puede aparecer un código en un campo en un momento dado, los códigos son mutuamente exclusivos y, por lo tanto, las acciones que ellos producen también lo son.

El diseño de un formato de microinstrucción codificado puede plantearse ahora en términos muy simples:



(a) Codificación directa



(b) Codificación indirecta

Figura 15.11. Codificación de la microinstrucción.

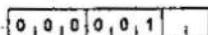
- Organizar el formato en campos independientes; es decir, cada campo representa un conjunto de acciones (patrón de señales de control), de forma que acciones de diferentes campos pueden ocurrir simultáneamente.
- Definir cada campo, de modo que las acciones alternativas que puede especificar ese campo sean mutuamente exclusivas; es decir, que sólo una de las acciones especificadas por un determinado campo pueda ocurrir en un momento dado.

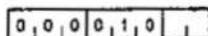
Se pueden usar dos planteamientos para la organización de la microinstrucción codificada en campos: funcional y por recursos. El método de *codificación funcional* identifica funciones dentro de la máquina, y designa los campos por el tipo de función. Por ejemplo, si se pueden utilizar varias fuentes para la transferencia de datos al acumulador, se puede utilizar un campo para este propósito, especificando cada código una fuente diferente. La *codificación por recursos* ve a la máquina como un conjunto de recursos independientes, y le dedica un campo a cada uno de ellos (por ejemplo, E/S, memoria, o ALU).

Otro aspecto de la codificación es si ésta es directa o indirecta (Figura 15.11b). En la codificación indirecta, se utiliza un campo para determinar la interpretación de otro campo. Por

## Transferencias sencillas entre registros

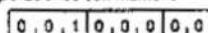
 MDR ← Registro

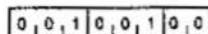
 Registro ← MDR

 MAR ← Registro

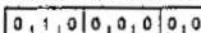
## Selección de registro

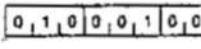
## Operaciones con memoria

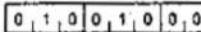
 Lectura

 Escritura

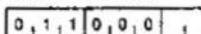
## Operaciones especiales de secuenciamiento

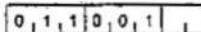
 CSAR ← MDR decodificado

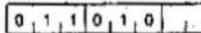
 CSAR ← Constante  
(en el siguiente byte)

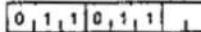
 Salto

## Operaciones con la ALU

 ACC ← ACC + Registro

 ACC ← ACC - Registro

 ACC ← Registro

 Registro ← ACC

 ACC ← Registro + 1

## Selección de registro

## (a) Repertorio vertical de microinstrucciones

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|
|   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |

Campo 1 2 3 4 5 6

## Definición de campos

- 1 - Transferencia entre registros
- 2 - Operación con memoria
- 3 - Operación de secuenciamiento
- 4 - Operación con la ALU
- 5 - Selección de registro
- 6 - Constante

## (b) Formato horizontal de microinstrucción

Figura 15.12. Formatos de microinstrucción alternativos para una máquina sencilla.

ejemplo, consideremos una ALU capaz de realizar ocho operaciones aritméticas y ocho operaciones de desplazamiento diferentes. Se podría usar un campo de 1 bit para indicar si se trata de una operación aritmética o de desplazamiento, y un campo de 3 bits podría indicar la operación. Generalmente, esta técnica implica dos niveles de decodificación, incrementando el retardo de propagación de las señales.

La Figura 15.12 es un ejemplo sencillo de estos conceptos. Se supone un procesador con un único acumulador y varios registros internos, tales como un contador de programa y un registro temporal para la entrada de la ALU. La Figura 15.12a muestra un formato muy vertical. Los 3 primeros bits indican el tipo de operación, los 3 siguientes codifican la operación y los 2 últimos seleccionan un registro interno. La Figura 15.12b es una aproximación más horizontal, aunque todavía usa cierta codificación. En este caso, las funciones diferentes aparecen en campos diferentes.

### EJECUCIÓN DE MICROINSTRUCCIONES EN EL LSI-11

El LSI-11 [SEBE76] es un buen ejemplo del planteamiento de microinstrucciones verticales. Veamos en primer lugar la organización de la unidad de control, y después el formato de la microinstrucción.

#### Organización de la unidad de control del LSI-11

El LSI-11 es el primer miembro de la familia PDP-11 que se ofreció como procesador en una sola tarjeta. La tarjeta contiene tres circuitos LSI, un bus interno conocido como *bus de microinstrucciones* (microinstruction bus, MIB), y alguna lógica de interfaz adicional.

La Figura 15.13 representa, de forma simplificada, la organización del procesador LSI-11. Los tres circuitos son: el de datos, el de control y el de memoria de control. El circuito de

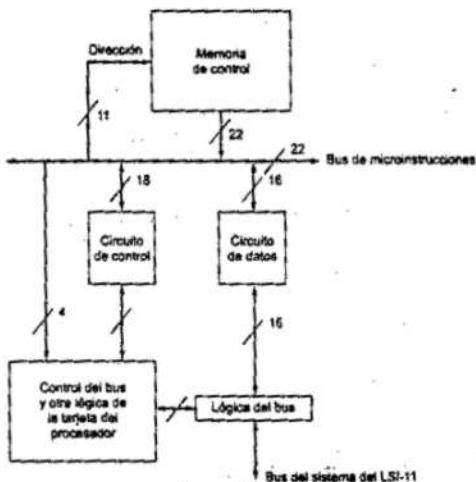


Figura 15.13. Diagrama de bloques simplificado del procesador LSI-11.

datos contiene una ALU de 8 bits, 26 registros de 8 bits, y almacenamiento para varios códigos de condición. Diecisésis de esos registros se usan para implementar los 8 registros de uso general de 16 bits del PDP-11. Otros, incluyen una palabra de estado del programa, un registro de dirección de memoria (MAR), y un registro intermedio de memoria. Como la ALU trata sólo 8 bits a la vez, se requieren dos pasos a través de ella para implementar una operación aritmética de 16 bits del PDP-11. Esto lo controla el microprograma.

El circuito o circuitos de memoria de control contienen la memoria de control de 22 bits de ancho. El circuito de control contiene la lógica de secuenciamiento y ejecución de las microinstrucciones; contiene también el registro de dirección de control, el registro de datos de control, y una copia del registro de instrucción máquina.

El MIB interconecta todos los componentes. Durante la captación de la microinstrucción, el circuito de control sitúa una dirección de 11 bits en el MIB. Se accede a la memoria de control para leer una microinstrucción de 22 bits, que se sitúa en el MIB. Los 16 bits menos significativos van al circuito de datos, y los 18 bits menos significativos van al circuito de control. Los 4 bits más significativos controlan funciones especiales de la tarjeta del procesador.

La Figura 15.14 proporciona una visión todavía simplificada, aunque más detallada, de la unidad de control del LSI-11; la figura no tiene en cuenta los límites de los circuitos. El esquema de secuenciamiento de direcciones descrito en la Sección 15.2 se implementa en dos módulos. El módulo de control de secuencia del microprograma suministra el control de secuencia global, y es capaz de incrementar el registro de dirección de microinstrucciones y realizar saltos incondicionales. Las otras formas de obtención de dirección se llevan a cabo por una tabla de traducción independiente. Ésta, es un circuito combinacional que genera una



Figura 15.14. Organización de la unidad de control del LSI-11.

Tabla 15.5. Algunas microinstrucciones del LSI-11

| Operaciones aritméticas                                                | Operaciones generales                             |
|------------------------------------------------------------------------|---------------------------------------------------|
| Sumar palabra (byte, literal)                                          | Transferir palabra (byte)                         |
| Comprobar palabra (byte, literal)                                      | Saltar                                            |
| Incrementar palabra (byte) en 1                                        | Retornar                                          |
| Incrementar palabra (byte) en 2                                        | Saltar condicionalmente                           |
| Negar palabra (byte)                                                   | Poner a uno (a cero) los indicadores              |
| Incrementar (decrementar) byte<br>condicionalmente                     | Cargar G bajo                                     |
| Sumar palabra (byte) condicionalmente                                  | Transferir palabra (byte) condicionalmente        |
| Sumar palabra (byte) con acarreo                                       | Operaciones de entrada/salida                     |
| Sumar dígitos condicionalmente                                         | Captar palabra (byte)                             |
| Restar palabra (byte)                                                  | Captar palabra (byte) de estado                   |
| Comparar palabra (byte, literal)                                       | Leer                                              |
| Restar palabra (byte) con acarreo                                      | Escribir                                          |
| Decrementar palabra (byte) en 1                                        | Leer (escribir) e incrementar palabra (byte) en 1 |
| Operaciones lógicas                                                    | Leer (escribir) e incrementar palabra (byte) en 2 |
| Y de palabra (byte, literal)                                           | Leer (escribir) reconocimiento                    |
| Comprobar palabra (byte)                                               | Enviar palabra (byte, estado) a la salida         |
| O de palabra (byte)                                                    |                                                   |
| O exclusiva de palabra (byte)                                          |                                                   |
| Poner a cero bit de palabra (byte)                                     |                                                   |
| Desplazar palabra (byte) a la derecha<br>(izquierda) con (sin) acarreo |                                                   |
| Complementar palabra (byte)                                            |                                                   |

dirección a partir de la microinstrucción, la instrucción máquina, el contador de programa de microinstrucciones y un registro de interrupción.

La tabla de traducción entra en juego en las siguientes ocasiones:

- El código de operación se utiliza para determinar el inicio de una microrutina.
- En el momento apropiado, los bits de modo de direccionamiento de la microinstrucción se comprueban para realizar el direccionamiento oportuno.
- Las condiciones de interrupción se comprueban periódicamente.
- Se evalúan las microinstrucciones de bifurcación condicional.

#### Formato de microinstrucción del LSI-11

El LSI-11 utiliza un formato de microinstrucción extremadamente vertical, con un ancho de sólo 22 bits. El repertorio de microinstrucciones se parece mucho al repertorio de instrucciones máquina, el mismo del PDP-11. Este diseño pretendía optimizar las prestaciones de la unidad de control, con la restricción de un diseño vertical y fácilmente programable. La Tabla 15.5 contiene una lista de algunas microinstrucciones del LSI-11.

La Figura 15.15 muestra el formato de 22 bits de la microinstrucción del LSI-11. Los 4 bits más significativos controlan funciones especiales en la tarjeta del procesador. El bit de traducción permite a la tabla de traducción comprobar si hay interrupciones pendientes. El bit de carga del registro de retorno se utiliza al final de una microrutina, para hacer que se use como dirección de la siguiente microinstrucción el contenido del registro de retorno.

Los 16 bits restantes se usan para microoperaciones muy codificadas. El formato se parece mucho al de una instrucción máquina, con un código de operación de longitud variable, y uno o más operandos.



(b) Formato de la parte codificada de la microinstrucción del LSI-11.

Figura 15.15. Formato de microinstrucción del LSI-11.

## EJECUCIÓN DE MICROINSTRUCCIONES EN EL IBM 3033

La memoria de control estándar del IBM 3033 consta de 4K palabras. La primera mitad de ellas (0000-07FF) contiene microinstrucciones de 108 bits, mientras que las restantes (0800-0FFF) se utilizan para almacenar microinstrucciones de 126 bits. El formato se representa en la Figura 15.16. Aún tratándose de un formato bastante horizontal, se sigue usando mucha codificación. Los campos principales del formato se resumen en la Tabla 15.6.

La ALU opera con las entradas que provienen de cuatro registros especializados no visibles para el usuario: A, B, C y D. El formato de microinstrucción contiene campos para cargar estos registros desde otros registros visibles para el usuario, realizar una función de la ALU, y especificar un registro visible para el usuario para el almacenamiento del resultado. Hay también campos para carga y almacenamiento de datos entre registros y memoria.

El mecanismo de secuenciamiento del IBM 3033 se discutió en la Sección 15.2.

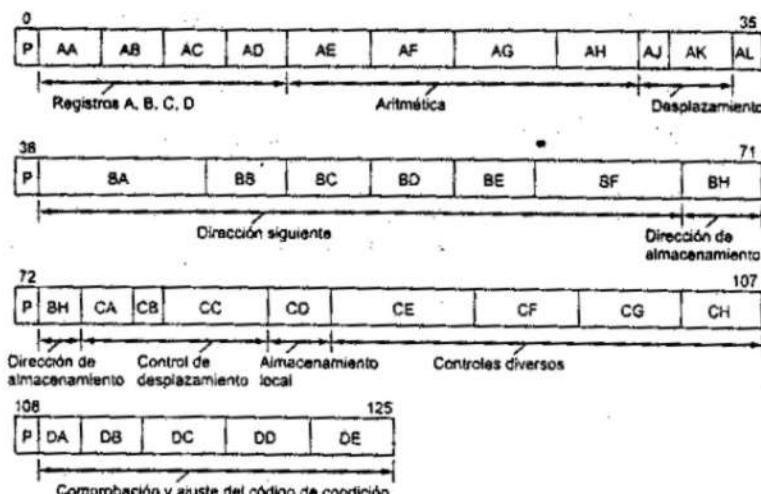


Figura 15.16. Formato de microinstrucción del IBM 3033.

Tabla 15.6. Campos de control de la microinstrucción del IBM 3033

| Campos de control de la ALU       |                                                                                     |
|-----------------------------------|-------------------------------------------------------------------------------------|
| AA(3)                             | Carga registro A desde uno de los registros de datos                                |
| AB(3)                             | Carga registro B desde uno de los registros de datos                                |
| AC(3)                             | Carga registro C desde uno de los registros de datos                                |
| AD(3)                             | Carga registro D desde uno de los registros de datos                                |
| AE(4)                             | Encamina los bits especificados de A hacia la ALU                                   |
| AF(4)                             | Encamina los bits especificados de B hacia la ALU                                   |
| AG(5)                             | Especifica la operación aritmética de la ALU en la entrada A                        |
| AH(4)                             | Especifica la operación aritmética de la ALU en la entrada B                        |
| AJ(1)                             | Especifica la entrada D o B a la ALU en la parte B                                  |
| AK(4)                             | Encamina la salida aritmética hacia el desplazador                                  |
| CB(1)                             | Activa el desplazador                                                               |
| CC(5)                             | Especifica las funciones lógicas y de acarreo                                       |
| CE(7)                             | Especifica el número de desplazamientos                                             |
| CA(3)                             | Carga el registro F                                                                 |
| Campos de secuenciamiento y salto |                                                                                     |
| AL(1)                             | Finaliza la operación y realiza un salto                                            |
| BA(8)                             | Fija los bits de orden superior (00-07) del registro de dirección de control        |
| BB(4)                             | Especifica la condición para ajustar el bit 8 del registro de dirección de control  |
| BC(4)                             | Especifica la condición para ajustar el bit 9 del registro de dirección de control  |
| BD(4)                             | Especifica la condición para ajustar el bit 10 del registro de dirección de control |
| BE(4)                             | Especifica la condición para ajustar el bit 11 del registro de dirección de control |
| BF(4)                             | Especifica la condición para ajustar el bit 12 del registro de dirección de control |

**TI 8800**

La tarjeta de desarrollo de software (Software Development Board, SDB) Texas Instruments 8800, es una tarjeta que contiene un computador de 32 bits microprogramable. El sistema tiene una memoria de control que puede escribirse (Writable Control Store, WCS), implementada en RAM en lugar de en ROM. Este sistema no alcanza la velocidad o densidad de un sistema microprogramado con la memoria de control en ROM; sin embargo, se usa para el desarrollo de prototipos y con fines educativos.

La SDB 8800 consta de los siguientes componentes (Figura 15.17):

- Memoria de microcódigo
- Microsecuenciador
- ALU de 32 bits
- Procesador de enteros y de coma flotante
- Memoria local de datos

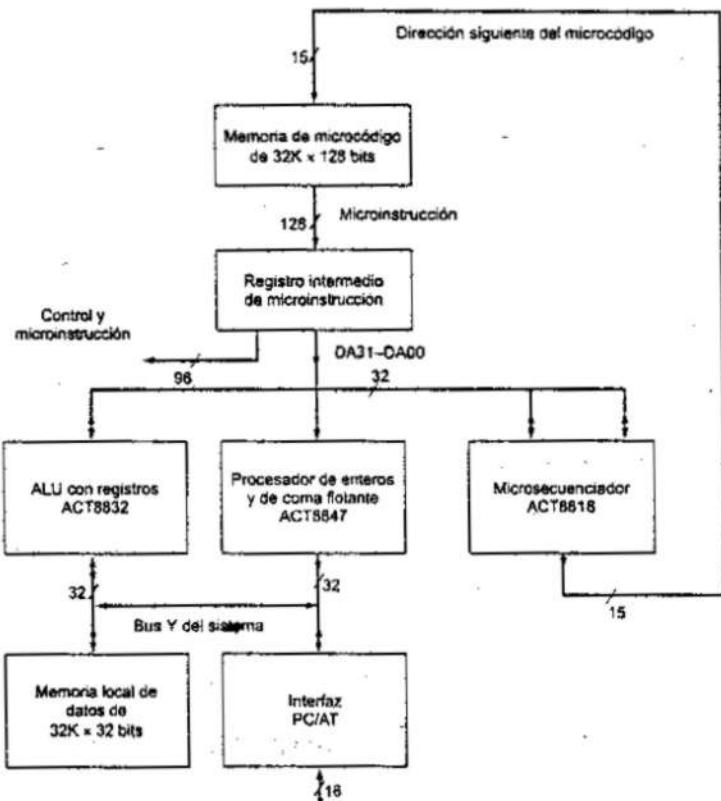


Figura 15.17. Diagrama de bloques del TI 8800.

Los componentes internos del sistema se conectan por medio de dos buses. El bus DA suministra el dato desde el campo de datos de la microinstrucción a la ALU, al procesador de coma flotante, o al microsecuenciador. En el último caso, el dato consta de una dirección, que se utiliza en una instrucción de salto. El bus se puede usar también para el suministro de datos desde la ALU o el microsecuenciador a otros componentes. El bus Y del sistema conecta la ALU y el procesador de coma flotante, con la memoria local y con los módulos externos por medio de la interfaz del PC.

La tarjeta se introduce en un computador anfitrión IBM PC o compatible. El computador anfitrión proporciona una plataforma adecuada para el ensamblaje y la depuración del microcódigo.

### FORMATO DE MICROINSTRUCCIÓN

El formato de microinstrucción del 8800 consta de 128 bits, divididos en 30 campos funcionales, como se indica en la Tabla 15.7. Cada campo consta de uno o más bits, y los campos se agrupan en cinco categorías principales:

- Control de la tarjeta
- Circuito procesador de enteros y de coma flotante 8847
- ALU con registros 8832
- Microsecuenciador 8818
- Campo de datos del WCS

Como se indica en la Figura 15.17, los 32 bits del campo de datos del WCS se introducen en el bus DA, para suministrárslos como dato a la ALU, al procesador de coma flotante, o al microsecuenciador. Los otros 96 bits (campos 1-28) de la microinstrucción, son señales de control que se introducen directamente en el módulo apropiado. Por simplicidad, estas conexiones no se muestran en la Figura 15.17.

Los seis primeros campos se ocupan de operaciones que están relacionadas con el control de la tarjeta, en lugar de controlar un componente particular. Las operaciones de control incluyen lo siguiente:

- Selección de códigos de condición para el control del secuenciador. El primer bit del campo 1 indica si el indicador de condición se va a poner a 1 ó a 0, y los 4 bits restantes indican cuál es ese indicador de condición.
- Envío de una petición de E/S al PC/AT.
- Habilitación de operaciones de lectura/escritura en la memoria local de datos.
- Determinación de la unidad que controla el bus Y del sistema. Se selecciona uno de los cuatro dispositivos que están conectados al bus (Figura 15.17).

Los últimos 32 bits son el campo de datos, que contiene información específica de cada microinstrucción particular.

Los campos restantes de la microinstrucción se estudian mejor en el contexto del dispositivo que controlan. En el resto de esta sección, examinaremos el microsecuenciador y la ALU con registros. Como la unidad de coma flotante no introduce nuevos conceptos, se omite.

### MICROSECUENCIADOR

La función principal del microsecuenciador 8818 es generar la dirección de la siguiente microinstrucción del microprograma. Esta dirección de 15 bits se suministra a la memoria de microcódigo (Figura 15.17).

Tabla 15.7. Formato de microinstrucción del TI 8800

| Número de campo                                                      | Número de bits | Descripción                                                                                                                  |
|----------------------------------------------------------------------|----------------|------------------------------------------------------------------------------------------------------------------------------|
| <b>Control de la tarjeta</b>                                         |                |                                                                                                                              |
| 1                                                                    | 5              | Seleccionar entrada de código de condición                                                                                   |
| 2                                                                    | 1              | Habilitar/inhabilitar señal de petición de E/S externa                                                                       |
| 3                                                                    | 2              | Habilitar/inhabilitar operaciones de lectura/escritura en la memoria de datos local                                          |
| 4                                                                    | 1              | Cargar el estado/no cargar el estado                                                                                         |
| 5                                                                    | 2              | Determinar la unidad que controla el bus Y                                                                                   |
| 6                                                                    | 2              | Determinar la unidad que controla el bus DA                                                                                  |
| <b>Círcuito de procesamiento de enteros y decimala flotante 8847</b> |                |                                                                                                                              |
| 7                                                                    | 1              | Control del registro C: señal de reloj activa o no                                                                           |
| 8                                                                    | 1              | Seleccionar los bits más o menos significativos para el bus Y                                                                |
| 9                                                                    | 1              | Fuente de datos del registro C: ALU, multiplexor                                                                             |
| 10                                                                   | 4              | Seleccionar modo IEEE o FAST para la ALU y el MUL                                                                            |
| 11                                                                   | 8              | Seleccionar fuente para operandos: registros RA, registros RB, registro P, registro S y registro C                           |
| 12                                                                   | 1              | Control del registro RB: señal de reloj activa o no                                                                          |
| 13                                                                   | 1              | Control del registro RA: señal de reloj activa o no                                                                          |
| 14                                                                   | 2              | Confirmación de la fuente de datos                                                                                           |
| 15                                                                   | 2              | Habilitar/inhabilitar registros intermedios                                                                                  |
| 16                                                                   | 11             | Función de la ALU 8847                                                                                                       |
| <b>ALU con registros 8832</b>                                        |                |                                                                                                                              |
| 17                                                                   | 2              | Habilitar/no habilitar la salida de datos hacia el registro seleccionado: mitad más significativa, mitad menos significativa |
| 18                                                                   | 2              | Seleccionar la fuente de datos del banco de registros: bus DA, bus DB, salida ALU y MUX, bus Y del sistema                   |
| 19                                                                   | 3              | Modificador de la instrucción de desplazamiento                                                                              |
| 20                                                                   | 1              | Acarreo de entrada: ponerlo a 1 o a 0                                                                                        |
| 21                                                                   | 2              | Fijar el modo de configuración de la ALU: 32, 16 u 8 bits                                                                    |
| 22                                                                   | 2              | Seleccionar entrada del multiplexor S: banco de registros, bus DB, registro MQ                                               |
| 23                                                                   | 1              | Seleccionar entrada del multiplexor R: banco de registros, bus DA                                                            |
| 24                                                                   | 6              | Seleccionar registro del banco C para escritura                                                                              |
| 25                                                                   | 6              | Seleccionar registro del banco B para lectura                                                                                |
| 26                                                                   | 6              | Seleccionar registro del banco A para escritura                                                                              |
| 27                                                                   | 8              | Función de la ALU                                                                                                            |
| 4 <sup>o</sup> Microsecuenciador 8818                                |                |                                                                                                                              |
| 28                                                                   | 12             | Señales de control que entran al 8818                                                                                        |
| <b>Campo de datos del WCS</b>                                        |                |                                                                                                                              |
| 29                                                                   | 16             | Bits más significativos del campo de datos del WCS                                                                           |
| 30                                                                   | 16             | Bits menos significativos del campo de datos del WCS                                                                         |

La próxima dirección se puede seleccionar desde una de las cinco fuentes:

1. El registro contador de microprograma («microprogram counter register», MPC) se usa para repetir (reutilizar la misma dirección) y continuar (incrementar la dirección en 1) la ejecución de microinstrucciones.
2. La pila, que da soporte a llamadas a subrutinas del microprograma, así como a bucles iterativos y a retornos de interrupciones.
3. Los puertos DRA y DRB, que proporcionan dos caminos adicionales desde el hardware externo por los que se pueden enviar direcciones del microprograma. Estos dos puertos están conectados a los 16 bits más significativos y menos significativos del bus DA, respectivamente. Esto permite al microsecuenciador obtener la dirección de la siguiente microinstrucción a partir del campo de datos del WCS de la microinstrucción en curso, o de un resultado calculado por la ALU.
4. Registros contadores RCA y RCB, que se pueden usar para almacenamiento adicional de direcciones.
5. Una entrada externa en el puerto bidireccional Y para admitir interrupciones externas.

La Figura 15.18 es un diagrama de bloques lógico del 8818. El dispositivo consta de los grupos funcionales principales siguientes:

- Un contador de microprograma (MPC) de 16 bits, compuesto por un registro y un incrementador.
- Dos registros contadores, RCA y RCB, para llevar la cuenta de bucles e iteraciones, almacenar direcciones de salto, o controlar dispositivos externos.
- Una pila de 65 palabras de 16 bits, que permite llamadas a subrutinas de microprograma e interrupciones.
- Un registro de retorno de interrupción que, junto a la salida Y, permite el procesamiento de interrupciones a nivel de microinstrucción.
- Un multiplexor de salida Y, mediante el cual la dirección siguiente se puede seleccionar desde MPC, RCA, RCB, los buses externos DRA y DRB, o desde la pila.

### Registros/contadores

Los registros RCA y RCB se pueden cargar desde el bus DA, bien desde la microinstrucción en curso, o bien desde la salida de la ALU. Los valores se pueden usar como contadores para controlar el flujo de ejecución, y pueden decrementarse automáticamente cuando son accedidos. Los valores también se pueden utilizar como direcciones de microinstrucción y suministrarse al multiplexor de salida Y. Se dispone de un control independiente de ambos registros durante un mismo ciclo de microinstrucción, con la excepción del decremento simultáneo de los dos registros.

### Pila

La pila permite niveles múltiples de llamadas o interrupciones anidadas, y puede utilizarse tanto para saltos como para bucles. No hay que olvidar que estas operaciones se refieren a la unidad de control, no al procesador en su totalidad, y que las direcciones implicadas son las de microinstrucciones en la memoria de control.

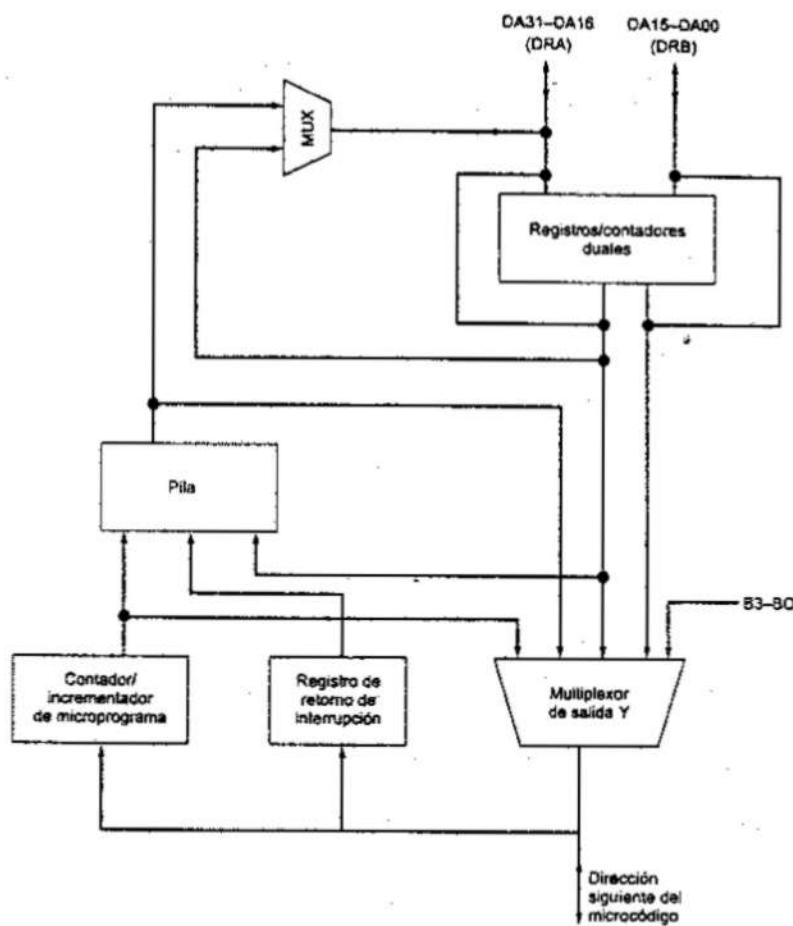


Figura 15.18. El microsecuenciador TI 8818.

Es posible realizar seis operaciones de pila:

1. Poner a cero: pone el puntero de pila a cero, vaciando la pila.
2. Extraer: decrementa el puntero de pila.
3. Apilar: pone el contenido de MPC, del registro de retorno de interrupción, o del bus DRA en la pila, e incrementa el puntero de pila.
4. Leer: pone la dirección indicada por el puntero de lectura a disposición del multiplexor de salida Y.
5. Retener: hace que la dirección del puntero de pila permanezca inalterada.
6. Cargar el puntero de pila: introduce los siete bits menos significativos del DRA en el puntero de pila.

### Control del microsecuenciador

El microsecuenciador está controlado principalmente por el campo de 12 bits de la microinstrucción en curso: campo 28 (Tabla 15.7). Este campo consta de los siguientes subcampos:

- **OSEL (1 bit):** Selección de salida. Determina cuál es el valor que se situará en la salida del multiplexor que introduce información en el bus DRA (esquina superior izquierda de la Figura 15.18). La salida se selecciona para que provenga de la pila o del registro RCA. DRA sirve como entrada del multiplexor de salida Y, o del registro RCA.
- **SELD.R (1 bit):** Selección del bus DR. Si está puesto a 1, este bit selecciona el bus externo DA como entrada a los buses DRA/DRB. Si está puesto a 0, selecciona la salida del multiplexor DRA al bus DRA (controlado por OSEL), y el contenido de RBC al bus DRB.
- **ZEROIN (1 bit):** Se utiliza para indicar un salto condicional. El comportamiento de microsecuenciador dependerá entonces del código de condición seleccionado en el campo 1 (Tabla 15.7).
- **RC2-RC0 (3 bits):** Controles de registros. Estos bits determinan el cambio en el contenido de los registros RCA y RCB. Cada registro puede mantenerse igual, decrementarse o cargarse desde los buses DRA/DRB.
- **S2-S0 (3 bits):** Controles de la pila. Estos bits determinan la operación que se va a realizar en la pila.
- **MUX2-MUX0:** Controles de salida. Estos bits, junto con el código de condición, si se usa, controlan el multiplexor de salida Y y, por tanto, la dirección de la próxima microinstrucción. El multiplexor puede seleccionar su salida desde la pila, DRA, DRB o MPC.

El programador puede ajustar estos bits individualmente. Sin embargo, esto no es lo que se hace normalmente. En su lugar, el programador utiliza mnemotécnicos que equivalen a los patrones de bits que se requerían normalmente. La Tabla 15.8 contiene una lista de los 15 mnemotécnicos para el campo 28. Un ensamblador de microcódigo los convierte en los patrones de bits adecuados.

Por ejemplo, si el código de condición seleccionado actualmente es 1, la instrucción INC88181 se usa para seleccionar la siguiente microinstrucción secuencial. De la Tabla 15.8, tenemos:

**INC88181 = 000000111110**

que se decodifica directamente como:

- **OSEL = 0:** Selecciona RCA como salida del MUX de salida a DRA; en este caso la selección es irrelevante.
- **SELD.R = 0:** Como se definió anteriormente; de nuevo, es irrelevante para esta instrucción.
- **ZEROIN = 0:** Si se combina con el valor para MUX, indica que no se debe realizar el salto.
- **RC = 000:** Conserva el valor actual de RA y RC.
- **S = 111:** Conserva el estado actual de la pila.
- **MUX = 110:** Elige MPC cuando el código de condición es 1, y DRA cuando el código de condición es 0.

Tabla 15.8. Bits de la microinstrucción correspondientes al microsecuenciador TI 8818 (campo 28)

| Mnemotécnico | Valor        | Descripción                                               |
|--------------|--------------|-----------------------------------------------------------|
| RST8818      | 000000000110 | Reinicio de instrucción                                   |
| BRA88181     | 011000111000 | Bifurcar a la instrucción en DRA                          |
| BRA88180     | 010000111110 | Bifurcar a la instrucción en DRA                          |
| INC88181     | 000000111110 | Continuar instrucción                                     |
| INC88180     | 001000001000 | Continuar instrucción                                     |
| CAL88181     | 010000110000 | Salto a la subrutina en la dirección especificada por DRA |
| CAL88180     | 010000101110 | Salto a la subrutina en la dirección especificada por DRA |
| RET8818      | 000000110110 | Retornar de la subrutina                                  |
| PUSH8818     | 000000110111 | Apilar la dirección de retorno de interrupción en la pila |
| POP8818      | 100000010000 | Retornar de interrupción                                  |
| LOADDRA      | 000010111110 | Cargar el contador DRA desde el bus DA                    |
| LOADDRB      | 000110111110 | Cargar el contador DRB desde el bus DA                    |
| LOADDRAB     | 000110111100 | Cargar DRA/DRB                                            |
| DECRDRA      | 010001111100 | Decrementar el contador DRA y saltar si no es cero        |
| OECRDRB      | 010101111100 | Decrementar el contador DRB y saltar si no es cero        |

## ALU CON REGISTROS

El 8832 es una ALU de 32 bits con 64 registros, que se puede configurar para funcionar como cuatro ALU de 8 bits, dos ALU de 16 bits, o una única ALU de 32 bits.

El 8832 se controla con los 39 bits que componen los campos 17 a 27 de las microinstrucciones (Tabla 15.7); éstos se suministran a la ALU como señales de control. Además, como se indica en la Figura 15.17, el 8832 tiene conexiones externas con el bus DA, de 32 bits, y con el bus Y del sistema, también de 32 bits. Las entradas desde DA se pueden suministrar simultáneamente como datos de entrada, al banco de registros de 64 palabras y al módulo de lógica de la ALU. Se proporciona una entrada desde el bus Y del sistema al módulo de lógica de la ALU. Los resultados de la ALU y de las operaciones de desplazamiento, se llevan al bus DA o al bus Y del sistema. Estos resultados también pueden realimentar al banco de registros interno.

En el banco de registros hay tres puertos de direcciones de 6 bits, que permiten realizar simultáneamente una operación de captación de dos operandos, y una escritura de un operando. El desplazador MQ y el registro MQ también pueden configurarse para funcionar independientemente, e implementar operaciones de desplazamiento en doble precisión de 8 bits, 16 bits y 32 bits.

Los campos 17 a 26 de cada microinstrucción controlan el modo en el que los datos fluyen dentro del 8832, y entre el 8832 y el entorno externo. Los campos son los siguientes:

17. *Habilitación de escritura.* Estos dos bits especifican: escritura de los 32 bits, de los 16 bits más significativos, de los 16 menos significativos, o no escritura en el banco de registros. El registro destino se define en el campo 24.
18. *Selección de la fuente de datos del banco de registros.* Si va a ocurrir una escritura en el banco de registros, estos dos bits especifican la fuente: bus DA, bus DB, salida de la ALU, o bus Y del sistema.
19. *Modificador de la instrucción de desplazamiento.* Especifica opciones relacionadas con el suministro de bits de relleno finales, y con la lectura de los bits que se desplazan en las instrucciones de desplazamiento.

20. *Acarreo de entrada.* Este bit indica si se transmite un bit de acarreo a la ALU en la presente operación.
21. *Configuración del modo de la ALU.* El 8832 se puede configurar para operar como una única ALU de 32 bits, dos ALU de 16 bits, o cuatro ALU de 8 bits.
22. *Entrada S.* Dos multiplexores internos, denominados «multiplexores S y R» proporcionan las entradas del módulo de lógica de la ALU. Este campo selecciona la entrada que proporciona el multiplexor S: banco de registros, bus DB, o registro MQ. El registro fuente está definido por el campo 25.
23. *Entrada R.* Selecciona la entrada que suministra el multiplexor R: banco de registros o bus DA.
24. *Registro destino.* Dirección en el banco de registros del registro a usar como operando destino.
25. *Registro fuente.* Dirección en el banco de registros del registro a usar como operando fuente, proporcionada por el multiplexor S.
26. *Registro fuente.* Dirección en el banco de registros del registro a usar como operando fuente, proporcionada por el multiplexor R.

Por último, el campo 27 es un código de operación de 8 bits que especifica la función aritmética o lógica que va a realizar la ALU. La Tabla 15.9 lista las diferentes operaciones que se pueden llevar a cabo.

**Tabla 15.9. Campo de instrucciones de la ALU con registros del TI 8832 (campo 27)**

| Grupo 1 |      | Función          |
|---------|------|------------------|
| ADD     | H#01 | R + S + Cn       |
| SUBR    | H#02 | (NOT R) + S + Cn |
| SUBS    | H#03 | R + (NOT S) + Cn |
| INCS    | H#04 | S + Cn           |
| INCNS   | H#05 | (NOT S) + Cn     |
| INCR    | H#06 | R + Cn           |
| INCNR   | H#07 | (NOT R) + Cn     |
| XOR     | H#09 | R XOR S          |
| AND     | H#0A | R AND S          |
| OR      | H#0B | R OR S           |
| NAND    | H#0C | R NAND S         |
| NOR     | H#0D | R NOR S          |
| ANDNR   | H#0E | (NOT R) AND S    |

Tabla 15.9. Campo de instrucciones de la ALU con registros del TI 8832 (cont.)

| Grupo 2 |       | Función                                                        |
|---------|-------|----------------------------------------------------------------|
| SRA     | H#00  | Desplazamiento aritmético a la derecha en precisión sencilla   |
| SRAD    | H#10  | Desplazamiento aritmético a la derecha en precisión doble      |
| SRL     | H#20  | Desplazamiento lógico a la derecha en precisión sencilla       |
| SRD     | H#30  | Desplazamiento lógico a la derecha en precisión doble          |
| SLA     | H#40  | Desplazamiento aritmético a la izquierda en precisión sencilla |
| SLAD    | H#50  | Desplazamiento aritmético a la izquierda en precisión doble    |
| SLC     | H#60  | Desplazamiento circular a la izquierda en precisión sencilla   |
| SLCD    | H#70  | Desplazamiento circular a la izquierda en precisión doble      |
| SRC     | H#80  | Desplazamiento circular a la derecha en precisión sencilla     |
| SRCD    | H#90  | Desplazamiento circular a la derecha en precisión doble        |
| MQSRA   | HH#A0 | Desplazamiento aritmético a la derecha del registro MQ         |
| MQSRL   | H#B0  | Desplazamiento lógico a la derecha del registro MQ             |
| MQSLL   | H#C0  | Desplazamiento lógico a la izquierda del registro MQ           |
| MQSLC   | H#D0  | Desplazamiento circular a la izquierda del registro MQ         |
| LOADMQ  | H#E0  | Carga del registro MQ                                          |
| PASS    | H#F0  | Pasar ALU a Y (sin desplazar)                                  |
| Grupo 3 |       | Función                                                        |
| SET1    | H#08  | Fijar bit 1                                                    |
| SET0    | H#18  | Fijar bit 0                                                    |
| TB1     | H#28  | Comprobar bit 1                                                |
| TB0     | H#38  | Comprobar bit 0                                                |
| ABS     | H#48  | Valor absoluto                                                 |
| SMTC    | H#58  | Signo y magnitud/complemento a dos                             |
| ADDI    | H#68  | Suma inmediata                                                 |
| SUBI    | H#78  | Resta inmediata                                                |
| BADD    | H#88  | Suma (byte) R a S                                              |
| BSUBS   | H#98  | Resta (byte) S de R                                            |

Tabla 15.9. Campo de instrucciones de la ALU con registros del TI 8832 (cont.)

| Grupo 3 |      | Función                                           |
|---------|------|---------------------------------------------------|
| BSUBR   | H#A8 | Resta (byte) R de S                               |
| BINCS   | H#B8 | Incremento (byte) S                               |
| BINCNS  | H#C8 | Incremento negativo (byte) S                      |
| BXOR    | H#D8 | XOR (byte) R y S                                  |
| BAND    | H#E8 | AND (byte) R y S                                  |
| BOR     | H#F8 | OR (byte) R y S                                   |
| Grupo 4 |      | Función                                           |
| CRC     | H#00 | Acumular caracteres de redundancia cíclica        |
| SEL     | H#10 | Seleccionar S ó R                                 |
| SNORM   | H#20 | Normalización de longitud sencilla                |
| DNORM   | H#30 | Normalización de longitud doble                   |
| DIVRF   | H#40 | Determinar resto de división                      |
| SDIVQF  | H#50 | Determinar cociente de división                   |
| SMULI   | H#60 | Iterar multiplicación con signo                   |
| SMULT   | H#70 | Terminar multiplicación con signo                 |
| SDIVIN  | H#80 | Inicializar división con signo                    |
| SDIVIS  | H#90 | Comenzar división con signo                       |
| SDIVI   | H#A0 | Iterar división con signo                         |
| UDIVIS  | H#B0 | Comenzar división sin signo                       |
| UDIVI   | H#C0 | Iterar división sin signo                         |
| UMULI   | H#D0 | Iterar multiplicación sin signo                   |
| SDIVIT  | H#E0 | Terminar división con signo                       |
| UDIVIT  | H#F0 | Terminar división sin signo                       |
| Grupo 5 |      | Función                                           |
| LOADFF  | H#0F | Cargar los biestables de división/BCD             |
| CLR     | H#1F | Borrar                                            |
| DUMPFF  | H#5F | Enviar a la salida los biestables de división/BCD |

Tabla 15.9. Campo de instrucciones de la ALU con registros del TI 8832 (cont.)

| Grupo 5 | Función |                                                         |
|---------|---------|---------------------------------------------------------|
| BCDOBIN | H#7F    | BCD a binario                                           |
| EX3BC   | H#8F    | Corrección de byte en exceso_3                          |
| EX3C    | H#9F    | Corrección de palabra en exceso_3                       |
| SDIVO   | H#AF    | Comprobación de desbordamiento en la división con signo |
| BINEX3  | H#DF    | Binario a exceso_3                                      |
| NOP32   | H#FF    | No operar                                               |

Como ejemplo de la codificación que se usa para especificar los campos 17 a 27, se considera la instrucción que suma los contenidos del registro 1 y el registro 2, y deposita el resultado en el registro 3. La instrucción simbólica es:

CONT11 [17], WELH, SELRFYMX, [24], R3, R2, R1, PASS+ADD

El ensamblador la traducirá al patrón de bits conveniente. Los componentes individuales de la instrucción se pueden describir como sigue:

- CONT11 es la instrucción básica NOP.
- El campo [17] se transforma en WELH (habilitación de escritura, baja y alta; «write enable, low and high»), de manera que se escribe en un registro de 32 bits.
- El campo [18] se transforma en SELRFYMX para seleccionar la realimentación desde la salida ALU Y MUX.
- El campo [24] se transforma para designar el registro R3 como registro destino.
- El campo [25] se modifica para designar el registro R2 como uno de los registros fuente.
- El campo [26] se modifica para designar el registro R1 como uno de los registros fuente.
- El campo [27] se modifica para especificar una operación ALU de suma (ADD). La instrucción de desplazamiento de la ALU es PASS; por tanto, la salida de la ALU no está desplazada.

Se pueden destacar varios puntos sobre la notación simbólica. No es necesario especificar el número de campo para campos consecutivos. Es decir,

CONT11 [17], WELH, [18], SELRFYMX

se puede escribir como:

CONT11 [17], WELH, SELRFYMX

ya que SELRFYMX está en el campo 18.

Las instrucciones de la ALU del Grupo 1 de la Tabla 15.9 se deben utilizar siempre conjuntamente con las del Grupo 2. Las de los Grupos 3-5 no deben usarse con las del Grupo 2.

## 15.5. APLICACIONES DE LA MICROPROGRAMACIÓN

Desde la introducción de la microprogramación, y especialmente desde finales de los sesenta las aplicaciones de la microprogramación se han hecho cada vez más variadas y extendidas. Ya en 1971, la mayoría de las utilidades contemporáneas de la microprogramación, si no todas, eran bien visibles [FLYN71]. Los estudios más recientes tratan esencialmente el mismo conjunto de aplicaciones (por ejemplo [RAUS80]). El conjunto de aplicaciones actuales de la microprogramación incluye lo siguiente:

- Realización de computadores
- Emulación
- Soporte de sistemas operativos
- Realización de dispositivos de uso especial
- Soporte de lenguajes de alto nivel
- Microdiagnósticos
- Adaptación al usuario

Este capítulo se ha dedicado al estudio de la *realización de computadores*. La aproximación microprogramada ofrece una técnica sistemática para la implementación de una unidad de control. Una técnica relacionada, es la *emulación* [MALL75]. Ésta se refiere al uso de un microprograma en una máquina, para ejecutar programas escritos originalmente para otra. La utilidad más común de la emulación es ayudar a los usuarios en la migración de un computador a otro. Esto lo hace con frecuencia un fabricante para facilitar a sus clientes actuales el cambio de máquinas viejas por otras más nuevas, haciendo de este modo poco atractivo el cambio a otro fabricante. Los usuarios a menudo se sorprenden, al darse cuenta del tiempo que permanece esta herramienta de transición. Un observador, [MALL83], se dio cuenta de que en 1983 era posible todavía encontrar un IBM System/370 emulando un IBM 1401, que había sido reemplazado físicamente más de una década y media antes.

Otro uso provechoso de la microprogramación está en el área de *soporte de sistemas operativos*. Los microprogramas se pueden utilizar para implementar primitivas que reemplazan partes importantes del software del sistema operativo. Esta técnica puede simplificar las labores de implementación de sistemas operativos y mejorar sus prestaciones.

La microprogramación se usa como medio de implementación de *dispositivos de uso especial*, que se pueden incorporar en un computador anfitrión. Un buen ejemplo de esto es una tarjeta de comunicaciones de datos. La tarjeta dispondrá de su propio microprocesador. Como se está utilizando para uso especial, tiene sentido implementar algunas de sus funciones en firmware en vez de en software, para aumentar las prestaciones.

El *soporte de lenguajes de alto nivel* es otra área donde resulta provechosa la aplicación de técnicas de microprogramación. Se pueden implementar varias funciones y tipos de datos directamente en firmware. El resultado es que es más fácil compilar el programa en forma de lenguaje máquina eficiente. En efecto, el lenguaje máquina está adaptado para satisfacer las necesidades del lenguaje de alto nivel (por ejemplo, FORTRAN, COBOL, o Ada).

La microprogramación se puede utilizar para ofrecer soporte a la supervisión, detección, aislamiento y reparación de errores del sistema. Estas características se conocen como *microdiagnósticos*, y pueden mejorar significativamente el mantenimiento del sistema. Esta aproximación permite al sistema reconfigurarse cuando se detecta un fallo; por ejemplo, si un multiplicador de alta velocidad está funcionando mal, un multiplicador microprogramado puede tomar el relevo.

Una categoría general de aplicación es la *adaptación a los usuarios*. Varias máquinas proporcionan un *memoria de control con posibilidad de escritura*, es decir, una memoria de control implementada en RAM en vez de en ROM, y permiten al usuario escribir microprogramas. Generalmente, se proporciona un repertorio de microinstrucciones fácil de usar y muy vertical. Esto permite al usuario adaptar la máquina a la aplicación deseada.

### 15.6. LECTURAS RECOMENDADAS

Hay varios libros dedicados a la microprogramación. Quizás el más completo es [LYNC93]. [SEGE91] presenta los fundamentos de la microcodificación y el diseño de sistemas microcodificados, mediante un diseño paso a paso de un procesador sencillo de 16 bits. [CART96] también presenta los conceptos básicos usando una máquina de ejemplo. [PARK89] y [TI90] proporcionan una descripción detallada de la tarjeta de desarrollo de software TI 8800.

- CART96 Carter, J. *Microprocessor Architecture and Microprogramming*. Upper Saddle River, NJ: Prentice Hall, 1996.
- LYNC93 Lynch, M. *Microprogrammed State Machine Design*. Boca Raton, FL: CRC Press, 1993.
- PARK89 Parker, A., y Hamblen, J. *An Introduction to Microprogramming with Exercises Designed for the Texas Instruments SN74ACT8800 Software Development Board*. Dallas, TX: Texas Instruments, 1989.
- SEGE91 Segee, B., y Field, J. *Microprogramming and Computer Architecture*. New York: Wiley, 1991.
- TI90 Texas Instruments Inc. *SN74ACT8800 Family Data Manual*. SCSS006C, 1990.

### 15.7. PROBLEMAS

- 15.1. Describa la implementación de la instrucción de multiplicación, en la máquina hipotética diseñada por Wilkes. Utilice una descripción narrada y un organigrama.
- 15.2. Suponga un repertorio de microinstrucciones, que incluye una microinstrucción con la siguiente forma simbólica:

IF ( $AC_0 = 1$ ) THEN  $CAR \leftarrow (C_{5:4})$  ELSE  $CAR \leftarrow (CAR) + 1$

donde  $AC_0$  es el bit de signo del acumulador, y  $C_{5:4}$  son los siete primeros bits de la microinstrucción. Utilizando esta microinstrucción, escriba un microprograma que implemente un instrucción máquina (Branch Register Minus BRM), que salte si el AC es negativo. Suponga que los bits  $C_1 \dots C_4$  de la microinstrucción especifican un conjunto paralelo de microoperaciones. Exprese el programa simbólicamente.

- 15.3. Un procesador sencillo tiene cuatro fases principales en su ciclo de instrucción: captación, ciclo indirecto, ejecución e interrupción. Dos indicadores de 1 bit señalan la fase en curso en una implementación cableada.
  - a) ¿Por qué se necesitan estos indicadores?
  - b) ¿Por qué no se necesitan en una unidad de control microprogramada?

- 15.4. Considere la unidad de control de la Figura 15.7. Suponga que la memoria de control tiene una anchura de 24 bits. La parte de control del formato de microinstrucción está dividida en dos campos. Un campo de microoperación de 13 bits especifica las microoperaciones que se van a realizar. Un campo de selección de dirección especifica una condición, basada en los indicadores, que originará un salto de microinstrucción. Hay ocho indicadores.
- ¿Cuántos bits hay en el campo de selección de dirección?
  - ¿Cuántos bits hay en el campo de dirección?
  - ¿Cuál es el tamaño de la memoria de control?
- 15.5. ¿Cómo se puede realizar un salto incondicional bajo las circunstancias del problema anterior? ¿Cómo se puede evitar el salto? (Es decir, describa una microinstrucción que no especifique ningún salto, ni condicional ni incondicional.)
- 15.6. Deseamos proporcionar 8 palabras de control para cada rutina de instrucción máquina. Los códigos de operación de la instrucción máquina tienen 5 bits, y la memoria de control tiene 1.024 palabras. Sugiera una traducción del registro de instrucción al registro de dirección de control.
- 15.7. Se va a usar un formato de microinstrucción codificado. Muestre cómo un campo de microoperación de 9 bits se puede dividir en subcampos para especificar 46 acciones diferentes.
- 15.8. Un procesador tiene 16 registros, una ALU con 16 funciones lógicas y 16 funciones aritméticas, y un desplazador con 8 operaciones, todo conectado por un bus interno del procesador. Diseñe el formato de microinstrucción para especificar las distintas microoperaciones del procesador.

## PARTE V

# ORGANIZACIÓN PARALELA

### CUESTIONES A TRATAR EN LA QUINTA PARTE

La parte final del libro considera los aspectos de la cada vez más importante organización paralela. En una organización paralela, varias unidades de proceso cooperan en la ejecución de aplicaciones. Mientras que un procesador superescalar aprovecha las posibilidades de ejecución paralela de instrucciones, una organización de procesamiento paralelo utiliza un nivel de paralelismo más grueso que permite que varios procesadores puedan ejecutar el trabajo a realizar en paralelo, y cooperativamente. Existen una serie de cuestiones que deben resolverse en este tipo de organizaciones. Por ejemplo, si muchos procesadores, cada uno con su cache, comparten el acceso a la misma memoria, deben emplearse mecanismos hardware o software para asegurar que todos los procesadores comparten una imagen válida de la memoria principal. Esto se conoce como problema de coherencia de cache. Este aspecto, junto con otros, se analizará en la Parte V.

### ESQUEMA DE LA QUINTA PARTE

#### CAPÍTULO 16. PROCESAMIENTO PARALELO

El Capítulo 16 proporciona un revisión de las cuestiones relativas al procesamiento paralelo. Después, el capítulo considera tres organizaciones de varios procesadores: los multiprocesadores simétricos (SMP, «symmetric multiprocessors»), los clusters, y las máquinas de acceso a memoria no uniforme (NUMA, «nonuniform memory access»). Los SMP y los clusters son las formas más comunes de conseguir mejorar las prestaciones y la disponibilidad, mediante el uso de varios procesadores. Los sistemas NUMA constituyen un concepto más reciente, que todavía no ha alcanzado un éxito comercial amplio, aunque representan una alternativa muy prometedora. Por último, el Capítulo 16 considera una organización de propósito específico, conocida como «procesador vectorial».



## CAPÍTULO 16

# Procesamiento paralelo

### 16.1. Organizaciones con varios procesadores

Tipos de sistemas de paralelos  
Organizaciones paralelas

### 16.2. Multiprocesadores simétricos

Organización  
Consideraciones en el diseño de un sistema operativo de un multiprocesador  
Un SMP como gran computador

### 16.3. Coherencia de cache y el protocolo MESI

Soluciones software  
Soluciones hardware  
El protocolo MESI

### 16.4. «Clusters»

Configuraciones de «clusters»  
Consideraciones en el diseño del sistema operativo  
«Clusters» frente a SMP

### 16.5. Acceso no uniforme a memoria

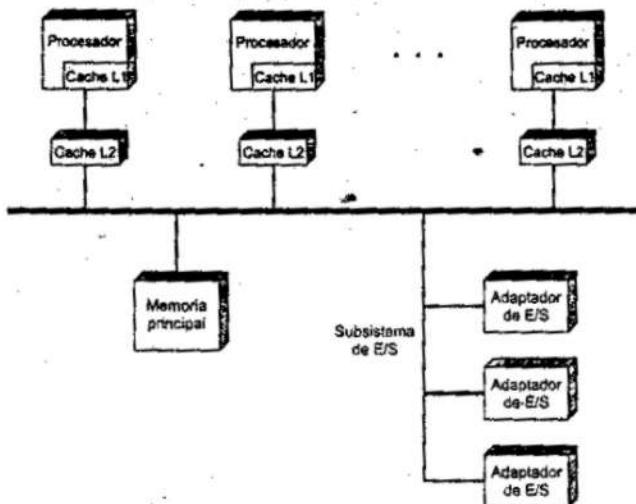
Motivación  
Organización  
Pros y contras de un computador NUMA

### 16.6. Computación vectorial

Aproximaciones a la computación vectorial  
Unidad vectorial del IBM 3090

### 16.7. Lecturas recomendadas

### 16.8. Problemas



- Una manera tradicional de incrementar las prestaciones de un sistema consiste en utilizar varios procesadores que puedan ejecutar en paralelo una carga de trabajo dada. Las dos organizaciones de múltiples procesadores más comunes son los multiprocesadores simétricos (SMP) y los «clusters». Recientemente, los sistemas de acceso a memoria no uniforme (NUMA) han aparecido comercialmente.
- Un SMP es un computador constituido por varios procesadores similares, interconectados mediante un bus o algún tipo de estructura de conmutación. El problema más crítico a resolver en un SMP es la coherencia de cache. Cada procesador tiene su propia cache, y es posible que una línea de datos dada esté presente en más de una cache. Si esa línea se altera en una cache, entonces tanto la memoria principal como las otras caches tienen versiones no válidas de dicha línea.
- Un «cluster» es un grupo de computadores completos interconectados y trabajando juntos como un sólo recurso de cómputo, proporcionando la ilusión de ser una única máquina. El término *computador completo* significa que puede funcionar autónomamente fuera del cluster.
- Un sistema NUMA es un multiprocesador de memoria compartida, en el que el tiempo de acceso de un procesador a una palabra de memoria varía según la ubicación de la palabra en memoria.

Tradicionalmente, el computador se ha visto como una máquina secuencial. La mayoría de los lenguajes de programación del computador requieren que el programador especifique los algoritmos mediante una secuencia de instrucciones. Los procesadores ejecutan los programas procesando instrucciones máquina de una en una. Cada instrucción se ejecuta mediante una secuencia de operaciones (captar instrucción, captar operandos, realizar la operación y almacenar los resultados).

Esta visión del computador no es completamente cierta. En el nivel de microoperación, se generan al mismo tiempo múltiples señales de control. La segmentación de las instrucciones, al menos en cuanto al solapamiento de las operaciones de captación y ejecución, se ha utilizado desde hace tiempo. Ambos casos son ejemplos de funciones que se realizan en paralelo. Es el mismo enfoque de la organización superescalar, que aprovecha el paralelismo entre instrucciones. Un procesador superescalar dispone de varias unidades de ejecución, que pueden ejecutar en paralelo varias instrucciones del mismo programa.

A medida que la tecnología de los computadores se ha desarrollado, y ha disminuido el costo del hardware del computador, los diseñadores de computadores han visto más y más posibilidades en el paralelismo, normalmente para mejorar las prestaciones y, en algunos casos, para mejorar la fiabilidad. Después de una revisión de conceptos, en este capítulo se examinan las tres organizaciones paralelas de más éxito. En primer lugar se estudian los multiprocesadores simétricos (SMP), una de las primeras y, todavía, el ejemplo más común de organización paralela. Un SMP incluye varios procesadores que comparten la memoria principal común. La organización SMP pone de manifiesto el problema de la coherencia de cache, al que se dedica una sección específica. Después se describen los «clusters», que están constituidos por varios computadores independientes, organizados para poder trabajar cooperativamente. Los «clusters» han llegado a ser bastante comunes a la hora de procesar cargas de trabajo que sobrepasan la capacidad de un SMP. La tercera aproximación al uso de varios procesadores está representada por las máquinas de acceso no uniforme a memoria (NUMA). La alternativa NUMA es relativamente nueva, y todavía no se ha extendido comercialmente, pero, a menudo, se considera la alternativa a los computadores SMP y a los «clusters». Finalmente, el capítulo estudia las aproximaciones hardware a la computación vectorial. Estas propuestas optimizan la ALU para el procesamiento de vectores o matrices de números en coma flotante. Se han utilizado en la implementación de los sistemas conocidos como *supercomputadores*.

## 16.1. ORGANIZACIONES CON VARIOS PROCESADORES

### TIPOS DE SISTEMAS DE PARALELOS

La taxonomía introducida primeramente por Flynn [FLYN72] es todavía la forma más común de clasificar a los sistemas según sus capacidades de procesamiento paralelo. Flynn propuso las siguientes categorías o clases de computadores:

- Una secuencia de instrucciones y una secuencia de datos (SISD, Single Instruction, Single Data): Un único procesador interpreta una única secuencia de instrucciones, para operar con los datos almacenados en una única memoria. Los computadores monoprocesador caen dentro de esta categoría.
- Una secuencia de instrucciones y múltiples secuencias de datos (SIMD, Single Instruction Multiple Data): Una única instrucción máquina controla paso a paso la ejecución simultánea y sincronizada de un cierto número de elementos de proceso. Cada elemento de proceso tiene una memoria asociada, de forma que cada instrucción es ejecutada por

cada procesador, con un conjunto de datos diferentes. Los procesadores vectoriales y los matriciales pertenecen a esta categoría.

- **Múltiples secuencias de instrucciones y una secuencia de datos (MISD):** Se transmite una secuencia de datos a un conjunto de procesadores, cada uno de los cuales ejecuta una secuencia de instrucciones diferente. Esta estructura nunca ha sido implementada.
- **Múltiples secuencias de instrucciones y múltiples secuencias de datos (MIMD):** Un conjunto de procesadores ejecuta simultáneamente secuencias de instrucciones diferentes con conjuntos de datos diferentes. Los SMP, los «clusters», y los sistemas NUMA son ejemplos de esta categoría.

En la organización MIMD, los procesadores son de uso general; cada uno es capaz de procesar todas las instrucciones necesarias para realizar las transformaciones apropiadas de los datos. Los computadores MIMD se pueden subdividir, además, según la forma que tienen los procesadores para comunicarse (Figura 16.1). Si los procesadores comparten una memoria común, entonces cada procesador accede a los programas y datos almacenados en la memoria compartida, y los procesadores se comunican unos con otros a través de esa memoria. La forma más común de este tipo de sistemas se conoce como multiprocesador simétrico (SMP), que se examinará en la Sección 16.2. En un SMP, varios procesadores comparten una única memoria mediante un bus compartido u otro tipo de mecanismo de interconexión. Una característica distintiva de estos sistemas es que el tiempo de acceso a memoria principal es aproximadamente el mismo para cualquier procesador. Un desarrollo más reciente es la organización con acceso no uniforme a memoria (NUMA), que se describe en la Sección 16.5. Como el propio nombre indica, el tiempo de acceso a zonas de memoria diferentes puede diferir en un computador NUMA.



Figura 16.1. Taxonomía de las arquitecturas paralelas.

Un conjunto de computadores monoprocesador independientes, o de SMP, pueden interconectarse para formar un «cluster». La comunicación entre los computadores se realiza mediante conexiones fijas o mediante algún tipo de red.

## ORGANIZACIONES PARALELAS

La Figura 16.2 muestra los esquemas generales de las clases de la taxonomía de la Figura 16.1. La Figura 16.2a corresponde a la estructura de un SISD. Se dispone de una unidad de control (UC) que proporciona una secuencia de instrucciones (SI) a una unidad de proceso (UP). La unidad de proceso actúa sobre una única secuencia de datos (SD) captados desde la unidad de memoria (UM). En una máquina SIMD, también existe una sola unidad de control, que proporciona una única secuencia de instrucciones a cada elemento de proceso. Cada elemento de proceso puede tener su propia memoria dedicada (mostrada en la Figura 16.2b), o puede haber una memoria compartida. Finalmente, en un computador MIMD hay múltiples unidades de control, y cada una proporciona una secuencia de instrucciones separada a su propio elemento de proceso. El MIMD puede ser un multiprocesador de memoria compartida (Figura 16.2c), o un multicamputador de memoria distribuida (Figura 16.2d).

Los aspectos de diseño relacionados con los SMP, los «clusters», y los NUMA, son complejos, implicando cuestiones relativas a la organización física, las estructuras de interconexión, el diseño de los sistemas operativos, y el software de las aplicaciones. Nuestro interés se centra fundamentalmente en la organización, aunque se describirán brevemente aspectos del diseño de los sistemas operativos.

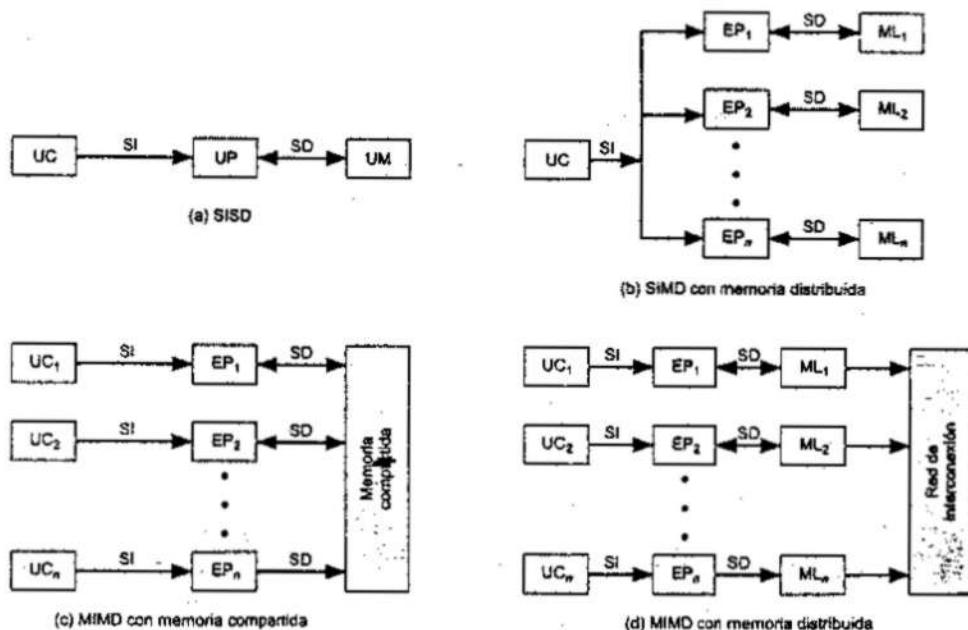


Figura 16.2. Organizaciones de computador alternativas.

## 16.2. MULTIPROCESADORES SIMÉTRICOS

Hasta hace poco, prácticamente todos los computadores personales y estaciones de trabajo utilizaban un único microprocesador de uso general. A medida que aumenta la demanda de mayores prestaciones, y dado que el coste de los microprocesadores continúa reduciéndose, los fabricantes han introducido los sistemas SMP. El término SMP se refiere a la arquitectura hardware del computador, y también al comportamiento del sistema operativo que utiliza dicha arquitectura. Un SMP puede definirse como un computador con las siguientes características:

1. Hay dos o más procesadores similares de capacidades comparables.
2. Estos procesadores comparten la memoria principal y las E/S, y están interconectados mediante un bus u otro tipo de sistema de interconexión, de forma que el tiempo de acceso a memoria es aproximadamente el mismo para todos los procesadores.
3. Todos los procesadores comparten los dispositivos de E/S, bien a través de los mismos canales, o bien mediante canales distintos que proporcionan caminos de acceso al mismo dispositivo.
4. Todos los procesadores pueden desempeñar las mismas funciones (de ahí el término *simétrico*).
5. El sistema está controlado por un sistema operativo integrado, que proporciona la interacción entre los procesadores y sus programas en los niveles de trabajo, tarea, fichero, y datos.

El significado de los puntos 1 a 4 es claro. El punto 5 corresponde a una de las diferencias con los sistemas multiprocesador débilmente acoplados, tales como los «clusters». En éstos, la unidad de interacción física es normalmente un mensaje o un fichero completo. En un SMP, la interacción se puede producir a través de elementos de datos individuales, y puede existir un elevado nivel de cooperación entre procesadores.

El sistema operativo de un SMP planifica la distribución de procesos o hilos (threads) entre todos los procesadores. Un SMP tiene las siguientes ventajas potenciales con respecto a una arquitectura monoprocesador:

- **Prestaciones:** Si el trabajo a realizar por un computador puede organizarse de forma que partes del mismo se puedan realizar en paralelo, entonces un sistema con varios procesadores proporcionará mejores prestaciones que uno con un sólo procesador del mismo tipo (Figura 16.3).
- **Disponibilidad:** En un multiprocesador simétrico, debido a que todos los procesadores pueden realizar las mismas funciones, un fallo en un procesador no hará que el computador se detenga.
- **Crecimiento incremental:** Se pueden aumentar las prestaciones del sistema, añadiendo más procesadores.
- **Escalado:** Los fabricantes pueden ofrecer una gama de productos con precios y prestaciones diferentes, en función del número de procesadores que configuran el sistema.

Es importante resaltar que los anteriores son beneficios potenciales, más que beneficios garantizados. El sistema operativo debe disponer de herramientas y funciones que permitan explotar el paralelismo presente en un SMP.

Una característica atractiva de un SMP es que la existencia de varios procesadores es transparente al usuario. El sistema operativo se encarga de la sincronización entre los procesadores, y de la planificación de los hilos o de los procesos, asignándolos a los distintos procesadores.

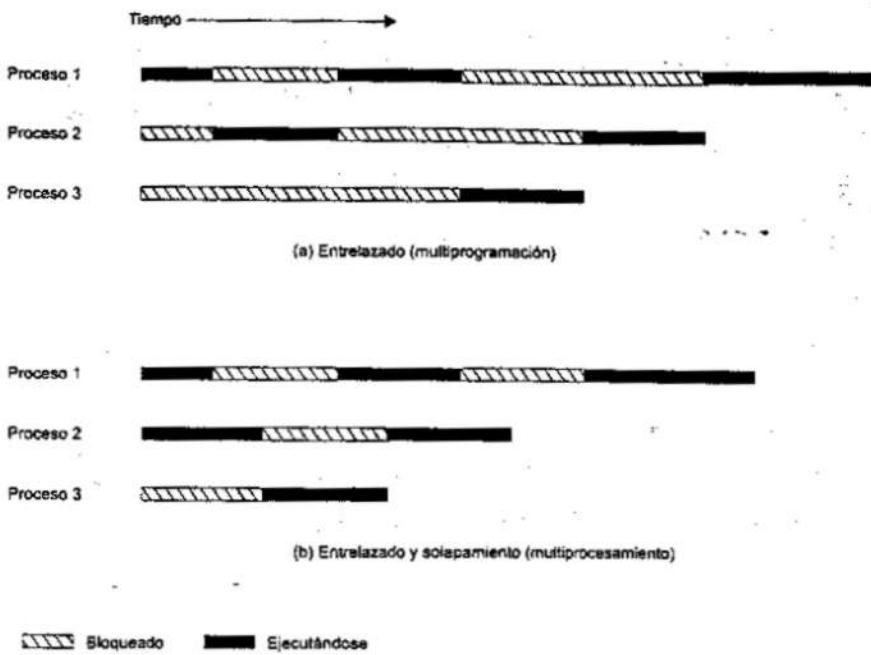


Figura 16.3. Multiprogramación y multiprocесamiento.

## ORGANIZACIÓN

La Figura 16.4 describe en términos generales la organización de un sistema multiprocесador. Hay dos o más procesadores. Cada procesador es autónomo, incluyendo una unidad de control, una ALU, registros y, posiblemente, cache. Cada procesador tiene acceso a una memoria principal compartida y a los dispositivos de E/S, a través de alguna forma de mecanismo de interconexión. Los procesadores pueden comunicarse entre si a través de la memoria (mensajes e información de control almacenada en áreas comunes para datos). También es posible que los procesadores intercambien señales directamente. La memoria, a menudo, se organiza de forma que sean posibles los accesos simultáneos a bloques de memoria separados. En algunas configuraciones, cada procesador puede tener también su propia memoria principal privada y sus canales de E/S, además de los recursos compartidos.

La organización de un sistema de multiprocесador puede clasificarse como sigue:

- Bus de tiempo compartido o común
- Memoria multipuerto
- Unidad de control central

### Bus de tiempo compartido

El bus de tiempo compartido es el mecanismo más simple para construir un sistema multiprocесador (Figura 16.5). La estructura y las interfaces son básicamente las mismas que las de

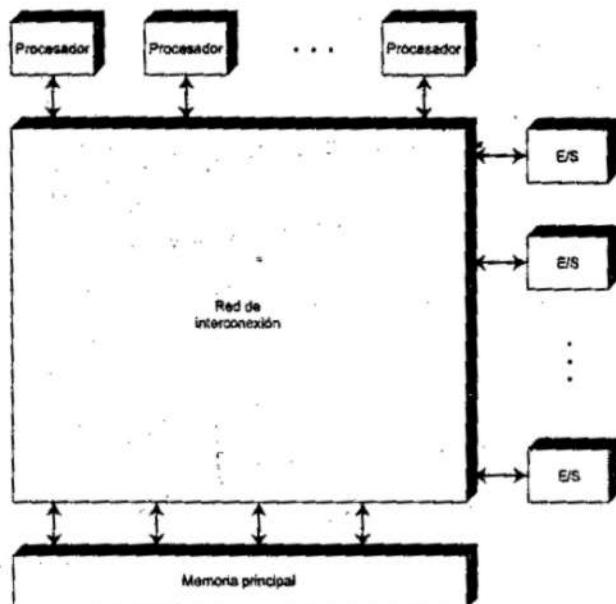


Figura 16.4. Diagrama de bloques genérico de un multiprocesador fuertemente acoplado.

un sistema de un único procesador que utilice un bus para la interconexión. El bus consta de líneas de control, dirección y datos. Para facilitar las transferencias de DMA con los procesadores de E/S, se proporcionan los elementos para el:

- **Direccionalamiento:** Debe ser posible distinguir los módulos del bus para determinar la fuente y el destino de los datos.
- **Arbitraje:** Cualquier módulo de E/S puede funcionar temporalmente como maestro. Se proporciona un mecanismo para arbitrar entre las peticiones que compiten por el control del bus, utilizando algún tipo de esquema de prioridad.
- **Tiempo compartido:** Cuando un módulo está controlando el bus, los otros módulos no tienen acceso al mismo, y deben, si es necesario, suspender su operación hasta que dispongan del bus.

Estas características monoprocesador son utilizables directamente en una configuración de SMP. En este caso, hay varias CPU, además de varios procesadores de E/S, que intentan tener acceso a uno o más módulos de memoria a través del bus.

La organización del bus presenta diversas ventajas en comparación con otras propuestas:

- **Simplicidad:** Es la aproximación más simple para organizar el multiprocesador. La interfaz física, y la lógica de cada procesador para el direccionamiento, para el arbitraje, y para compartir el tiempo del bus, es el mismo que el de un sistema con un solo procesador.

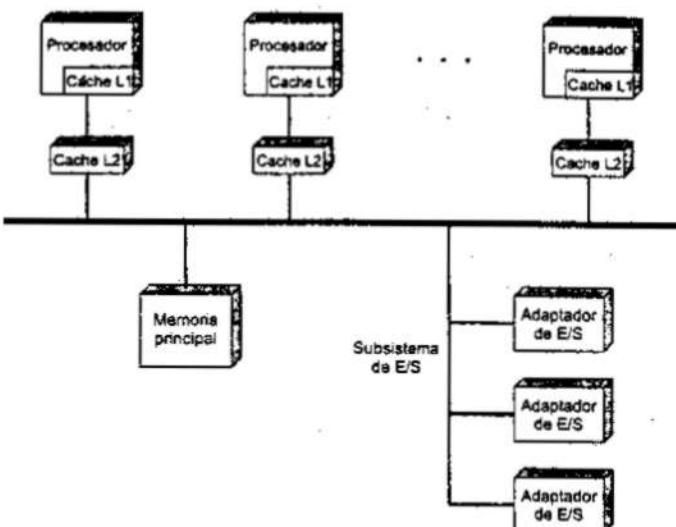


Figura 16.5: Organización de multiprocesador simétrico.

- **Flexibilidad:** Es generalmente sencillo expandir el sistema conectando más procesadores al bus.
- **Fiabilidad:** El bus es esencialmente un medio pasivo, y el fallo de cualquiera de los dispositivos conectados no provocaría el fallo de todo el sistema.

La principal desventaja de la organización de bus son las prestaciones. Todas las referencias a memoria pasan por el bus. En consecuencia, la velocidad del sistema está limitada por el tiempo de ciclo. Para mejorar las prestaciones, es deseable equipar a cada procesador con una memoria cache. Ésta reduciría drásticamente el número de accesos. Típicamente, los PC y las estaciones de trabajo de tipo SMP tienen dos niveles de cache: una cache L1 interna (en el mismo chip que el procesador), y una cache L2 externa o interna.

El uso de caches introduce algunas consideraciones de diseño nuevas. Puesto que cada cache local contiene una imagen de una parte de la memoria, si se altera una palabra en una cache, es conceible que eso podría invalidar una palabra en otra cache. Para evitarlo, se debe avisar a los otros procesadores de que se ha producido una actualización de memoria. Este problema se conoce como problema de *coherencia de cache*, que es resuelto típicamente por el hardware, más que por el sistema operativo. La Sección 16.3 trata este punto.

### Memoria multipuerto

La propuesta de memoria multipuerto permite el acceso directo e independiente a los módulos de memoria desde cada uno de los procesadores y los módulos de E/S (Figura 16.6). Se necesita una cierta lógica asociada a la memoria para resolver los conflictos. El método que se utiliza a menudo para resolver conflictos consiste en asignar prioridades fijas a cada puerto de memoria. Normalmente, la interfaz física y eléctrica en cada puerto es idéntica a la que aparece en un módulo de memoria de un sólo puerto. Así, se necesitan pocas o ninguna

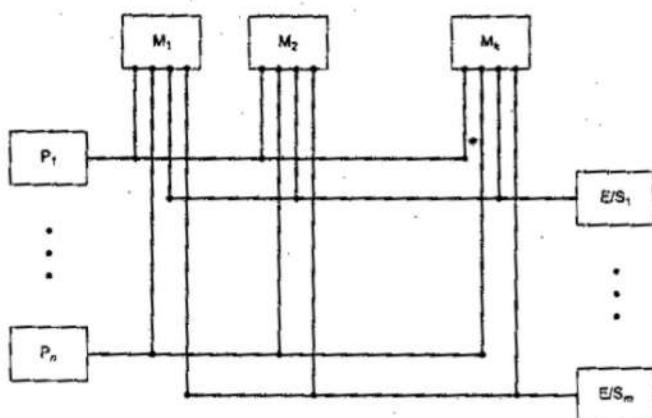


Figura 16.6. Memoria multipuerto.

modificación en los procesadores o en los módulos de E/S para acomodar la memoria multipuerto.

La aproximación de la memoria multipuerto es más compleja que la aproximación de bus, precisándose añadir al sistema de memoria una buena cantidad de lógica. No obstante, se consiguen mejores prestaciones, puesto que cada procesador tiene un camino dedicado a cada módulo de memoria. Otra ventaja del multipuerto es que permite configurar partes de la memoria como «privadas» para uno o más procesadores y/o módulos de E/S. Esta característica permite incrementar la seguridad frente a accesos no autorizados, y para el almacenamiento de rutinas de restablecimiento en zonas de memoria no susceptibles de ser modificadas por otros procesadores.

Otra cuestión más: se debe utilizar una estrategia de escritura directa («write-through») para controlar la cache, puesto que no hay forma de avisar a los otros procesadores de una actualización de memoria.

#### Unidad de control central

La unidad de control central encauza las distintas secuencias de datos entre los distintos módulos independientes: procesador, memoria y E/S. El controlador puede almacenar temporalmente peticiones, y realizar las funciones de arbitraje y temporización. Además, puede transmitir mensajes de estado y control entre los procesadores y alertar sobre cambios en las cachés.

Puesto que toda la lógica de coordinación de la configuración de multiprocesador se concentra en la unidad central de control, las interfaces de E/S, memoria y procesador, no sufren cambios esenciales. Esto proporciona la flexibilidad y la simplicidad de las interfaces en la aproximación de bus. Las desventajas clave de esta aproximación son que la unidad de control es bastante compleja, y que representa un cuello de botella potencial para las prestaciones.

La estructura de unidad de control central es bastante común en grandes computadores (mainframes) de varios procesadores, tales como los miembros más grandes de la familia S/370 de IBM. Hoy día, esta alternativa es poco frecuente.

## CONSIDERACIONES DE DISEÑO DE UN SISTEMA OPERATIVO DE MULTIPROCESADOR

Un sistema operativo de SMP gestiona los procesadores y demás recursos del computador, para que el usuario perciba un sólo sistema operativo controlando los recursos del sistema. De hecho, el computador debería parecer un sistema monoprocesador con multiprogramación. Tanto en un SMP como en un sistema monoprocesador, pueden estar activos varios trabajos o procesos al mismo tiempo, y es responsabilidad del sistema operativo planificar su ejecución y asignar los recursos. Un usuario puede desarrollar aplicaciones que utilizan varios procesos o varios hilos dentro de un proceso, sin tener en cuenta si se dispone de uno o de varios procesadores. Así, un sistema operativo de multiprocesador, debe proporcionar toda la funcionalidad de un sistema operativo con multiprogramación, más las características adicionales que permitan utilizar varios procesadores. Entre los puntos clave de diseño están:

- **Procesos concurrentes simultáneos:** Las rutinas del sistema operativo deben ser reentrantes, para permitir que varios procesadores puedan ejecutar simultáneamente el mismo código IS. Con varios procesadores ejecutando la misma o distintas partes del sistema operativo, las tablas y las estructuras de gestión del sistema operativo deben manejarse apropiadamente, para evitar bloqueos u operaciones no válidas.
- **Planificación:** La planificación puede realizarla cualquier procesador, por lo que deben evitarse los conflictos. El planificador debe asignar los procesos preparados a los procesadores disponibles.
- **Sincronización:** Puesto que hay varios procesos que pueden acceder a espacios de memoria y a recursos de E/S compartidos, debe proporcionarse una sincronización efectiva. La sincronización asegura la exclusión mutua y la ordenación de eventos.
- **Gestión de Memoria:** La gestión de memoria en un multiprocesador debe comprender todos los aspectos propios de los computadores monoprocesadores, discutidos en el Capítulo 7. Además, el sistema operativo debe explotar el paralelismo que proporciona el hardware (por ejemplo, las memorias multipuerto) para obtener mejores prestaciones. Los mecanismos de paginación en procesadores distintos deben coordinarse para mantener la consistencia cuando varios procesadores comparten una página o un segmento y para decidir sobre el reemplazo de páginas.
- **Fiabilidad y tolerancia ante los fallos:** El sistema operativo debería hacer posible una degradación gradual, cuando se produce un fallo en un procesador. El planificador y otros elementos del sistema operativo, deben reconocer la pérdida de un procesador y reestructurar las tablas de gestión en consecuencia.

## UN SMP COMO GRAN COMPUTADOR

La mayoría de los PC y estaciones de trabajo de tipo SMP utilizan una configuración basada en un bus, tal y como muestra la Figura 16.5. Resulta una aproximación alternativa, que se utiliza en las implementaciones más avanzadas de grandes computadores (mainframes) IBM S/390 [MAK97]. La Figura 16.6 ilustra la organización general del SMP S/390. Esta familia de sistemas incluye un procesador monoprocesador con un módulo de memoria principal, hasta tres procesadores adicionales, que se utilizan como procesadores de trabajo. Las configuraciones de memoria y de configuración son los siguientes:

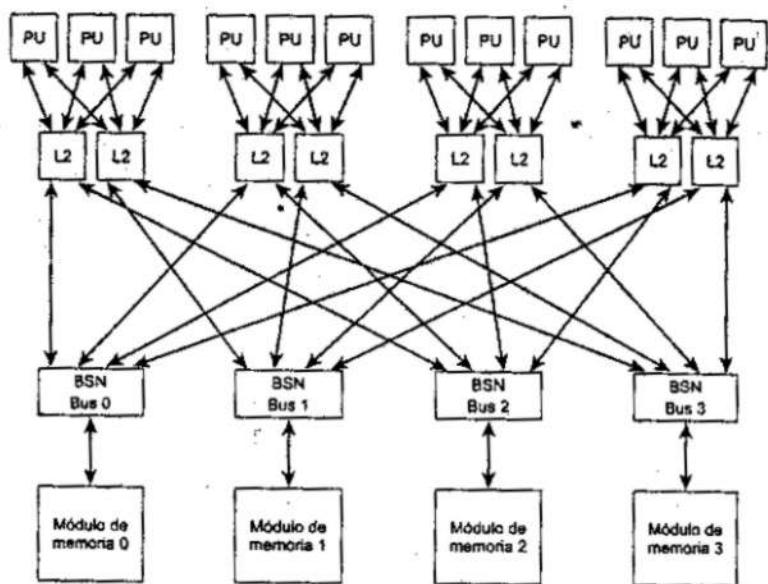


Figura 16.7. Organización del SMP S/390 de IBM.

- **Procesador (PU, Processor Unit):** Es un microprocesador CISC, en el que las instrucciones más frecuentemente ejecutadas se encuentran cableadas, mientras que las restantes se ejecutan por el «firmware». Cada PU incluye una cache L1 unificada (almacena datos e instrucciones) de 64 Kbytes. El tamaño de la cache L1 se ha determinado para que quepa en el mismo chip del PU, de forma que se consiga acceder a ella en un sólo ciclo.
- **Cache L2:** Cada cache L2 contiene 384 Kbytes. Las caches L2 se organizan en grupos de dos, de forma que cada grupo recibe accesos de tres PU, y proporciona acceso a todo el espacio de memoria principal.
- **Adaptador bus-red de interconexión (BSN, Bus-switching network):** Los BSN interconectan las caches L2 y la memoria principal. Cada BSN también incluye una cache de nivel 3 (L3), con un tamaño de 2 Mbytes.
- **Módulos de memoria:** Cada módulo dispone de 8 Gbytes de memoria, con una capacidad total de 32 Gbytes de memoria.

Hay una serie de características interesantes en la configuración del SMP S/390 que pasamos a discutir:

- Interconexión commutada
- Caches L2 compartidas
- Cache L3

### Interconexión conmutada

En los PC y las estaciones de trabajo de tipo SMP, es común utilizar una organización basada en un único bus compartido (Figura 16.5). Con esta organización, el bus pasa a ser un cuello de botella que afecta a la escalabilidad (escalado) de las prestaciones cuando se amplía el sistema) del diseño. El S/390 se enfrenta a este problema de dos formas. En primer lugar, la memoria principal se distribuye en cuatro módulos, cada uno con su propio controlador de almacenamiento que puede gestionar los accesos a memoria a velocidades elevadas. El tráfico de cargas desde memoria se reduce en un factor de cuatro, gracias a los cuatro caminos independientes que hay a las cuatro áreas en las que se ha dividido la memoria. En segundo lugar, las conexiones desde los procesadores (en concreto desde las caches L2) a cada módulo de memoria, no se producen como en un bus compartido, sino más bien como en un enlace punto-a-punto, donde cada enlace conecta un grupo de tres procesadores, a través de una cache L2, con un BSN. El BSN, por otra parte, realiza la función de un conmutador que puede encaminar datos entre sus cinco enlaces (cuatro enlaces con las memorias L2 y uno con el módulo de memoria). Con respecto a los cuatro enlaces con las caches L2, el BSN conecta los cuatro enlaces físicos a un bus de datos lógico. De esta forma, una señal proveniente de uno de los cuatro enlaces con las caches L2, se reenvía a cada uno de los restantes tres enlaces a las caches L2. Esto es necesario para permitir la coherencia de cache.

Hay que tener en cuenta que, aunque hay cuatro módulos de memoria distintas, cada PU y cada L2 sólo tienen dos puertos físicos en la dirección de memoria principal. Esto se debe a que cada cache L2 sólo almacena datos de la mitad de la memoria principal. Se necesitan dos caches para dar servicio a toda la memoria principal, y cada PU debe conectarse a ambas caches.

### Caches L2 compartidas

En un esquema típico de cache de dos niveles para un SMP, cada procesador tiene caches L1 y L2 propias. En los últimos años, ha aumentado el interés en el concepto de utilizar una cache L2 compartida. En una de las primeras versiones del SMP S/390, conocida como generación 3 (G3), IBM utilizaba caches L2 específicas para cada procesador. En las versiones más recientes (G4 y G5), se utilizan caches L2 compartidas. Dos consideraciones han causado este cambio:

1. En el cambio de la versión G3 a la G4, IBM pasó a utilizar microprocesadores con el doble de velocidad. Si se hubiese mantenido la organización G3, se hubiese producido un incremento significativo del tráfico a través del bus. Al mismo tiempo, se deseaba utilizar tantos componentes de las versiones G3 como fuese posible. A no ser que se mejorase significativamente el bus, el BSN podría llegar a ser un cuello de botella.
2. El análisis de las cargas de trabajo de un S/390 típico mostraba un nivel elevado de instrucciones y datos compartidos por los procesadores.

Estas consideraciones llevaron al equipo de diseño de la versión G4 del S/390 a considerar el uso de una o más caches L2 compartidas por varios procesadores (cada procesador dispone de una cache L1 interna). A primera vista, compartir la cache L2 podría parecer una mala idea. El acceso a memoria desde los procesadores podría ser más lento, debido a que los procesadores deben pugnar por el acceso a la cache L2. Sin embargo, si, de hecho, varios procesadores comparten un elevado volumen de datos, una cache compartida puede incrementar el rendimiento en lugar de disminuirlo, ya que los datos compartidos que se encuentran en la cache compartida se obtienen más rápidamente que si se debiera acceder a ellos a través del bus.

Una propuesta considerada en el diseño de la versión G4 del S/370 fue utilizar una única cache de gran tamaño y compartida por todos los procesadores. Esta alternativa podría proporcionar una mejora de las prestaciones del sistema gracias a la elevada eficiencia de la cache, pero se hubiera requerido un rediseño completo de la organización del bus del sistema. No obstante, el análisis de prestaciones indicaba que utilizar caches compartidas en cada uno de los BSN existentes, proporcionaría gran parte de las ventajas de utilizar caches compartidas con un tráfico reducido a través del bus. La eficacia de las caches compartidas se confirmó a través de medidas de prestaciones, que mostraban que la cache compartida mejoraba significativamente los niveles de aciertos frente al esquema de caches locales que se utilizaba en la organización de las versiones G3 [MAK97]. Estudios sobre la eficacia de las caches compartidas por los microprocesadores de los SMP de pequeña escala confirman que esta aproximación es correcta (por ejemplo [NAYF96]).

### Cache L3

Otra característica interesante del SMP S/390 es el uso de un tercer nivel de cache (L3)<sup>1</sup>. Las caches L2 se sitúan en los BSN y de esta forma, cada cache L3 proporciona un buffer entre las caches L2 y una tarjeta de memoria. La cache L3 reduce el retardo de acceso a los datos que no estén, ni en la cache L1, ni en la cache L2 del procesador que los solicita. El dato está disponible mucho más rápidamente que si hubiera que acceder a la memoria principal, en el caso de que la línea de cache esté siendo compartida con otro procesador y no se haya utilizado recientemente por el procesador que la solicita.

La Tabla 16.1 muestra las prestaciones que se obtienen en este SMP con tres niveles de cache, para el caso de una carga de trabajo típica de una aplicación comercial del S/390, que carga considerablemente el bus y la memoria [DOET97]<sup>2</sup>. La penalización de acceso al almacenamiento es el retardo entre la petición del dato a la jerarquía de caches, y el momento en el que se obtiene el primer bloque de datos de 16 bytes. La cache L1 muestra un porcentaje de aciertos del 89 %, de forma que el 11 % de los restantes accesos a memoria deben resolverse en el nivel L2, L3, o en memoria. De este 11 %, un 5 % se resuelven en la cache L2, y así sucesivamente. Utilizando estos tres niveles de cache, sólo un 3 %- de las referencias necesitan acceder a memoria. Sin el tercer nivel, la proporción de accesos a memoria principal se duplicaría.

**Tabla 16.1. Porcentajes típicos de aciertos de cache en una configuración del SMP S/390**

| Subsistema de memoria | Penalización de acceso (ciclos de PUI) | Tamaño de cache | Porcentaje de aciertos |
|-----------------------|----------------------------------------|-----------------|------------------------|
| Cache L1              | 1                                      | 32 KB           | 89                     |
| Cache L2              | 5                                      | 256 KB          | 5                      |
| Cache L3              | 14                                     | 2 MB            | 3                      |
| Memoria               | 32                                     | 8 GB            | 3                      |

<sup>1</sup> La literatura de IBM denomina a esta cache como cache L2.5. No parece que exista ninguna ventaja en este término, porque, de hecho, esta cache constituye un tercer nivel de memoria.

<sup>2</sup> Los datos corresponden a un sistema G3, que utiliza caches L2 locales para cada procesador. No obstante, estos resultados sugieren las prestaciones que se pueden esperar de las caches L2 compartidas de las versiones G4 y G5 del S/390.

### 16.3.1. COHERENCIA DE CACHE Y PROTOCOLO MESI

En los sistemas multiprocesador contemporáneos es común disponer de uno o dos niveles de cache asociados a cada procesador. Esta organización es esencial para conseguir unas prestaciones razonables. Esto, no obstante, origina un problema, conocido como el problema de *coherencia de cache*. La esencia del problema es ésta: pueden existir varias copias del mismo dato simultáneamente en caches diferentes y, si los procesadores actualizan sus copias, puede producirse una visión inconsistente de la memoria. En el Capítulo 4 se definieron dos políticas de escritura usuales:

- **Post-escritura (Write back):** Las operaciones de escritura se hacen usualmente sólo en la cache. La memoria principal sólo se actualiza cuando la línea de cache correspondiente se reemplaza.
- **Escritura directa (Write through):** Todas las operaciones de escritura se realizan en memoria principal a la vez que en la cache, asegurándose así de que el contenido de la memoria principal siempre es válido.

Resulta evidente que una política de post-escritura puede ocasionar inconsistencia. Si dos caches contienen la misma línea, y la línea se actualiza en una cache, la otra cache tendrá un valor no válido. Las lecturas siguientes a dicha línea producirán resultados no válidos. Incluso con la política de escritura directa, puede existir inconsistencia, a no ser que las otras caches comprueben los accesos a la memoria principal o reciban algún tipo de notificación directa de la escritura realizada.

En esta sección, revisaremos brevemente distintas aproximaciones al problema de la coherencia de cache y, después, nos centraremos en la aproximación más ampliamente utilizada: el protocolo MESI. Una versión de este protocolo se utiliza, tanto en implementaciones del Pentium II como del PowerPC.

El objetivo de un protocolo de coherencia de cache es situar las variables locales utilizadas recientemente en la cache apropiada, y mantenerlas allí para las distintas escrituras y lecturas, al mismo tiempo que se mantiene la consistencia de las variables compartidas que pudieran encontrarse en varias caches al mismo tiempo. Las aproximaciones de coherencia de cache generalmente se han dividido en aproximaciones software y hardware. Algunas implementaciones adoptan una estrategia que implica, tanto elementos software, como hardware. No obstante, la distinción entre aproximaciones software y hardware es todavía instructiva, y es comúnmente utilizada en la presentación de las estrategias de coherencia de cache.

### SOLUCIONES SOFTWARE

Los esquemas software de coherencia de cache intentan evitar la necesidad de circuitería y lógica hardware adicional, dejando que el compilador y el sistema operativo se encarguen del problema. Las propuestas software son atractivas, porque transfieren el costo de la detección de posibles problemas desde el hardware al software. Por otra parte, en el momento de la compilación, el software generalmente debe tomar ciertas decisiones conservadoras, que pueden ocasionar una utilización ineficiente de la cache.

Los mecanismos de coherencia basados en el compilador realizan un análisis del código para determinar qué datos pueden dar problemas al pasar a cache, y los marcan en consecuencia. Después, el sistema operativo, o el hardware, impiden que se pasen a cache los datos marcados como no almacenables en cache (non-cachable).

El enfoque más sencillo consiste en impedir que cualquier dato compartido pase a cache. Esto es demasiado conservador, puesto que una estructura de datos compartida puede utilizarse de manera exclusiva durante ciertos períodos de tiempo, y sólo para lectura en otros períodos. Es sólo durante aquellos períodos en los que al menos un procesador pueda actualizar una variable, y otro procesador pueda acceder a la variable, cuando hay que considerar la coherencia de cache.

Hay aproximaciones más eficientes, que analizan el código y determinan períodos seguros para las variables compartidas. El compilador inserta entonces instrucciones en el código generado para reforzar la coherencia de cache en los períodos críticos. Se han desarrollado cierto número de técnicas para realizar el análisis y para mejorar los resultados; véanse las revisiones de [LIL93] y [STEN90].

## SOLUCIONES HARDWARE

Las soluciones basadas en el hardware generalmente se denominan «protocolos de coherencia de cache». Estas soluciones permiten reconocer dinámicamente, en el momento de la ejecución, las situaciones de inconsistencias potenciales. Puesto que el problema se considera sólo en el momento en que aparece, existe un uso más efectivo de las caches, mejorándose las prestaciones en relación a las aproximaciones software. Además, estas aproximaciones son transparentes para el programador y el compilador, reduciendo la complejidad en el desarrollo del software.

Los esquemas hardware difieren en una serie de aspectos, que incluyen: el lugar donde se encuentra la información de estado de las líneas de datos, cómo se organiza esa información, dónde se impone la coherencia, y los mecanismos para imponerla. En general, los esquemas hardware se pueden dividir en dos categorías: protocolos de directorio y protocolos de sondeo (snoopy protocols).

### Protocolos de directorio

Los protocolos de directorio recogen y mantienen la información acerca de dónde residen las copias de las líneas. Usualmente, hay un controlador centralizado, que es parte del controlador de memoria principal, y un directorio, que se almacena en la memoria principal. El directorio contiene información de estado global en relación con los contenidos de las diferentes caches locales. Cuando el controlador individual de una cache hace una petición, el controlador centralizado comprueba y emite las órdenes precisas para la transferencia entre memoria y cache o entre distintas caches. Además, es responsable de mantener actualizada la información de estado; de esta forma, cualquier acción local que pueda afectar al estado global de una línea, debe comunicarse al controlador central.

Normalmente, el controlador posee información acerca de los procesadores que tienen una copia de cada línea. Antes de que un procesador pueda escribir en una copia local de una línea, debe solicitar al controlador el acceso exclusivo a dicha línea. Antes de ceder este acceso exclusivo, el controlador envía un mensaje a todos los procesadores con una copia de la línea en su cache, forzando a que cada procesador invalide su copia. Después de recibir el reconocimiento de cada uno de esos procesadores, el controlador cede el acceso exclusivo al procesador que lo solicitó. Cuando otro procesador intenta leer una línea cedida para acceso exclusivo de otro procesador, enviará una notificación de fallo de cache al controlador. Entonces, el controlador manda una orden al procesador que posee la línea requerida, para que lo vuelva a escribir en memoria principal. Ahora, la línea puede compartirse para lectura por el procesador original y el que solicitaba el acceso.

Los esquemas de directorio presentan las desventajas propias de tener un cuello de botella central y del costo de comunicación entre los controladores de las distintas caches y el controlador central. No obstante, son efectivos en sistemas de gran escala que poseen múltiples buses o algún esquema complejo de interconexión.

### Protocolos de sondeo («Snoopy Protocols»)

Los protocolos de sondeo distribuyen la responsabilidad de mantener la coherencia de cache entre todos los controladores de cache del multiprocesador. Una cache debe reconocer cuándo una línea de las que contiene está compartida con otras caches. Cuando se realiza una actualización en una línea de cache compartida, debe anunciarse a todas las otras caches mediante un mecanismo de difusión («broadcast»). Cada controlador de cache es capaz de sondear o «espiar» (snoop) la red para observar las notificaciones que se difunden, y reaccionar adecuadamente.

Los protocolos de sondeo se adaptan bien a los multiprocesadores basados en un bus, puesto que el bus compartido proporciona una forma sencilla para la difusión y el sondeo. No obstante, puesto que uno de los objetivos de utilizar caches locales es evitar los accesos al bus, hay que cuidar que el incremento en el tráfico del bus que requiere la difusión y el sondeo, no anule los beneficios de las caches locales.

Se han explorado dos enfoques básicos del protocolo de sondeo: invalidar-si-escritura (write-invalidate) y actualizar-si-escritura o difundir-escritura («write-update» o «write-broadcast»). Con un protocolo de invalidar-si-escritura, puede haber múltiples procesadores que leen, pero un sólo procesador que escribe en un momento dado. Inicialmente, una línea puede compartirse por varias caches con el propósito de lectura. Cuando se quiere hacer una escritura en la línea de una cache, ésta envía primero una notificación que invalida la línea en las otras caches, haciendo que dicha línea sea exclusiva para la cache donde se va a escribir. Una vez que la línea es exclusiva, el procesador puede realizar escrituras locales en la misma hasta que otro procesador solicita la misma línea.

Con un protocolo de actualizar-si-escritura, puede haber varios procesadores que escriben, igual que varios procesadores que leen. Cuando un procesador desea escribir en una línea compartida, la palabra a actualizar se distribuye a los demás, y las caches que contienen esa línea lo pueden actualizar.

Ninguna de estas dos aproximaciones es mejor que la otra en todas las situaciones. Las prestaciones dependen del número de caches locales y del patrón de escrituras y lecturas de memoria. Algunos sistemas implementan protocolos adaptativos, que utilizan, tanto el mecanismo de invalidar-si-escritura, como el de actualizar-si-escritura.

La aproximación de invalidar-si-escritura es la que más se utiliza en los multiprocesadores comerciales, tales como los basados en el Pentium II y en el PowerPC. Se marca el estado de cada línea de cache (usando dos bits adicionales en el campo de marca de la cache) como modificado (modified), exclusivo (exclusive), compartido (shared), o no válido (invalid). Por esta razón, el protocolo de invalidar-si-escritura se llama MESI, de las iniciales de los estados en inglés. En el Capítulo 4 ya consideramos el protocolo MESI en relación con el protocolo de coordinación entre los niveles 1 y 2 de caches locales. En el resto de esta sección, consideraremos su uso para las caches locales en un multiprocesador. Por simplicidad, en la presentación no se examinan los mecanismos necesarios para la coordinación local del nivel 1 y el 2 al mismo tiempo que los de coordinación en el multiprocesador distribuido. Esto no añade ningún concepto nuevo y, en cambio, complicaría considerablemente la discusión.

## EL PROTOCOLO MESI

Recuérdese del Capítulo 4 que, con el protocolo MESI, la cache de datos incluye dos bits de estado en la marca, puesto que cada línea puede estar en uno de estos cuatro estados:

- **Modificado (Modified):** La línea de cache ha sido modificada (es distinta a su valor en memoria principal), y está disponible sólo en esta cache.
- **Exclusivo (Exclusive):** La línea de cache tiene el mismo contenido que en memoria principal, y no está presente en ninguna otra cache.
- **Compartido (Shared):** La línea de cache tiene el mismo contenido que en memoria principal, y puede estar presente en otra cache.
- **No válido (Invalid):** La línea de cache no contiene datos válidos.

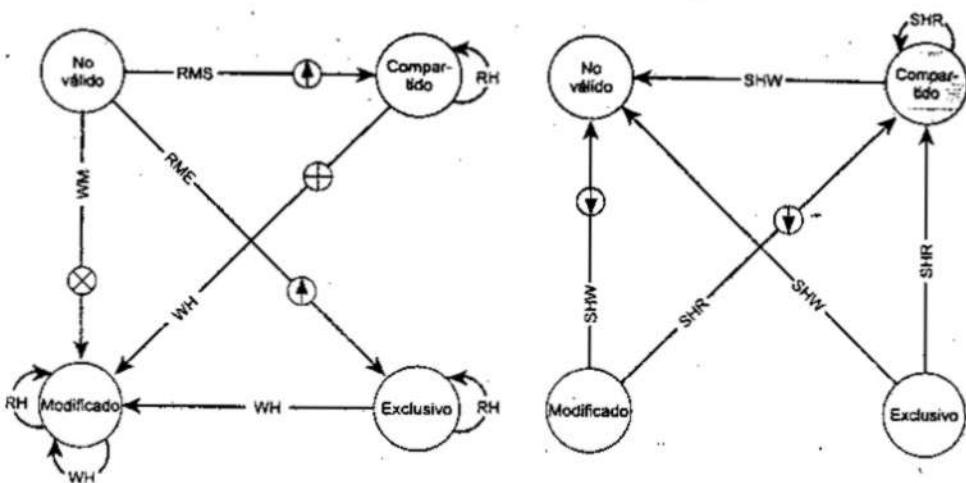
La Figura 16.8 muestra el diagrama de estados del protocolo MESI. Cada línea de la cache tiene sus propios bits de estado y su propia implementación del diagrama de estados. La Figura 16.8a muestra las transiciones que se producen debido a las acciones iniciadas por el procesador al que pertenece la cache. La Figura 16.8b describe las transiciones ocasionadas por eventos que se producen en el bus común. La presentación de las acciones ocasionadas por el procesador en un diagrama de estado, distinto al utilizado para las acciones ocasionadas por eventos del bus, ayuda a clarificar la lógica del protocolo MESI. Sin embargo, en cada instante, una línea de cache está en un único estado. Si el siguiente evento proviene del procesador al que pertenece la cache, entonces la transición se describe en la Figura 16.8a, y si el siguiente evento proviene del bus, la transición se describe en la Figura 16.8b. A continuación se describen cada una de las transiciones.

### Fallo de lectura

Cuando se produce un fallo de lectura en la cache local, el procesador inicia una lectura en memoria, para acceder a la línea de memoria principal que contiene la dirección que no está en cache. El procesador inserta una señal en el bus, que avisa a todos los otros procesadores/caches para que sondeen la transacción. Hay una serie de posibilidades:

- Si una de las caches tiene una copia limpia (clean) de la línea (es decir, una copia no modificada desde que se leyó de memoria) en el estado exclusivo, devuelve una señal, indicando que comparte la línea. El procesador que envía esta señal pasa su copia al estado compartido, y el procesador inicial lee la línea, y pasa el estado de ésta en su cache de no válido a compartido.
- Si una o más caches tienen una copia limpia de la línea en estado compartido, cada una de ellas indica que comparte la línea. El procesador inicial lee la línea, y pasa su estado en cache de no válido a compartido.
- Si una de las caches tiene una copia modificada de la línea, entonces esa cache bloquea la lectura de memoria, y proporciona la línea a la cache que la solicita a través del bus compartido. La cache que proporciona la línea pasa ésta del estado modificado al estado compartido<sup>3</sup>.

<sup>3</sup> En algunas implementaciones, la cache con la línea modificada indica al procesador que inició el acceso que vuelve a intentarlo. Mientras tanto, el procesador con la copia modificada accede al bus, actualiza la línea modificada en memoria principal, y pasa la línea desde el estado modificado al compartido. Después, el procesador que solicitó el acceso lo intenta de nuevo, y encuentra que uno o varios procesadores tienen una copia actualizada de la línea en estado compartido, como se describe en el punto precedente.



(a) Línea en el procesador que inicia la transferencia

(b) Línea en una caché sondeando el bus

|     |                                                            |
|-----|------------------------------------------------------------|
| RH  | Acierto de lectura                                         |
| RMS | Fallo de lectura, compartida                               |
| RME | Fallo de lectura, exclusiva                                |
| WH  | Acierto de escritura                                       |
| WM  | Fallo de escritura                                         |
| SHR | Acierto de sondeo en lectura                               |
| SHW | Acierto de sondeo en escritura o lectura-para-modificación |

- |   |                                                  |
|---|--------------------------------------------------|
| ⊖ | Escribir en memoria la línea de cache modificada |
| ⊕ | Invalidez de transacción                         |
| ⊗ | Lectura-para-modificación (RWITM)                |
| ↑ | Cargar línea de cache                            |

Figura 16.8. Diagrama de transición de estados MESI.

- Si ninguna otra caché tiene una copia de la línea (limpia o en estado modificado), no se envía ninguna señal. El procesador lee la línea, y cambia el estado de la línea en su cache de no-válido a exclusivo.

### Acierto de lectura

Cuando se produce un acierto en una lectura dentro de una línea presente en la cache local, el procesador simplemente lee el dato solicitado. No hay ningún cambio de estado: se mantiene como modificado, compartido o exclusivo.

### Fallo de escritura

Cuando se produce un fallo en una escritura en la cache local, el procesador comienza una lectura de memoria, para acceder a la línea de memoria principal que contiene la dirección que no está en cache. Para ello, el procesador envía una señal a través del bus que indica lectura-para-modificación (RWITM, read-with-intent-to-modify). Cuando se carga la línea, se

marca inmediatamente como modificada. Con respecto a las otras caches, hay dos escenarios posibles previos a la carga del bloque de datos.

En el primero, otro procesador puede haber modificado una copia de esta línea (estado modificado) en su cache. En este caso, el procesador en cuestión indica al procesador inicial que hay una copia modificada de la línea. El procesador inicial deja libre el bus y espera. El procesador con la copia modificada accede al bus, escribe la línea modificada en la memoria principal, y cambia el estado de la línea en cache a no-válido (puesto que el procesador inicial va a modificar esta línea). A continuación, el procesador inicial activará una señal RWITM en el bus, y luego lee la línea de la memoria principal, modifica la línea en la cache, y la marca con el estado modificado.

El segundo escenario corresponde al caso de que ninguna cache tenga una copia modificada de la línea en su cache. En este caso, no se responde con ninguna señal, y el procesador inicial prosigue leyendo la línea y modificándola. Mientras tanto, si una o más caches tienen copias limpias de la línea en estado compartido, cada cache pasa su copia a estado no válido. Si una cache tiene una copia de la línea no modificada en estado exclusivo, la invalida.

### Acierto de escritura

Cuando se produce un acierto de escritura en una línea de cache local, el efecto depende del estado de la línea:

- **Compartido:** Antes de realizar la actualización, el procesador debe conseguir el acceso exclusivo a la línea. El procesador señala esta intención a través del bus. Todo procesador que tenga una copia de la línea en su cache la cambia del estado compartido al no válido. Después, el procesador inicial actualiza la línea, y cambia su copia de la línea del estado compartido al estado modificado.
- **Exclusivo:** Puesto que el procesador tiene el control exclusivo de esta línea, simplemente la actualiza, y cambia el estado de la línea de exclusivo a modificado.
- **Modificado:** Puesto que el procesador tiene el control exclusivo de la línea, y la tiene marcada en el estado modificado, sólo tiene que actualizarla.

### Consistencia de caches L1-L2

Hasta ahora se han descrito los protocolos de coherencia de cache en términos de la cooperación entre las caches conectadas al mismo bus o a otro sistema de interconexión utilizado por un SMP. Normalmente, estas caches son caches L2, y cada procesador tiene además una cache L1 que no está conectada directamente al bus y que, por lo tanto, no puede participar en un protocolo de sondeo. De esta forma, se necesita un esquema para mantener la integridad de los datos en los dos niveles de cache y en todas las caches del SMP.

La estrategia utilizada consiste en extender el protocolo MESI (o cualquier otro protocolo de coherencia de cache) a las caches L1. Así, cada línea de la cache L1 incluye bits para indicar el estado. Básicamente, el objetivo es el siguiente: para cualquier línea, presente, tanto en la una cache L2 como en su correspondiente cache L1, el estado de la línea en L1 debe seguir el trayecto del estado de la línea en L2. Una forma sencilla de hacer esto es adoptar una política de escritura directa en la cache L1, pero escribiendo sobre la cache L2, y no sobre la memoria principal. La política de escritura directa en L1 hace que cualquier modificación en una línea de L1 se propague a la cache L2, haciéndose visible a las otras caches L2. El uso de la política de escritura directa en L1 requiere que el contenido de L1 sea un subconjunto del contenido de L2. Esto sugiere además, que la asociatividad de L2 debería ser

igual o mayor que la asociatividad de L1. La política de escritura directa para L1 se utiliza en el SMP IBM S/390.

Si la cache L1 utiliza una política de post-escritura, la relación entre las dos caches es más compleja. Hay varias aproximaciones para mantener la coherencia en este caso. El procedimiento utilizado en el Pentium II se describe con detalle en [SHAN98].

## 16.4. «CLUSTERS»

Una de las áreas actualmente más activas es el diseño de «clusters». Los «clusters» constituyen la alternativa a los multiprocesadores simétricos (SMP) para disponer de prestaciones y disponibilidad elevadas, y son particularmente atractivos en aplicaciones propias de un servidor. Se puede definir un «cluster» como un grupo de computadores completos interconectados, que trabajan conjuntamente como un único recurso de cómputo, creándose la ilusión de que se trata de una sola máquina. El término *computador completo* hace referencia a un sistema que puede funcionar por sí solo, independientemente del «cluster». Usualmente, en la literatura, cada computador del «cluster» se denomina *nodo*.

En [BREW97] se enumeran cuatro beneficios que pueden conseguirse con un «cluster». Estos beneficios también pueden contemplarse como objetivos o requisitos de diseño:

- **Escalabilidad absoluta:** Es posible configurar «clusters» grandes, que incluso superan las prestaciones de los computadores independientes más potentes. Un cluster puede tener decenas de máquinas, cada una de las cuales puede ser un multiprocesador.
- **Escalabilidad incremental:** Un «cluster» se configura de forma que sea posible añadir nuevos sistemas al «cluster» en ampliaciones sucesivas. Así, un usuario puede comenzar con un sistema modesto y ampliarlo a medida que lo necesite, sin tener que sustituir el sistema de que dispone por uno nuevo que proporcione mayores prestaciones.
- **Alta disponibilidad:** Puesto que cada nodo del «cluster» es un computador autónomo, el fallo de uno de los nodos no significa la pérdida del servicio. En muchos casos, es el software el que proporciona automáticamente la tolerancia a fallos.
- **Mejor relación precio/prestaciones:** Al utilizar elementos estandarizados, es posible configurar un «cluster» con mayor o igual potencia de cómputo que un computador independiente mayor, a mucho menos costo.

## CONFIGURACIONES DE «CLUSTERS»

En la literatura, los «clusters» se clasifican de formas muy diversas. Quizá la clasificación más sencilla es la que considera si los computadores comparten el acceso al mismo disco. La Figura 16.9a muestra un «cluster» de dos nodos, en el que la interconexión se realiza mediante un enlace de alta velocidad, que puede utilizarse para intercambiar mensajes que coordinan la actividad del «cluster». El enlace puede ser una LAN que se comparte con otros computadores no incluidos en el «cluster», o puede tratarse de un medio de interconexión específico. En este último caso, uno o varios computadores del «cluster» tendrán un enlace a una LAN o a una WAN, de forma que sea posible la conexión entre el «cluster», actuando como servidor, y los clientes remotos. Considerese que, en la figura, cada computador se representa como multiprocesador. Esto no es imprescindible pero proporciona un aumento tanto de las prestaciones como de la disponibilidad.

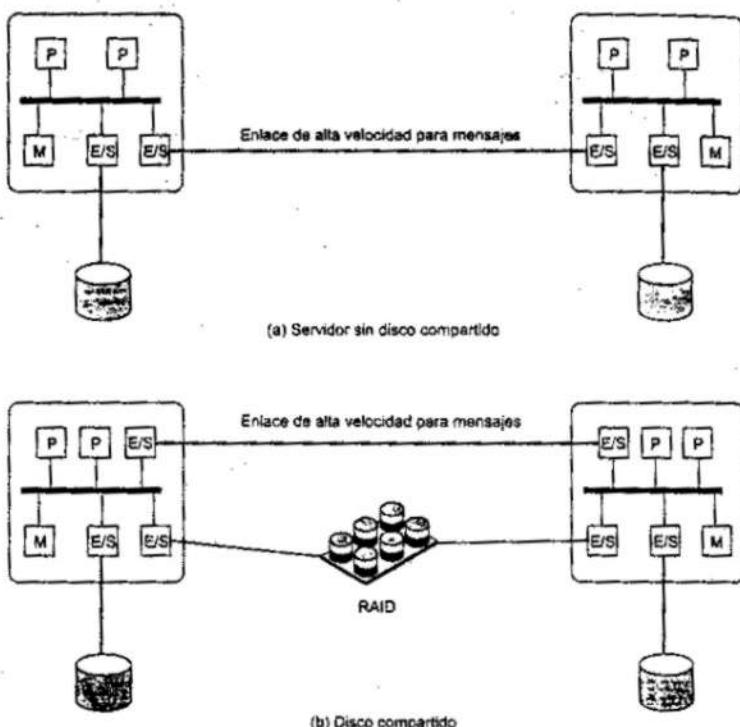


Figura 16.9. Configuraciones de «clusters».

En la clasificación simple de la Figura 16.9, la otra alternativa representada es un «cluster» con disco compartido. En este caso, generalmente también existe un enlace entre los nodos. Además, existe un subsistema de disco, que se conecta directamente a los computadores del «cluster». En la figura, el subsistema de disco común es un RAID. El uso del RAID o algún tipo de tecnología de discos redundantes, es común en los «clusters» para que la elevada disponibilidad que se consigue con la presencia de varios computadores, no se vea comprometida por un disco compartido que pueda convertirse en un único punto de fallo.

Se puede tener un panorama más claro del abanico de posibilidades de configuración de un «cluster» a partir de las alternativas funcionales. Un documento de Hewlett Packard [HP96] proporciona una clasificación muy útil a partir de las características de funcionamiento (Tabla 6.2). Se discute a continuación.

Existe un procedimiento conocido como espera pasiva (passive standby), bastante antiguo y común, que consiste simplemente en mantener toda la carga de trabajo en un computador, mientras que otro permanece inactivo hasta tomar el relevo del primero, cuando se produce un fallo de éste. Para coordinar las máquinas, el computador activo, o primario, envía un «mensaje de actividad» (heartbeat message) al computador en espera. En el caso de que

Tabla 16.2. Métodos de configuración de Clusters: beneficios y limitaciones

| Método de configuración del «cluster» | Descripción                                                                                                                                           | Beneficios                                                                                                     | Limitaciones                                                                                                    |
|---------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------|
| Espera pasiva                         | Un servidor secundario sustituye al servidor primario en caso de que falle.                                                                           | Fácil de implementar.                                                                                          | Alto coste debido a que el servidor secundario no está disponible para otras tareas de procesamiento.           |
| Secundario activo                     | El servidor secundario también se utiliza para tareas de procesamiento.                                                                               | Coste reducido porque los servidores secundarios pueden utilizarse para el procesamiento.                      | Aumenta la complejidad.                                                                                         |
| Servidores separados                  | Cada servidor tiene su propio disco. Los datos se copian desde el servidor primario al secundario.                                                    | Alta disponibilidad.                                                                                           | Penalización elevada en la red y el servidor debido a las operaciones de copia.                                 |
| Servidores conectados a discos        | Los servidores se conectan a los mismos discos, pero cada servidor tiene sus propios discos. Si falla un servidor, otro servidor accede a sus discos. | Penalización reducida en la red y en el servidor, debido a la eliminación de las operaciones de copia.         | Usualmente, se necesitan discos espejo o tecnología RAID para afrontar el riesgo de fallo de disco.             |
| Servidores compartiendo discos        | Los servidores comparten simultáneamente el acceso a los discos.                                                                                      | Penalización baja en la red y en el servidor. Riesgo reducido de fallo del sistema debido a un fallo de disco. | Se necesita software de control de acceso exclusivo. Usualmente se utiliza con discos espejo o tecnología RAID. |

estos mensajes dejan de llegar, el computador en espera asume que el servidor ha fallado y se pone en marcha. Esta alternativa aumenta la disponibilidad, pero no las prestaciones. Es más, si los dos computadores sólo se intercambian el mensaje de actividad, y si no tienen discos comunes, entonces el computador en espera constituye un computador de reserva para ejecutar procesos, pero no tiene acceso a las bases de datos gestionadas por el primer computador.

La espera activa no suele aplicarse en un «cluster». El término «cluster» hace referencia a varios computadores interconectados que se encuentran activos realizando algún tipo de procesamiento a la vez, y que proporcionan una imagen de sistema único al exterior. El término **secundario activo** (*active secondary*) se utiliza a menudo para referirse a esta configuración. Se pueden distinguir tres métodos para componer el «cluster»: con servidores separados (*separate server*), sin compartir nada (*shared nothing*), y compartiendo memoria (*shared memory*).

En el primero de los métodos, cada computador es un servidor independiente con su propio disco, y no existen discos compartidos por los sistemas (Figura 16.9a). Esta organización proporciona, tanto disponibilidad como unas prestaciones elevadas. Se precisa algún tipo de software de gestión o planificación para asignar a los servidores las peticiones que se van recibiendo de los clientes, de forma que se equilibre la carga de los mismos y se consiga una utilización elevada. Es deseable tener una cierta capacidad de fallos, lo que significa que, si un computador falla mientras está ejecutando una aplicación, otro computador del «cluster» puede acceder a la aplicación y completarla. Para que esto suceda, los datos se deben copiar constantemente en los distintos sistemas, de manera que cada procesador pueda acceder a los

datos actuales de los demás. El coste que supone este intercambio de datos ocasiona una penalización en las prestaciones, pero es el precio por conseguir una disponibilidad elevada.

Para reducir el coste que suponen las comunicaciones, la mayoría de los «clusters» están constituidos por servidores conectados a discos comunes (Figura 16.9b). Una variación a esta alternativa se designa como configuración sin compartir nada (*shred nothing*). En esta aproximación, los discos comunes se dividen en volúmenes diferentes, y cada volumen pasa a pertenecer a un sólo procesador. Si el computador falla, el «cluster» se debe reconfigurar, de forma que los volúmenes que pertenecían al computador defectuoso pasen a los otros computadores.

También es posible hacer que los computadores comparten el disco al mismo tiempo (aproximación de disco compartido), para que todos los computadores tengan acceso a todos los volúmenes de todos los discos. Esta aproximación necesita utilizar algún tipo de procedimiento para el acceso exclusivo, para asegurar que, en un momento dado, sólo un computador puede acceder a los datos.

## CONSIDERACIONES EN EL DISEÑO DEL SISTEMA OPERATIVO

Un completo aprovechamiento de la configuración hardware de un «cluster», exige una cierta ampliación de un sistema operativo propio de un único computador.

### Gestión de los fallos

La forma en que se actúa frente a un fallo en un «cluster» depende del tipo de configuración del mismo (Tabla 16.2). En general, se pueden utilizar dos alternativas para enfrentarse a los fallos: «clusters» de alta disponibilidad y «clusters» tolerantes a fallos. Un «cluster» de alta disponibilidad es el que ofrece una probabilidad elevada de que todos sus recursos estén en servicio. Si se produce un fallo, tal como la caída del sistema o la pérdida de un volumen de disco, se pierden las tareas en curso. Si una de esas tareas se reinicia, puede ser un computador distinto el que le de servicio. No obstante, el sistema operativo del «cluster» no garantiza el estado de las transacciones ejecutadas parcialmente. Esto debería gestionarse en el nivel de aplicación.

Un «cluster» tolerante a fallos garantiza que todos los recursos estén disponibles. Esto se consigue utilizando discos compartidos redundantes, y mecanismos para salvar las transacciones no terminadas y concluir las transacciones completadas.

La función de comutar aplicaciones y datos en el «cluster», desde un sistema defectuoso a otro alternativo, se denomina *transferencia por fallo* (failover). Una función adicional es la restauración de las aplicaciones y los datos por el sistema original, una vez superado el fallo; se denomina *recuperación después de un fallo* (fallback). La recuperación puede hacerse automáticamente, pero esto es deseable sólo si el fallo ha sido completamente reparado y es poco probable que vuelva a producirse. En caso contrario, la recuperación automática puede ocasionar transferencias continuas, en un sentido y en otro, de programas y datos, debido a la reaparición de un fallo, y se producirán problemas en cuanto a las prestaciones y a la recuperación.

### Equilibrado de carga

Un «cluster» necesita una capacidad efectiva para equilibrar la carga entre los computadores disponibles. Esta capacidad es necesaria para satisfacer el requisito de la escalabilidad incre-

mental. Cuando un computador se añade al «cluster», las aplicaciones encargadas de la asignación de tareas deberían incluir automáticamente a dicho computador junto con los restantes, para distribuir la carga de forma equilibrada. Los mecanismos de un nivel de software intermedio entre el sistema operativo y las aplicaciones (middleware) necesitan reconocer los servicios que pueden aparecer en los distintos miembros del «cluster», y pueden migrar desde un miembro a otro.

### «CLUSTERS» FRENTE A SMP

Tanto los «clusters» como los multiprocesadores simétricos constituyen configuraciones con varios procesadores que pueden ejecutar aplicaciones con una alta demanda de recursos. Ambas soluciones están disponibles comercialmente, aunque los SMP lo están desde hace más tiempo.

La principal ventaja de un SMP es que resulta más fácil de gestionar y configurar que un «cluster». El SMP está mucho más cerca del modelo de computador de un sólo procesador, para el que están disponibles casi todas las aplicaciones. El principal cambio que se necesita para pasar de un computador monoprocesador a un SMP se refiere al funcionamiento del planificador. Otra ventaja de un SMP es que necesita menos espacio físico y consume menos energía que un «cluster» comparable. Una última e importante ventaja es que los SMP son plataformas estables y bien establecidas.

Con el tiempo, no obstante, las ventajas de los «clusters» serán las que, probablemente, harán que sean éstos los que dominen en el mercado de servidores de altas prestaciones. Los «clusters» son superiores a los SMP en términos de escalabilidad absoluta e incremental y, además, también son superiores en términos de disponibilidad, puesto que todos los componentes del sistema pueden hacerse altamente redundantes.

## 16.5. ACCESO NO UNIFORME A MEMORIA

En términos comerciales, las dos alternativas para ejecutar aplicaciones en un sistema con varios procesadores son los SMP y los «clusters». Durante algunos años, otra alternativa conocida como acceso no uniforme a memoria (NUMA, nonuniform memory access) ha sido objeto de investigación y, recientemente, han aparecido computadores NUMA comerciales.

Antes de continuar, se deberían definir algunos términos que se encuentran a menudo en la literatura de computadores NUMA.

- **Acceso uniforme a memoria (UMA, uniform memory access):** Todos los procesadores pueden acceder a toda la memoria principal utilizando instrucciones de carga y almacenamiento. El tiempo de acceso de un procesador a cualquier región de la memoria es el mismo. El tiempo de acceso a memoria por parte de todos los procesadores es el mismo. La organización SMP, discutida en las Secciones 16.2 y 16.3, es una organización UMA.
- **Acceso no uniforme a memoria (NUMA, nonuniform memory access):** Todos los procesadores tienen acceso a todas las partes de memoria principal utilizando instrucciones de carga y almacenamiento. El tiempo de acceso a memoria de un procesador depende de la región a la que se acceda. La última frase es cierta para todos los procesadores; no obstante, para procesadores distintos, las regiones de memoria que son más lentas o más rápidas son diferentes.

- **NUMA con coherencia de cache (CC-NUMA, cache-coherent NUMA):** Un computador NUMA en el que la coherencia de cache se mantiene en todas las caches de los distintos procesadores.
- Un sistema NUMA sin coherencia de cache, es más o menos equivalente a un «cluster». Las alternativas que han despertado más interés comercial son los CC-NUMA, que son bastante diferentes de los SMP y los «clusters». Usualmente, pero desafortunadamente no siempre, estos sistemas se denominan en la literatura comercial «sistemas CC-NUMA». Esta sección se refiere únicamente a ellos.

## MOTIVACIÓN

En un SMP existe un límite práctico en el número de procesadores que pueden utilizarse. Un esquema de cache eficaz reduce el tráfico en el bus entre los procesadores y la memoria principal. A medida que el número de procesadores se incrementa, el tráfico en el bus también aumenta. Además, el bus se utiliza para intercambiar señales de coherencia de cache, añadiendo más carga. A partir de cierto momento, el bus pasa a ser el cuello de botella para las prestaciones. La degradación de las prestaciones parece que limita el número de procesadores en una configuración SMP a entre 16 y 64 procesadores. Por ejemplo, el SMP Power Challenge de Silicon Graphics, está limitado a 64 procesadores MIPS R10000 para un sólo sistema, puesto que más allá de este número las prestaciones se degradan sustancialmente.

Precisamente, el límite de procesadores en un SMP es uno de los motivos para el desarrollo de los «clusters». Sin embargo, en un «cluster» cada nodo tiene su propia memoria principal privada, y las aplicaciones no «ven» la memoria global. De hecho, la coherencia se mantiene mediante software, en lugar de mediante hardware. Esta granularidad de memoria afecta a las prestaciones, y, para conseguir el máximo nivel en ellas, el software debe ajustarse a este entorno. Una alternativa para conseguir multiprocesamiento a gran escala mientras se mantienen las características SMP, son los computadores NUMA. Por ejemplo, el NUMA Origin de Silicon Graphics, está diseñado para incluir hasta 1.024 procesadores MIPS R10000 [WHIT97], y el computador NUMA-Q de Sequent se ha diseñado para soportar hasta 252 procesadores Pentium II [LOVE96].

El objetivo de un computador NUMA es mantener una memoria transparente desde cualquier parte del sistema, al tiempo que se permiten varios nodos de multiprocesador, cada uno con su propio bus u otro sistema de interconexión interna.

## ORGANIZACIÓN

La Figura 16.10 muestra una organización CC-NUMA típica. Hay varios nodos independientes, cada uno de los cuales es, de hecho, un SMP. Así, cada nodo contiene varios procesadores, cada uno con sus caches L1 y L2, más memoria principal. El nodo es el bloque básico de construcción de toda la organización CC-NUMA. Por ejemplo, cada nodo del Origin de Silicon Graphics incluye dos microprocesadores MIPS R10000; cada nodo del computador NUMA-Q de Sequent incluye cuatro procesadores Pentium II. Los nodos se interconectan a través de un medio de comunicación, que podría ser algún mecanismo de conmutación, un anillo, o algún tipo de red.

Cada nodo de un sistema CC-NUMA incluye cierta cantidad de memoria principal. Sin embargo, desde el punto de vista de los procesadores, existe un único espacio de memoria direccionable en el que, a cada posición, se asocia una única dirección válida para todo el sistema. Cuando un procesador inicia un acceso a memoria, si la posición solicitada no se

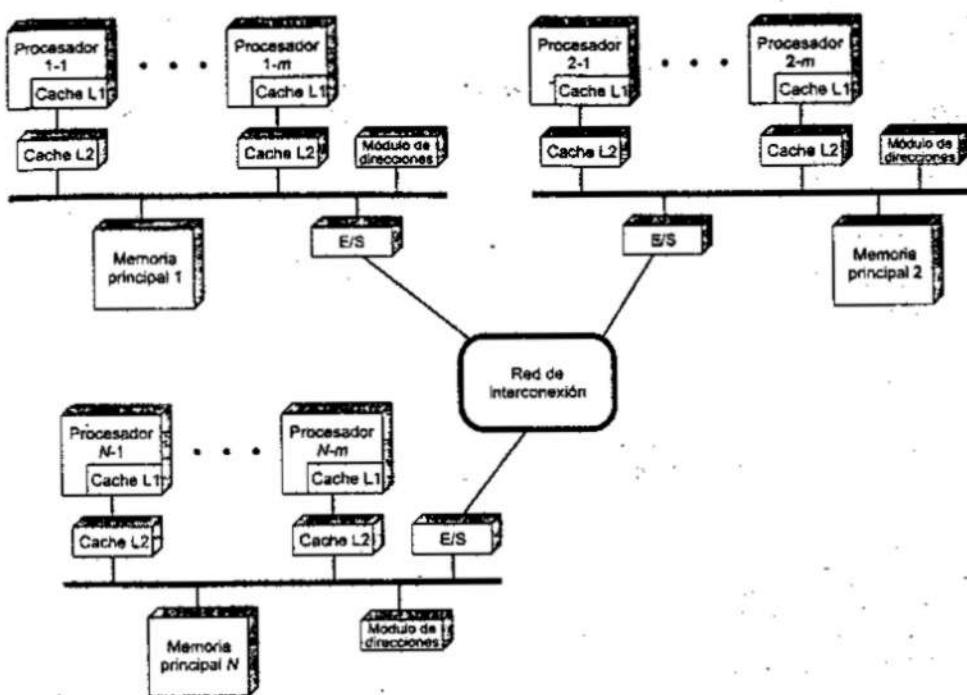


Figura 16.10. Organización CC-NUMA.

encuentra en la cache del procesador, entonces la cache L2 inicia una operación de captación. Si la línea deseada está en una posición remota de la memoria principal, la línea se capta a través del bus local. Si la línea solicitada está en una porción remota de la memoria, entonces se envía una petición automática para captar dicha línea a través de la red de interconexión y se proporciona, a través del bus local, a la cache que lo solicitaba en dicho bus. Toda esta actividad es automática y transparente al procesador y a su cache.

En esta configuración, la coherencia de cache es una cuestión central. Aunque las implementaciones pueden diferir en ciertos detalles, en términos generales podemos decir que cada nodo debe mantener algún tipo de orden de directorios, que dé una indicación de la situación de varias partes de la memoria, y también de la información de estado de la cache. Para ver como trabaja este esquema, utilizaremos un ejemplo tomado de [PFIS98]. Supóngase que el procesador 3 del nodo 2 (P2-3) solicita acceder a la posición 798, que está en la memoria del nodo 1. Se producirá la siguiente secuencia:

1. P2-3 genera la petición de lectura de la posición 798 a través del bus del nodo 2.
2. El módulo de direcciones del nodo 2 detecta la petición de acceso a memoria, y determina que la dirección está en el módulo 1.
3. El módulo de direcciones del nodo 2 envía la petición al nodo 1, y ésta es recogida por el módulo de direcciones del nodo 1.

4. El módulo de direcciones del nodo 1, actuando en sustitución del procesador P2-3, solicita la lectura de la posición 798.
5. La memoria principal del nodo 1 responde, situando el dato solicitado en el bus.
6. El módulo de direcciones del nodo 1 toma el dato del bus.
7. El dato se transfiere al módulo de direcciones del nodo 2.
8. El módulo de direcciones del nodo 2 pone el dato en el bus de dicho nodo, actuando como si fuera la memoria que originariamente proporcionó el dato.
9. El dato se lee pasando a la cache del procesador P2-3, desde donde se proporciona al procesador.

La secuencia precedente explica cómo se lee un dato desde una memoria remota, utilizando mecanismos hardware que hacen la transacción transparente al procesador. Sobre estos mecanismos, se necesita algún tipo de protocolo de coherencia de cache. Los distintos sistemas se diferencian según la forma exacta en que se implemente ese protocolo. Aquí faremos únicamente algunas precisiones. En primer lugar, si en la secuencia previa el módulo de direcciones del nodo 1 guarda un registro que indique que alguna cache remota tiene una copia de la línea que contiene a la posición 798, se necesita un protocolo cooperativo que tenga en cuenta las posibles modificaciones. Por ejemplo, si se hace una modificación en la cache, este hecho debe comunicarse al resto de nodos. En el módulo de direcciones de cada nodo que recibe la notificación de una modificación, se puede determinar si hay alguna cache local que tenga esa línea y, si es así, hace que la misma se invalide. Si la posición de memoria considerada se encuentra en el nodo que recibe la notificación de modificación, el módulo de direcciones necesita mantener un registro que indique que la línea de memoria en cuestión se ha invalidado, y permanece en este estado hasta que sea sustituida. Si otro procesador (local o remoto) solicita la línea invalidada, entonces el módulo de direcciones local debe forzar una escritura que actualice el dato en memoria antes de proporcionar el dato.

## PROS Y CONTRAS DE UN COMPUTADOR NUMA

La principal ventaja de un computador CC-NUMA es que puede proporcionar un grado efectivo de prestaciones con mayores niveles de paralelismo que un SMP, sin que se precisen cambios importantes en el software. Con varios nodos NUMA, el tráfico del bus en cualquier nodo se limita a las peticiones que el bus puede manejar. No obstante, si muchos de los accesos a memoria se producen a nodos remotos, las prestaciones empiezan a reducirse. Existe una razón para creer que esta reducción de prestaciones puede evitarse. En primer lugar, el uso de las caches L1 y L2 permite reducir los accesos a memoria, incluyendo los remotos. Si la mayoría del software tiene una localidad temporal grande, los accesos a memorias remotas no deberían ser muchos. Segundo, si el software tiene una localidad espacial buena y se utiliza memoria virtual, los datos que se necesitan en la aplicación residirán en un número limitado de páginas frecuentemente usadas que pueden cargarse en la memoria local de la aplicación que se está ejecutando. Un informe de los diseñadores de Sequent indica que esta localidad espacial aparece en ciertas aplicaciones representativas [LOVE96]. Finalmente, el esquema de memoria virtual puede mejorarse, incluyendo en el sistema operativo un mecanismo de migración de páginas que mueva una página de memoria virtual al nodo que la utiliza frecuentemente. Un informe de los diseñadores de Silicon Graphics indica que han tenido éxito con esta aproximación [WHIT97].

La alternativa CC-NUMA también tiene desventajas. Dos de ellas se discuten con detalle en [PFIS98]. En primer lugar, un computador CC-NUMA no parece tan transparente como un SMP: se necesitan ciertos cambios en el software para adaptar el sistema operativo y las aplicaciones desde un SMP a un CC-NUMA. Entre los cambios en el sistema operativo es-

tán la ya mencionada asignación de páginas, la asignación de procesos, y el equilibrado de la carga. Un segundo aspecto a considerar es la disponibilidad. Se trata de una cuestión bastante compleja, que depende de la implementación exacta del CC-NUMA. Los lectores interesados pueden consultar [PFIS98].

## 16.6. COMPUTACIÓN VECTORIAL

Aunque las prestaciones de los grandes computadores («mainframes») de propósito general continúan aumentando sin tregua, siguen existiendo aplicaciones que están fuera del alcance de los «mainframes» actuales. Se necesitan computadores que resuelvan problemas matemáticos de procesos reales, tales como los que aparecen en disciplinas como la aerodinámica, la sismología, la meteorología, y la física atómica, nuclear, y de plasmas [WILS84].

Típicamente, estos problemas se caracterizan por necesitar una precisión elevada, y programas que realicen de forma repetitiva operaciones aritméticas en coma flotante con grandes matrices de números. La mayoría de estos problemas pertenecen a la categoría conocida como *simulación de espacios continuos* (continuous-field simulation). En esencia, una situación física (por ejemplo, el flujo de aire próximo a la superficie de un cohete) se puede describir mediante una región de tres dimensiones. Esta región se aproxima mediante una retícula de puntos. Un conjunto de ecuaciones diferenciales definen el comportamiento físico de la superficie en cada punto. Esas ecuaciones se representan como una matriz de valores y coeficientes, y su resolución implica repetidas operaciones aritméticas sobre las matrices de datos.

Para manejar este tipo de problemas, se han desarrollado supercomputadores. Típicamente, estas máquinas son capaces de realizar cientos de millones de operaciones en coma flotante por segundo, y cuestan entre 10 y 15 millones de dólares. A diferencia de los «mainframes», que se diseñan para la multiprogramación y las E/S intensivas, los supercomputadores están optimizados para el tipo de cálculo numérico descrito.

El supercomputador tiene un uso limitado y, debido a su precio, un mercado limitado. Comparativamente, pocas máquinas de este tipo están operativas, en su mayoría en centros de investigación y en algunas agencias gubernamentales con actividades científicas o de ingeniería. Igual que en otras áreas de la tecnología de computadores, hay una constante demanda para mejorar las prestaciones de los supercomputadores. Algunas aplicaciones actuales del campo de la aerodinámica y la física nuclear implican del orden de  $10^{13}$  operaciones, y precisan tiempos de cómputo de más de dos días en los supercomputadores actuales [LEVI82]. En consecuencia, la tecnología y las prestaciones de los supercomputadores continúan evolucionando.

Hay otro tipo de sistemas, que han sido diseñados para las necesidades de la computación vectorial: se trata de los *procesadores matriciales*. Aunque un supercomputador está optimizado para la computación vectorial, es un computador de propósito general, capaz de procesar escalares y realizar tareas generales de procesamiento de datos. Los procesadores matriciales no realizan el procesamiento escalar; están configurados como dispositivos periféricos, para que los usuarios de grandes computadores y minicomputadores puedan ejecutar partes vectorizadas de sus programas.

## APROXIMACIONES A LA COMPUTACIÓN VECTORIAL

La clave para el diseño de un supercomputador o de un procesador matricial, es reconocer que la tarea principal es realizar operaciones sobre matrices o vectores de números en coma

$$\begin{bmatrix} 1,5 \\ 7,1 \\ 6,9 \\ 100,5 \\ 0 \\ 59,7 \end{bmatrix} + \begin{bmatrix} 2,0 \\ 39,7 \\ 1000,003 \\ 11 \\ 21,1 \\ 19,7 \end{bmatrix} = \begin{bmatrix} 3,5 \\ 46,8 \\ 1006,903 \\ 111,5 \\ 21,1 \\ 79,4 \end{bmatrix}$$

$A + B = C$

Figura 16.11. Ejemplo de suma vectorial.

flotante. En un computador de propósito general, esto precisaría realizar una iteración para cada elemento de la matriz. Por ejemplo, considérense dos vectores (matriz unidimensional) de números,  $A$  y  $B$ . Se desea sumarlos y situar el resultado en  $C$ . En el ejemplo de la Figura 16.11, esto precisaría seis sumas. ¿Cómo podríamos acelerar este cálculo? La respuesta está en aprovechar alguna forma de paralelismo.

Se han seguido diversas aproximaciones para lograr paralelismo en la computación vectorial. Lo ilustramos con un ejemplo. Considérese la multiplicación  $C = A \times B$ , donde  $A$ ,  $B$  y  $C$  son matrices de  $N \times N$ . La fórmula para cada elemento de  $C$  es:

$$c_{i,j} = \sum_{k=1}^N a_{i,k} \times b_{k,j}$$

donde los elementos de  $A$ ,  $B$  y  $C$  son  $a_{i,j}$ ,  $b_{i,j}$  y  $c_{i,j}$ , respectivamente. La Figura 16.12a muestra un programa FORTRAN para este cálculo, que puede ejecutarse en un procesador escalar ordinario.

Una posibilidad para mejorar las prestaciones se denomina *procesamiento vectorial*. Éste asume que es posible operar sobre un vector de datos unidimensional. La Figura 16.12b es un programa FORTRAN con una nueva forma de instrucción que permite especificar cálculos vectoriales. La notación ( $J = 1, N$ ) indica que las operaciones con los índices  $J$  del intervalo dado se realizan como una sola operación. La forma de conseguir esto se tratará en breve.

El programa de la Figura 16.12b indica que todos los elementos de la fila  $i$ -ésima se calculan en paralelo. Cada elemento de la fila es una suma, y las sumas (sobre  $K$ ) se hacen en serie, no en paralelo. Incluso así, sólo se necesitan  $N^2$  multiplicaciones vectoriales para este algoritmo, en comparación con las  $N^3$  multiplicaciones escalares del algoritmo escalar.

Otro enfoque, el *procesamiento paralelo*, se ilustra en la Figura 16.12c. Esta aproximación asume que se dispone de  $N$  procesadores independientes que pueden funcionar en paralelo. Para utilizar los procesadores eficazmente, se deben repartir los cálculos de alguna forma entre los procesadores. Se utilizan dos primitivas. La primitiva FORK  $n$  hace que comience un proceso independiente en la posición  $n$ . Entretanto, el proceso original continúa la ejecución en la instrucción que sigue al FORK. Cada ejecución de un FORK genera un nuevo proceso. La instrucción JOIN es esencialmente la inversa al FORK. La sentencia JOIN  $N$  hace que  $N$  procesos independientes se fundan en uno, que continúa la ejecución en la instrucción que sigue al JOIN. El sistema operativo debe coordinar esta fusión, de forma que la ejecución no continúe hasta que los  $N$  procesos hayan alcanzado la instrucción JOIN.

El programa de la Figura 16.12c está escrito imitando el comportamiento del programa de procesamiento vectorial. En el programa de procesamiento paralelo, cada columna de  $C$  es calculada por un proceso distinto. Así, los elementos de una fila de  $C$  se calculan en paralelo.

```

DO 100 I = 1, N
DO 100 J = 1, N
C(I, J) = 0.0
DO 100 K = 1, N
C(I, J) = C(I, J) + A(I, K) * B(K, J)
100 CONTINUE

```

## (a) Procesamiento escalar

```

DO 100 I = 1, N
C(I, J) = 0.0 (J = 1, N)
DO 100 K = 1, N
C(I, J) = C(I, J) + A(I, K) * B(K, J) (J = 1, N)
100 CONTINUE

```

## (b) Procesamiento vectorial

```

DO 50 J = 1, N
FORK 100
50 CONTINUE
J = N
100 DO I = 1, N
C(I, J) = 0.0
DO 200 K = 1, N
C(I, J) = C(I, J) + A(I, K) * B(K, J)
200 CONTINUE
JOIN N

```

## (c) Procesamiento paralelo

Figura 16.12. Multiplicación de matrices ( $C = A \times B$ ).

La discusión precedente describe las aproximaciones a la computación vectorial en términos lógicos o de la arquitectura. Volvamos ahora a considerar las formas de organización de los procesadores que pueden utilizarse para implementar estas aproximaciones. Una amplia variedad de organizaciones han sido y están siendo consideradas. Las tres categorías principales son:

- ALU segmentada
- ALU paralelas
- Procesadores paralelos

La Figura 16.13 muestra las primeras dos aproximaciones. La segmentación de cauce ya se discutió en el Capítulo 11. Aquí se extiende el concepto a la operación de la ALU. Puesto que las operaciones en coma flotante son bastante complejas, existe la posibilidad de descomponer una operación en coma flotante en etapas, de manera que las diferentes etapas puedan operar concurrentemente sobre conjuntos de datos distintos. Esto se ilustra en la Figura 16.14a. La suma en coma flotante se divide en cuatro etapas (véase la Figura 8.22): comparación, desplazamiento, suma y normalización. Un vector de números se introduce secuencialmente a través de la primera etapa. A medida que prosigue el procesamiento, se operarán concurrentemente en el cauce cuatro conjuntos de números diferentes.

Debe quedar claro que esta organización es adecuada para el procesamiento vectorial. Para verlo, considérese el cauce de instrucciones descrito en el Capítulo 11. El procesador ejecuta un ciclo repetitivo de captación y procesamiento de instrucciones. En ausencia de

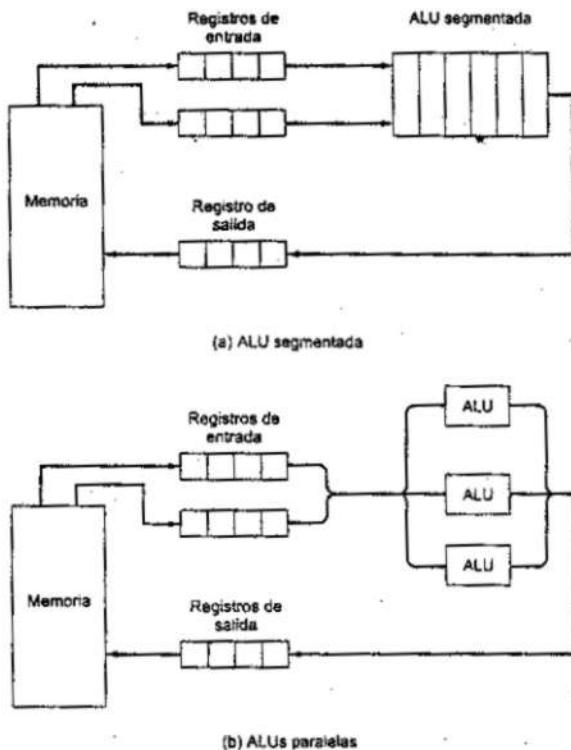


Figura 16.13. Alternativas para la computación vectorial.

saltos, el procesador está continuamente captando instrucciones de posiciones consecutivas. Por tanto, el cauce se mantiene lleno, y se consigue una reducción en el tiempo de ejecución. Igualmente, una ALU segmentada ganará tiempo, sólo si se alimenta con una secuencia de datos de posiciones consecutivas. La ejecución de una única operación en coma flotante aislada no se acelera con un cauce. El aumento de velocidad se consigue cuando se presenta a la ALU un vector de operandos. La unidad de control introduce ciclo a ciclo los datos en la ALU, hasta que se ha procesado el vector completo.

La operación del cauce se puede mejorar si los elementos del vector están disponibles en registros, en lugar de en memoria principal. Este hecho se sugiere en la Figura 16.13a. Los elementos de cada operando vectorial se cargan como un bloque en un registro vectorial, que es simplemente un banco grande de registros idénticos. El resultado también se sitúa en un registro vectorial. Así, la mayoría de las operaciones implican sólo el uso de registros, y sólo las operaciones de carga y almacenamiento, y el comienzo y el final de una operación vectorial, necesitan acceder a memoria.

Los mecanismos que se ilustran en la Figura 16.14 se podrían denominar *segmentación de cauce dentro de una operación*. Es decir, tenemos una única operación aritmética (por ejem-

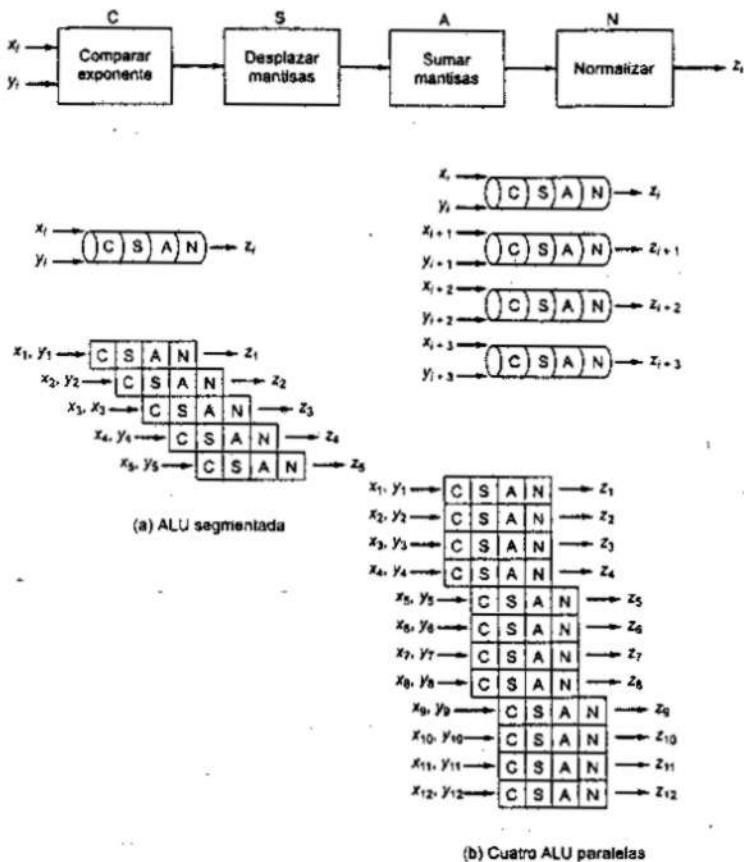


Figura 16.14. Procesamiento segmentado.

pio,  $C = A + B$ ), que se aplica a operandos vectoriales, y la segmentación de cauce permite que múltiples elementos vectoriales se procesen en paralelo. Este mecanismo se puede completar con la *segmentación de cauce de operaciones*. En este caso, hay una secuencia de operaciones aritméticas vectoriales, y se utiliza la segmentación de cauce de instrucciones para acelerar el procesamiento. Una aproximación a esto, conocida como encadenamiento (chaining), aparece en los supercomputadores Cray. La regla básica para el encadenamiento es ésta: una operación vectorial puede empezar tan pronto como el primer elemento del (de los) operando(s) vectorial(es) está disponible, y la unidad funcional (es decir: suma, resta, multiplicación o división) está libre. En esencia, el encadenamiento hace que los resultados generados por una unidad funcional pasen inmediatamente a alimentar otra unidad funcional, y así sucesivamente. Si se utilizan registros vectoriales, los resultados intermedios no tienen que almacenarse en memoria, y pueden utilizarse incluso antes de que la operación vectorial que los origina haya terminado por completo.

Por ejemplo, cuando se calcula  $C = (s \times A) + B$ , donde  $A$ ,  $B$  y  $C$  son vectores, y  $s$  es un escalar, el Cray puede ejecutar tres instrucciones de una vez. Los elementos captados por una instrucción de carga («Load») entran inmediatamente al multiplicador encauzado, los productos se envían al sumador encauzado, y las sumas se sitúan en un registro vectorial tan pronto como el sumador las realiza:

- |                             |                                          |
|-----------------------------|------------------------------------------|
| 1. Carga vectorial          | $A \rightarrow$ Registro vectorial (VR1) |
| 2. Carga vectorial          | $B \rightarrow$ VR2                      |
| 3. Multiplicación vectorial | $s \times VR1 \rightarrow VR3$           |
| 4. Suma vectorial           | $VR3 + VR2 \rightarrow VR4$              |
| 5. Almacenamiento vectorial | $VR4 \rightarrow C$                      |

Las instrucciones 2 y 3 pueden encadenarse (encauzadas), puesto que implican posiciones de memoria y registros distintos. La instrucción 4 necesita los resultados de las instrucciones 2 y 3, pero también puede encadenarse con ellas. Tan pronto como los primeros elementos del registro vectorial 2 y 3 estén disponibles, puede empezar la operación de la instrucción 4.

Otra forma de conseguir el procesamiento vectorial es con el uso de varias ALU en un solo procesador, bajo el control de una única unidad de control. En este caso, la unidad de control enruta los datos hacia las ALU para que puedan funcionar en paralelo. Es también posible utilizar segmentación de cauce en cada una de las ALU paralelas. Esto se ilustra en la Figura 16.14b. El ejemplo muestra un caso en el que cuatro ALU operan en paralelo.

Como la organización de segmentación de cauce, la organización de ALU paralelas se ajusta bien al procesamiento vectorial. La unidad de control introduce los elementos vectoriales en las ALU de forma cíclica, hasta que se han procesado todos los elementos. Esta organización es más compleja que una única ALU de un CPI (ciclo por instrucción).

Finalmente, el procesamiento vectorial puede conseguirse utilizando varios procesadores paralelos. En este caso, es necesario dividir la tarea en múltiples procesos, que se ejecutan en paralelo. Esta organización es efectiva sólo si se dispone de software y hardware para la coordinación efectiva de los procesadores paralelos. Esta es todavía un área de investigación activa, aunque han aparecido algunos productos [GEHR88].

Podemos expandir nuestra taxonomía de la Sección 16.1 para reflejar estas nuevas estructuras, tal y como muestra la Figura 16.15. Las organizaciones de computador pueden distinguirse por la presencia de una o más unidades de control. Múltiples unidades de control implican múltiples procesadores. Siguiendo nuestra discusión previa, si los múltiples procesadores pueden cooperar en la ejecución de una tarea dada, se denominan *procesadores paralelos*.

El lector debe tener cuidado con alguna terminología desafortunada que probablemente encuentre en la bibliografía. El término *procesador vectorial* se utiliza a menudo como equivalente a organización de ALU segmentada, aunque una organización de ALU paralelas



Figura 16.15. Taxonomía de organizaciones de computador.

también está diseñada para el procesamiento vectorial, y, como hemos indicado, una organización de procesadores paralelos también puede diseñarse para el procesamiento vectorial. El término *procesador matricial* se utiliza a veces para referirse a las ALU paralelas, aunque, nuevamente, cualquiera de las tres organizaciones está optimizada para el procesamiento de matrices. Para empeorar las cosas, el término *procesador matricial* usualmente hace referencia a un procesador auxiliar conectado a un procesador de propósito general, y utilizado para realizar cálculos vectoriales. Un procesador matricial puede utilizar tanto la aproximación de ALU segmentada, como la de ALU paralelas.

Actualmente, la organización de ALU segmentada domina el mercado. Los sistemas con segmentación de cauce son menos complejos que otras aproximaciones. El diseño de sus unidades de control y sistemas operativos está lo suficientemente desarrollado para conseguir una asignación de recursos eficiente y elevadas prestaciones. El resto de esta sección se dedica a un examen más detallado de esta aproximación, utilizando un ejemplo específico.

## UNIDAD VECTORIAL IBM 3090

Un buen ejemplo de una organización con ALU segmentada para el procesamiento vectorial, es la unidad vectorial desarrollada para la arquitectura IBM 370, e implementada en la serie 3090 de la gama alta [PADE88, TUCK87]. Esta unidad constituye una extensión opcional al sistema básico, pero está altamente integrada en él. Recuerda a los recursos vectoriales que se encuentran en supercomputadores, tales como los de la familia Cray.

La unidad IBM utiliza una serie de registros vectoriales. Cada registro es, en realidad, un banco de registros escalares. Para calcular la suma  $C = A + B$ , los vectores  $A$  y  $B$  se cargan en dos registros vectoriales. Desde estos registros, los datos pasan tan rápido como es posible a través de la ALU, y los resultados se almacenan en un tercer registro vectorial. El solapeamiento de los cálculos, y la carga en bloque de los datos de entrada en los registros, producen un significativo aumento de la velocidad respecto a la operación en una ALU ordinaria.

### Organización

La arquitectura vectorial de IBM, y de ALU segmentadas vectoriales similares, proporciona mejoras en las prestaciones respecto a los bucles de instrucciones aritméticas escalares, de tres maneras:

- La estructura fija y predeterminada de los datos vectoriales permite reemplazar las instrucciones incluidas en bucles, por rápidas operaciones máquina internas (cableadas o microprogramadas).
- El acceso a los datos y las operaciones aritméticas con elementos vectoriales sucesivos se pueden completar concurrentemente, solapando las operaciones en un cauce segmentado, o realizando en paralelo operaciones sobre múltiples elementos.
- El uso de registros vectoriales para los resultados intermedios evita referencias adicionales a memoria.

La Figura 16.16 muestra la organización general de la unidad vectorial. Aunque la unidad vectorial aparece como un elemento físicamente distinto que se añade al procesador, su arquitectura es una extensión de la arquitectura S/370, y es compatible con ella. La unidad vectorial está integrada en la arquitectura S/370 de la siguiente forma:

- Las instrucciones existentes en la S/370 se utilizan para todas las operaciones escalares.
- Las operaciones aritméticas sobre elementos individuales de un vector, producen exactamente el mismo resultado que las instrucciones escalares del S/370. Por ejemplo, esta

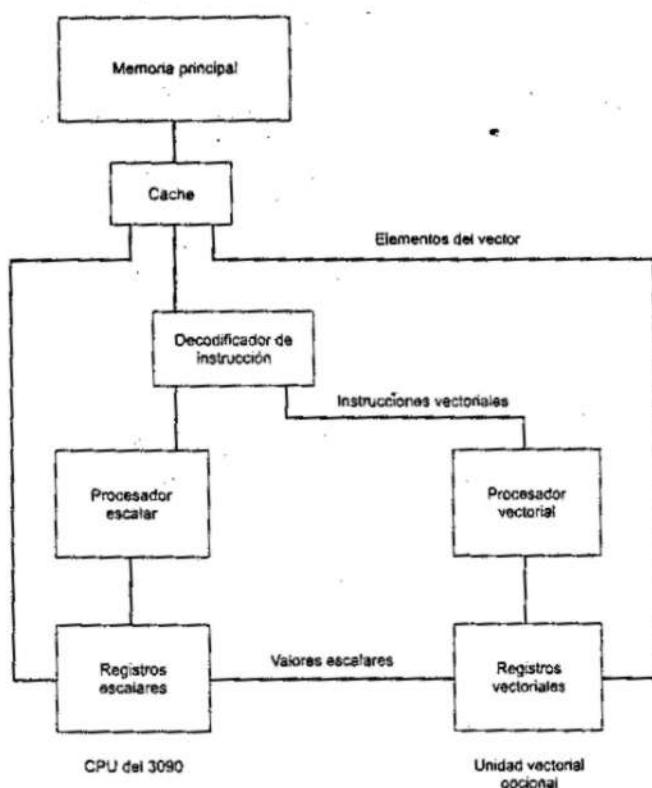


Figura 16.16. IBM 3090 con unidad vectorial.

decisión de diseño concierne a la definición del resultado de una operación DIVIDE en coma flotante. ¿El resultado debe ser exacto, como en la división en coma flotante escalar, o bien se admite una aproximación que permita una implementación de mayor velocidad, pero que en ciertas ocasiones podría producir un error en uno o en más bits de las posiciones menos significativas? La decisión tomada permite una compatibilidad ascendente completa con el S/370, a expensas de una pequeña reducción en las prestaciones.

- Las instrucciones vectoriales se pueden interrumpir, y su ejecución puede continuarse desde el punto de interrupción después de realizar la acción apropiada, de forma compatible con el esquema de interrupción de programas en el S/370.
- Las excepciones aritméticas son las mismas, o bien extensiones de las mismas excepciones propias de las instrucciones aritméticas escalares del S/370, y se utilizan rutinas similares para su tratamiento. Para esto, se emplea un índice de interrupción vectorial, que indica la posición del registro vectorial afectado por la excepción —por ejemplo,

desbordamiento (*overflow*)—. Así, cuando se reanuda la ejecución de la instrucción vectorial, se accede a la posición adecuada del registro vectorial.

- Los datos vectoriales residen en la memoria virtual, manejándose las faltas de página en la forma estándar.

Este nivel de integración proporciona una serie de beneficios. Los sistemas operativos existentes pueden dar cobertura a la unidad vectorial con pequeñas extensiones. Los programas de aplicación, compiladores, y demás software existente, puede ejecutarse sin cambios. El software que pueda beneficiarse de la unidad vectorial puede modificarse según convenga.

## Registros

Un aspecto clave del diseño de la unidad vectorial es si los operandos se sitúan en registros o en memoria. La organización IBM se denomina *registro-a-registro* porque los operandos, tanto los de entrada como los de salida, pueden encontrarse en registros vectoriales. Esta aproximación también se utiliza en el supercomputador Cray. Una alternativa, utilizada en las máquinas de Control Data, consiste en obtener los operandos directamente desde la memoria. La principal desventaja del uso de registros vectoriales es que el programador, o el compilador, deben tenerlo en cuenta. Por ejemplo, supóngase que la longitud de los registros vectoriales es  $K$  y, las longitudes de los vectores procesados es  $N > K$ . En este caso, debe introducirse un bucle vectorial, en el que la operación se realiza sobre  $K$  elementos cada vez y el bucle se repite  $N/K$  veces. La principal ventaja de la aproximación de registros vectoriales es que la operación se desacopla de la memoria, más lenta, y en cambio utiliza principalmente registros.

El incremento de velocidad que se puede conseguir al utilizar registros se muestra en la Figura 16.17 [PADE88]. La rutina FORTRAN multiplica el vector A por el vector B para producir el vector C, donde cada vector tiene una parte real (AR, BR, CR) y una imaginaria (AI, BI, CI). El 3090 puede realizar un ciclo de acceso (lectura o escritura) a memoria principal por ciclo de procesador, o de reloj, tiene registros que permiten mantener dos accesos para lectura y uno para escritura por ciclo, y produce un resultado por ciclo de su unidad aritmética. Asumamos que se utilizan instrucciones que pueden especificar dos operandos fuente y un resultado<sup>4</sup>. Una parte de la figura muestra que, con instrucciones de memoria-a-memoria, cada iteración del cálculo necesita un total de 18 ciclos. Con una arquitectura pura de registro-a-registro (parte b), este tiempo se reduce a 12 ciclos. Por supuesto, con la operación de registro-a-registro, las cantidades vectoriales deben cargarse en los registros vectoriales antes del cálculo, y almacenarse en memoria después. Para vectores largos, esta penalización ocasionada es relativamente pequeña. La Figura 16.17c muestra que la posibilidad de especificar en una instrucción datos, tanto en memoria como en registros vectoriales, reduce el tiempo a 10 ciclos por iteración. Este último tipo de instrucción está incluido en la arquitectura vectorial<sup>5</sup>.

La Figura 16.18 ilustra los registros que forman parte de la unidad vectorial IBM 3090. Hay 16 registros vectoriales de 32 bits. Los registros vectoriales también pueden acoplarse

<sup>4</sup> Para la arquitectura 370, las únicas instrucciones de tres operandos (instrucciones con registro y memoria: RS: Register and Store) especifican dos operandos en registros y uno en memoria. En la parte (a) de este ejemplo, asumimos la existencia de operaciones de tres operandos, en las que todos los operandos están en memoria. Esto se hace con la intención de realizar la comparación y, de hecho, se podría haber elegido ese formato de instrucción para la arquitectura vectorial.

<sup>5</sup> Las instrucciones compuestas, discutidas más adelante, proporcionan una reducción adicional.

## RUTINA FORTRAN

DO 100 J=1, 50

CR (J) = AR (J)\*BR (J) - AI (J)\*BI (J)

100 CI (J) = AR (J)\*BI (J) - AI (J)\*BR (J)

| Operación              | Ciclos    |
|------------------------|-----------|
| AR (J)*BR (J) → T1 (J) | 3         |
| AI (J)*BI (J) → T2 (J) | 3         |
| T1 (J)-T2 (J) → CR (J) | 3         |
| AR (J)*BI (J) → T3 (J) | 3         |
| AI (J)*BR (J) → T4 (J) | 3         |
| T3 (J)+T4 (J) → CI (J) | 3         |
| <b>TOTAL</b>           | <b>18</b> |

(a) Memoria a memoria

| Operación              | Ciclos    |
|------------------------|-----------|
| AR (J) → V1 (J)        | 1         |
| BR (J) → V2 (J)        | 1         |
| V1 (J)*V2 (J) → V3 (J) | 1         |
| AI (J) → V4 (J)        | 1         |
| BI (J) → V5 (J)        | 1         |
| V4 (J)*V5 (J) → V6 (J) | 1         |
| V3 (J)-V6 (J) → V7 (J) | 1         |
| V7 (J) → CR (J)        | 1         |
| V1 (J)*V5 (J) → V8 (J) | 1         |
| V4 (J)*V2 (J) → V9 (J) | 1         |
| V8 (J)+V9 (J) → V0 (J) | 1         |
| V0 (J) → CI (J)        | 1         |
| <b>TOTAL</b>           | <b>12</b> |

(b) Registro a registro

| Operación              | Ciclos    |
|------------------------|-----------|
| AR (J) → V1 (J)        | 1         |
| V1 (J)*BR (J) → V2 (J) | 1         |
| AI (J) → V3 (J)        | 1         |
| V3 (J)*BI (J) → V4 (J) | 1         |
| V2 (J)-V4 (J) → V5 (J) | 1         |
| V5 (J) → CR (J)        | 1         |
| V1 (J)*BI (J) → V6 (J) | 1         |
| V3 (J)*BR (J) → V7 (J) | 1         |
| V6 (J)+V7 (J) → V8 (J) | 1         |
| V8 (J) → CI (J)        | 1         |
| <b>TOTAL</b>           | <b>10</b> |

(c) Memoria a registro

| Operación                     | Ciclos   |
|-------------------------------|----------|
| AR (J) → V1 (J)               | 1        |
| V1 (J)*BR (J) → V2 (J)        | 1        |
| AI (J) → V3 (J)               | 1        |
| V2 (J)-V3 (J)*BI (J) → V2 (J) | 1        |
| V2 (J) → CR (J)               | 1        |
| V1 (J)*BI (J) → V4 (J)        | 1        |
| V4 (J)+V3 (J)*BR (J) → C4 (J) | 1        |
| V4 (J) → CI (J)               | 1        |
| <b>TOTAL</b>                  | <b>8</b> |

(d) Instrucciones compuestas

Figura 16.17. Programas alternativos de computación vectorial.

para constituir 8 registros vectoriales de 64 bits. Cualquier elemento del registro puede almacenar un valor entero o en coma flotante. Así pues, los registros vectoriales pueden utilizarse para valores enteros de 32 y 64 bits, y para valores en coma flotante de 32 y 64 bits.

La arquitectura especifica que cada registro contiene entre 8 y 512 elementos escalares. La selección de la longitud en cada caso significa un compromiso de diseño. El tiempo para realizar una operación vectorial consta esencialmente de la penalización (overhead) debida al tiempo de latencia de inicio del cauce (pipeline startup), y del llenado de los registros, más un ciclo por elemento vectorial. Por tanto, el uso de un número elevado de elementos en cada registro, reduce la importancia relativa del tiempo de inicio en el tiempo total de cómputo. No obstante, esta eficiencia debe contrapesarse con el tiempo extra que se necesita para guardar y restaurar los registros vectoriales al comutar un proceso, y con las limitaciones de costo

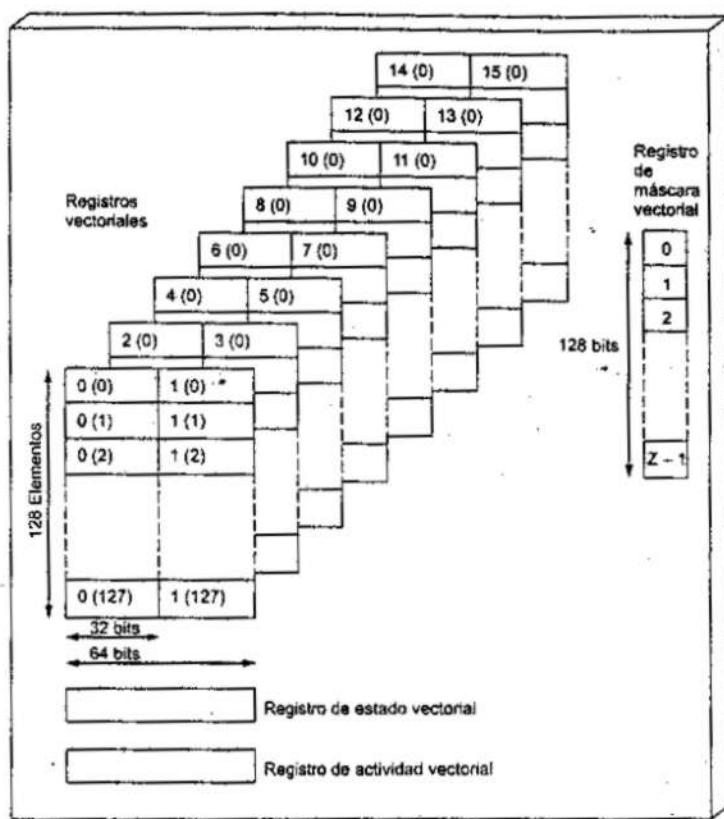


Figura 16.18. Registros de la unidad vectorial del IBM 3090.

y espacio. Estas consideraciones han motivado el uso de 128 elementos por registro en la implementación actual del 3090.

La unidad vectorial necesita tres registros adicionales. El registro de máscara de vector (vector-mask) contiene los bits de máscara que pueden utilizarse para seleccionar qué elementos de un registro vectorial se procesan en una operación concreta. El registro de estado vectorial (vector-status) contiene los campos de control, tales como la cuenta de vector que determina cuantos elementos de los registros vectoriales se van a procesar. La cuenta de actividad vectorial (vector-activity) almacena el tiempo transcurrido ejecutando instrucciones vectoriales.

#### Instrucciones compuestas

Como se discutió arriba, para mejorar las prestaciones se puede solapar la ejecución de varias instrucciones, utilizando encadenamiento. Los diseñadores de la unidad vectorial IBM

prefirieron no incluir esta capacidad por varias razones. La arquitectura del S/370 tendría que extenderse para manejar interrupciones complejas (incluyendo sus efectos en la gestión de la memoria virtual), y se necesitarían los correspondientes cambios en el software. Una cuestión más importante fue el costo de incluir las señales de control adicionales y los caminos de acceso a los registros en la unidad vectorial, para conseguir el encadenamiento generalizado.

En su lugar, se incluyeron tres operaciones que combinan en una instrucción (un sólo código de operación) las secuencias más comunes en los cálculos vectoriales, concretamente la multiplicación seguida por una suma, resta, o acumulación. La instrucción MULTIPLY-AND-ADD de memoria-a-registro, por ejemplo, capta un vector de la memoria, lo multiplica por un vector almacenado en un registro, y suma el producto a un tercer vector en otro registro. Utilizando las instrucciones compuestas, MULTIPLY-AND-ADD y MULTIPLY-AND-SUBTRACT, en el ejemplo de la Figura 16.17, el tiempo total de una iteración se reduce de 10 a 8 ciclos.

A diferencia del encadenamiento, las instrucciones compuestas no precisan utilizar registros adicionales para el almacenamiento temporal de los resultados intermedios, y requieren un acceso a registro menos. Por ejemplo, considere la siguiente cadena:

$$\begin{aligned} A &\rightarrow VR1 \\ VR1 + VR2 &\rightarrow VR1 \end{aligned}$$

En este caso, se requieren dos almacenamientos en el registro vectorial VR1. En la arquitectura del IBM hay una instrucción ADD de memoria-a-registro. Con esta instrucción, sólo la suma final se sitúa en VR1. La instrucción compuesta evita además la necesidad de reflejar en la descripción del estado de la máquina la ejecución concurrente de varias instrucciones, con lo que se simplifica el almacenamiento y la recuperación del estado por parte del sistema operativo y la gestión de interrupciones.

### El repertorio de instrucciones

La Tabla 16.3 presenta las operaciones lógicas y aritméticas definidas para la arquitectura vectorial. Además, hay instrucciones de carga de memoria-a-registro y de almacenamiento de registro-a-memoria. Obsérvese que muchas instrucciones utilizan un formato de tres operandos. Además, muchas instrucciones poseen un cierto número de variaciones, según la situación de los operandos. Un operando fuente puede ser un registro vectorial (V), un registro escalar (Q), o estar en memoria (S; storage). El destino es siempre un registro vectorial, excepto en la comparación, cuyo resultado va al registro de máscara vectorial (vector-mask). Con todas estas variantes, el número total de códigos de operación (instrucciones distintas) es 171. Este considerable número de instrucciones, sin embargo, no es tan costoso de implementar como podría imaginarse. Las unidades funcionales y los caminos de datos para proporcionar operandos a los cauces vectoriales desde la memoria, los registros vectoriales, y los registros vectoriales, son los principales responsables del costo del hardware. La arquitectura puede, con pocas diferencias en el costo, proporcionar un conjunto amplio de variantes en el uso de estos registros y cauces.

La mayoría de las instrucciones de la Tabla 16.3 son autoexplicativas. Las dos instrucciones de acumulación requieren una explicación adicional. La operación de acumulación suma todos los elementos de un vector (ACCUMULATE), o los elementos del producto de dos vectores (MULTIPLY-AND-ACCUMULATE). Estas instrucciones plantean un problema de diseño interesante. Nos gustaría realizar esta operación tan rápidamente como sea posible,

Tabla 16.3. Unidad vectorial del IBM 3090: instrucciones lógicas y aritméticas

| Operación               | Tipos de datos |       |                  | Posiciones de los operandos |               |               |               |
|-------------------------|----------------|-------|------------------|-----------------------------|---------------|---------------|---------------|
|                         | Punto flotante |       | Binario o lógico |                             |               |               |               |
|                         | Largo          | Corto |                  | V + V - V                   | V + S - V     | Q + V - V     | Q + S - V     |
| Add                     | FL             | FS    | BI               | V + V - V                   | V + S - V     | Q + V - V     | Q + S - V     |
| Subtract                | FL             | FS    | BI               | V - V - V                   | V - S - V     | Q - V - V     | Q - S - V     |
| Multiply                | FL             | FS    | BI               | V × V - V                   | V × V - V     | Q × V - V     | Q × S - V     |
| Divide                  | FL             | FS    | —                | V/V - V                     | V/S - V       | Q/V - V       | Q/S - V       |
| Compare                 | FL             | FS    | BI               | V - V - V                   | V - S - V     | Q - V - V     | Q - S - V     |
| Multiply and Add        | FL             | FS    | —                |                             | V + V - S - V | V + Q × V - V | V + Q · S - V |
| Multiply and Subtract   | FL             | FS    | —                |                             | V - V × S - V | V - Q × V - V | V - Q × S - V |
| Multiply and Accumulate | FL             | FS    | —                | P + V - V                   | P + S - V     |               |               |
| Complement              | FL             | FS    | BI               | -V - V                      |               |               |               |
| Positive Absolute       | FL             | FS    | BI               | V  - V                      |               |               |               |
| Negative Absolute       | FL             | FS    | BI               | - V  - V                    |               |               |               |
| Maximum                 | FL             | FS    | —                |                             |               | Q · V → Q     |               |
| Maximum Absolute        | FL             | FS    | —                |                             |               | Q · V → Q     |               |
| Minimum                 | FL             | FS    | —                |                             |               | Q · V → Q     |               |
| Shift Left Logical      | —              | —     | LO               | -V - V                      |               |               |               |
| Shift Right Logical     | —              | —     | LO               | -V - V                      |               |               |               |
| And                     | —              | —     | LO               | V & V - V                   | V & S - V     | Q & V - V     | Q & S - V     |
| Or                      | —              | —     | LO               | V   V - V                   | V   S - V     | - Q   V - V   | Q   S - V     |
| Exclusive-Or            | —              | —     | LO               | V ≠ V - V                   | V ≠ S - V     | Q ≠ V - V     | Q ≠ S - V     |

**Explicación:** Tipos de datos

FL punto-flotante largo

FS punto-flotante corto

BI entero binario.

LO lógico

**Posiciones de los operandos**

V registro vectorial

S Memoria

Q Escalar (general o registro de punto-flotante)

Operación especial

aprovechando las ventajas del cauce segmentado de la ALU. La dificultad está en que la suma de dos números que se introducen en el cauce no está disponible hasta varios ciclos más tarde. Como consecuencia, el tercer elemento del vector no se podría añadir a la suma de los dos primeros hasta que estos no hayan pasado a través de todo el cauce. Para superar este problema, los elementos del vector se suman de forma que produzcan cuatro sumas parciales. Concretamente, con los elementos 0, 4, 8, 12, ..., 124, se obtiene la suma parcial 0; con los elementos 1, 5, 9, 13, ..., 125, la suma parcial 1; con los elementos 2, 6, 10, 14, ..., 126, la suma parcial 2; y con los elementos 3, 7, 11, 15, ..., 127, la suma parcial 3. Cada una de estas sumas parciales puede realizarse en el cauce a la máxima velocidad, puesto que el retardo del cauce es de cuatro ciclos. Un registro vectorial distinto se utiliza para almacenar las sumas parciales. Cuando se han procesado todos los elementos del vector original, se suman las cuatro sumas parciales para obtener el resultado final. El tiempo de esta segunda fase no es crítico, puesto que sólo están implicados cuatro elementos.

**16.7. LECTURAS RECOMENDADAS**

[CATA94] revisa los principios de los multiprocesadores, y examina con detalle los multiprocesadores basados en el SPARC. Los SMP también se tratan en [STON93] y [HWAN93]. [PFIS98] es una lectura esencial para cualquiera que esté interesado en los «clusters»; el li-

bro cubre los aspectos de diseño de hardware y software, y compara los «clusters» con los SMP y los NUMA; el libro contiene también una sólida descripción técnica de las cuestiones de diseño de los SMP y los NUMA.

Una excelente revisión de los aspectos relativos a la coherencia de cache en multiprocesadores, se encuentra en [LILJ93]. [TOMA93] contiene reimpresiones de algunos de los artículos clave sobre el tema.

En [STON93] y [HWAN93] pueden encontrarse buenas discusiones de la computación vectorial.

CATA94 Catanzaro, B. *Multiprocessor System Architectures*. Mountain View, CA: Sunsoft Press, 1994.

HWAN93 Hwang, K. *Advanced Computer Architecture*. New York: McGraw-Hill, 1993.

LILJ93 Lilja, D. «Cache Coherence in Large-Scale Shared-Memory Multiprocessors: Issues and Comparisons.» *ACM Computing Surveys*, September, 1993.

PFIS98 Pfister, G. *In Search of Clusters*. Upper Saddle River, NJ: Prentice Hall, 1998.

STON93 Stone, H. *High-Performance Computer Architecture*. Reading, MA: Addison-Wesley, 1993.

TOMA93 Tomasevic, M., y Milutinovic, V. *The Cache Coherence Problem in Shared-Memory Multiprocessors: Hardware Solutions*. Los Alamitos, CA: IEEE Computer Society Press, 1993.

## 28. PROBLEMAS

- 16.1. Sea  $\alpha$  el porcentaje de código de programa que puede ejecutarse simultáneamente por los  $n$  procesadores de un computador. Asuma que el resto del código debe ejecutarse secuencialmente por un sólo procesador. Cada procesador tiene una velocidad de ejecución de  $x$  MIPS.
  - a) Proporcione una expresión para los MIPS efectivos en función de  $n$ ,  $\alpha$  y  $x$ , cuando se utiliza este sistema para ejecutar exclusivamente este programa.
  - b) Si  $n = 16$ , y  $x = 4$  MIPS, determine el valor de  $\alpha$  que hace que las prestaciones del sistema sean iguales a 40 MIPS.
- 16.2. Un multiprocesador con ocho procesadores tiene conectadas 20 unidades de cinta. Hay un gran número de trabajos enviados al sistema, y cada uno de ellos necesita un máximo de cuatro unidades de cinta para completar su ejecución. Asuma que cada trabajo comienza a ejecutarse utilizando tres unidades de cinta durante un período largo, antes de que necesite la cuarta unidad de cinta durante un corto período de tiempo antes de finalizar su ejecución. Asuma también una fuente continua que suministra trabajos.
  - a) Suponga que el planificador del sistema operativo no iniciará ningún trabajo, a no ser que existan cuatro unidades de cinta disponibles. Cuando un trabajo comienza, inmediatamente se le asignan cuatro unidades de cinta, y no se liberan hasta que el trabajo finalice. ¿Cuál es el número máximo de trabajos que pueden estar ejecutándose al mismo tiempo? ¿Cuál es el número máximo y mínimo de unidades de cinta que pueden estar inactivas como resultado de esta política?

- b) Sugiera una política alternativa que mejore la utilización de las unidades de cinta al mismo tiempo que se evita el bloqueo (deadlock) del sistema. ¿Cuál es el número máximo de trabajos que pueden estar ejecutándose al mismo tiempo? ¿Cuáles son los límites en el número de unidades de cinta inactivas?
- 16.3. ¿Puede existir algún problema con la aproximación de cache «write-once» en los multiprocesadores basados en bus? Si es así, sugiera una solución.
- 16.4. Considere que dos procesadores de un SMP necesitan acceder a la misma línea de datos de memoria principal. Ambos procesadores tienen cache y utilizan el protocolo MESI. Inicialmente, ambas caches tienen una copia no válida de la línea. La Figura 16.9 muestra el resultado de la lectura de la línea x por parte del procesador P1. Si éste es el inicio de una secuencia de accesos, dibuje las figuras correspondientes a la siguiente secuencia:
1. P2 lee x.
  2. P1 escribe en x (por claridad, marque con x' la línea en la cache de P1).
  3. P1 escribe en x (marque con x'' la línea de cache en P1).
  4. P2 lee x.
- 16.5. La Figura 16.20 muestra dos diagramas de estados posibles como protocolos de coherencia de cache. Deduzca y explique cada protocolo y compárelo con el protocolo MESI.
- 16.6. Considere un SMP con caches L1 y L2 que utilizan el protocolo MESI. Como se explica en la Sección 16.3, cada línea de la cache L2 puede estar en uno de los cuatro estados del protocolo. ¿Se necesitan todos los estados en cada línea de la cache L1? Si es así, ¿por qué? Si no, explique qué estado o estados se pueden eliminar.

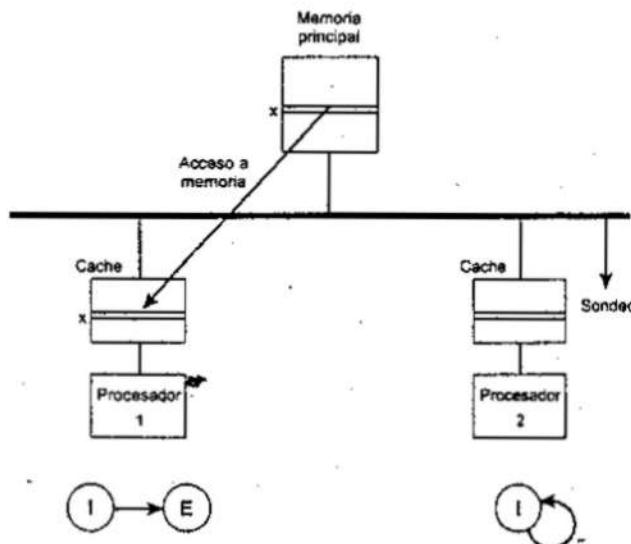


Figura 16.19. Ejemplo MESI: el procesador 1 lee la línea x.

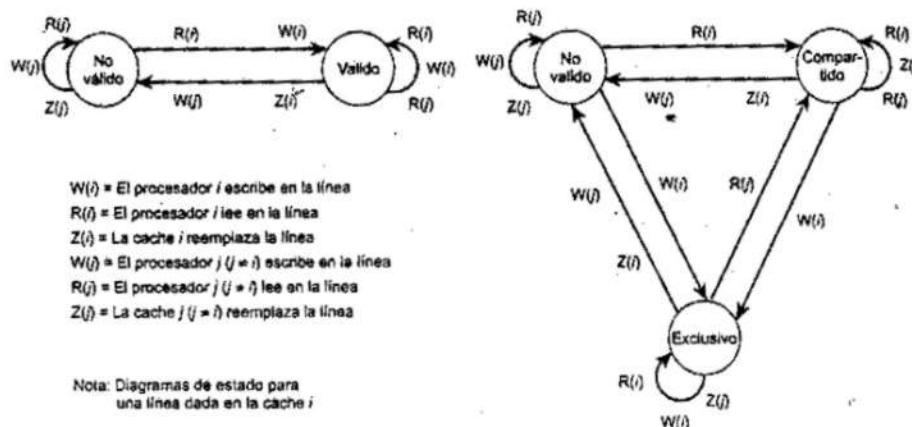


Figura 16.20. Protocolos de coherencia para dos cachés.

- 16.7. La Tabla 16.1 proporciona las prestaciones de una configuración de tres niveles de caché para el IBM S/390. El propósito de este problema es determinar si vale la pena incluir el tercer nivel de cache. Determine la penalización de acceso (número medio de ciclos del procesador, PU) para un sistema con una sola cache L1, y normalice ese valor a 1.0. Después determine la penalización de acceso cuando se utilizan dos niveles de cache, L1 y L2, y la penalización de acceso cuando se utilizan las tres caches. Obtenga la magnitud de la mejora en cada caso, e indique su opinión respecto al valor de la cache L3.
- 16.8. El siguiente segmento de código debe ejecutarse 64 veces para evaluar la expresión aritmética:  $D(I) = A(I) + B(I) \times C(I)$  para  $0 \leq I \leq 63$ .

|                        |                                  |
|------------------------|----------------------------------|
| Load R1, B( <i>I</i> ) | /R1 ← Memoria ( $\alpha + I$ ) / |
| Load R2, C( <i>I</i> ) | /R2 ← Memoria ( $\beta + I$ ) /  |
| Mult R1, R2            | /R1 ← (R1) × (R2) /              |
| Load R3, A( <i>I</i> ) | /R3 ← Memoria ( $\gamma + I$ ) / |
| Add R3, R1             | /R3 ← (R3) + (R1) /              |
| Load D1, R3            | /Memoria 0 ← (R3) /              |

R1, R2 y R3 son registros del procesador, y  $\alpha$ ,  $\beta$ ,  $\gamma$ , y  $\theta$  son las direcciones de memoria principal de comienzo de los vectores B(*I*), C(*I*), A(*I*) y D(*I*), respectivamente. Asuma cuatro ciclos de reloj para una operación de carga (Load) o almacenamiento (Store), dos ciclos para la suma (Add), y ocho para la multiplicación (Mult), tanto en el procesador de un computador monocomputador como en el de una máquina SIMD.

- a) Calcule el número total de ciclos de reloj que se necesitan para ejecutar este segmento de código 64 veces en un computador monoprocesador, SISD, ignorando todos los otros retardos.

- b) Considere que se utiliza un SIMD con 64 elementos de proceso para ejecutar operaciones vectoriales, utilizando seis instrucciones vectoriales sincronizadas por el mismo reloj, y que actúan sobre vectores de datos de 64 componentes. Calcule el tiempo de ejecución total en la máquina SIMD, ignorando la instrucción de difusión (broadcast) y los demás retardos.
- c) ¿Cuál es la ganancia de velocidad (speedup) del computador SIMD con respecto al SISD?

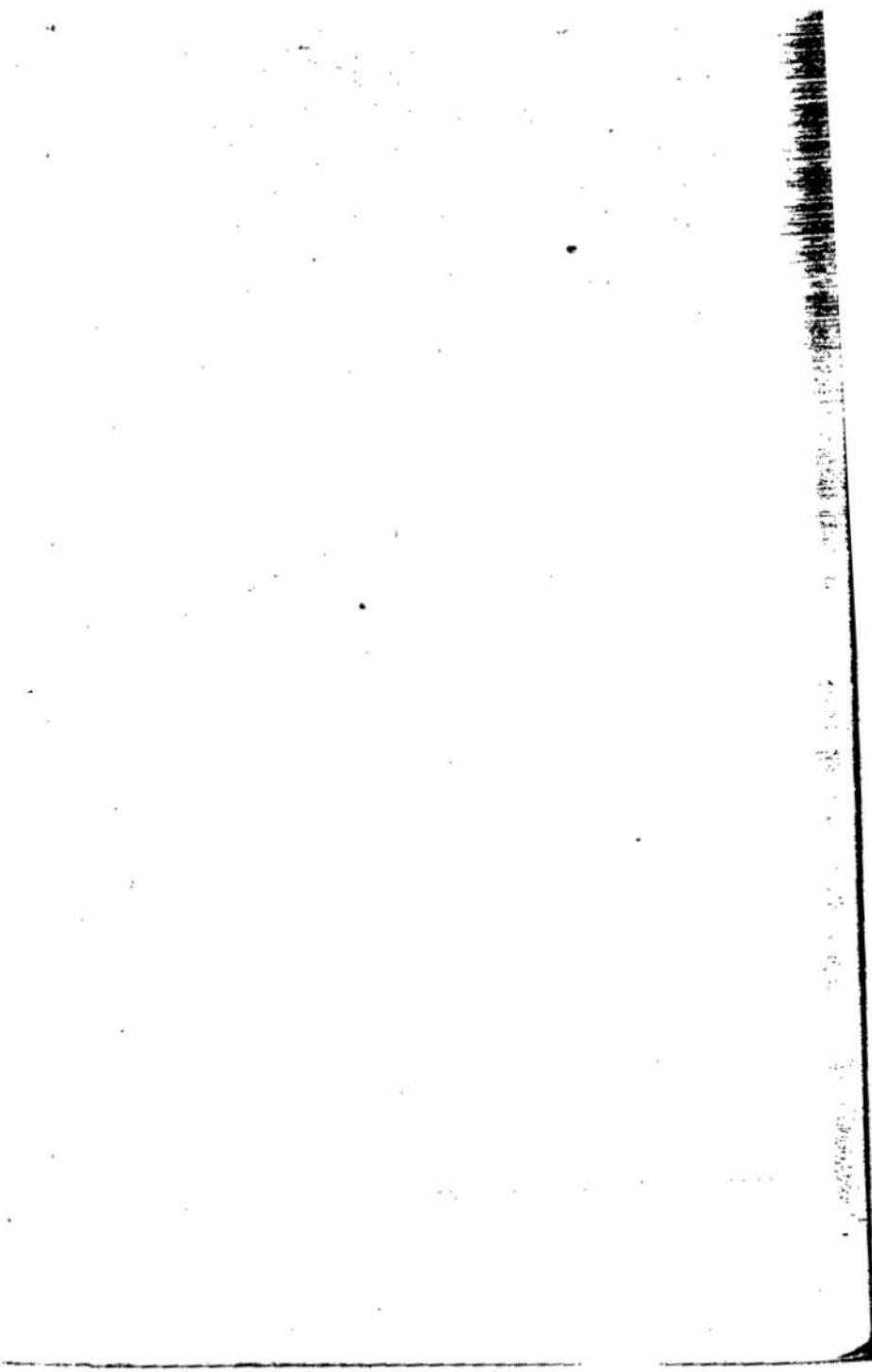
16.9. Obtenga la versión vectorizada del siguiente programa

```

DO 20 I = 1, N
B(I, 1) = 0
DO 10 J = 1, M
A(I) = A(I) + B(I, J) * C(I, J)
10 CONTINUE
D(I) = (E(I) + A(I))
20 CONTINUE

```

- 16.10. Un computador monoprocesador puede operar tanto en modo escalar como en modo vectorial, realizando los cálculos nueve veces más rápido en el modo vectorial. Un programa de prueba (benchmark) consume un tiempo  $T$  en este computador. De este tiempo, un 25 % correspondió al modo vectorial, y el resto del tiempo al modo escalar.
- a) Calcule la ganancia de velocidad efectiva en las condiciones mencionadas, con respecto a no utilizar el modo vectorial. Calcule también  $x$ , porcentaje de código que ha sido vectorizado (compilado para utilizar el modo vectorial) en el programa indicado.
- b) Suponga que se dobla la razón de velocidades entre el modo vectorial y el escalar mediante mejoras en el hardware. Calcule la ganancia de velocidad efectiva que se consigue.
- c) Considere que la misma ganancia de velocidad obtenida en (b) se obtiene a partir de mejoras en el compilador, en lugar de a través de cambios en el hardware. ¿Cuál debería ser ahora el nuevo porcentaje de vectorización ( $x$ ) que debería proporcionar el compilador para el mismo programa de prueba?



## **APÉNDICE A**

# **Lógica digital**

### **A.1. Álgebra de Boole**

### **A.2. Puertas**

### **A.3. Circuitos combinacionales**

Implementación de las funciones booleanas

Multiplexores

Decodificadores

Array lógico programable (PLA, Programmable Logic Array)

Memorias de sólo lectura (ROM, Read Only Memory)

Sumadores

### **A.4. Circuitos secuenciales**

Biestables

Registros

Contadores

### **A.5. Problemas**

**E**l funcionamiento de los computadores digitales se basa en la memorización y procesamiento de datos binarios. A lo largo de este libro, hemos supuesto la existencia de elementos de memoria que pueden estar en uno de dos estados estables, y de circuitos que pueden operar con datos binarios bajo la acción de señales de control, para implementar distintas funciones. En este apéndice, sugerimos cómo se pueden implementar estos elementos de memoria y circuitos en lógica digital, concretamente con circuitos combinacionales y secuenciales. El apéndice comienza con un breve repaso del álgebra de Boole, que es el fundamento matemático de la lógica digital. Luego presentaremos el concepto de puerta. Finalmente, se describen los circuitos combinacionales y secuenciales, que se construyen con puertas.

### APÉNDICE ALGEBRA DE BOOLE

La circuitería digital en computadores digitales y otros sistemas digitales, se diseña y se analiza con el uso de una disciplina matemática denominada *álgebra de Boole*. El nombre es en honor al matemático inglés George Boole, que propuso los principios básicos de este álgebra en 1854 en su tratado *An investigation of the laws of thought on which to found the mathematical theories of logic and probabilities*. En 1938, Claude Shannon, un investigador asistente en el Departamento de Ingeniería Eléctrica del M.I.T., sugirió que el álgebra de Boole podría usarse para resolver problemas de diseño de circuitos de conmutación [SHAN38]. Las técnicas de Shannon se usaron, consecuentemente, en el análisis y diseño de circuitos electrónicos digitales. El álgebra de Boole resulta ser una herramienta útil en dos áreas:

- Análisis: es una forma concisa de describir el funcionamiento de los circuitos digitales.
- Diseño: dada una función deseada, se puede aplicar el álgebra de Boole para desarrollar una implementación de complejidad simplificada de esta función.

Como con cualquier álgebra, el álgebra de Boole usa variables y operaciones. En este caso, las variables y las operaciones son lógicas. Por tanto, una variable puede tomar el valor 1 (VERDADERO), o 0 (FALSO). Las operaciones lógicas básicas son AND, OR y NOT, que se representan simbólicamente con los signos: punto, más y rayado superior.

$$\begin{aligned} A \text{ AND } B &= A \cdot B \\ A \text{ OR } B &= A + B \\ \text{NOT } A &= \bar{A} \end{aligned}$$

La operación AND es verdadera (valor binario 1) si, y sólo si, los dos operandos son verdaderos. El resultado de la operación OR es verdad si, y sólo si, uno o ambos operandos son verdad. La operación unitaria NOT invierte el valor del operando. Por ejemplo, consideremos la ecuación:

$$D = A + (\bar{B} + C)$$

D es igual a 1 si A es 1, o si B = 0 y C = 1. En otro caso, D es igual a 0.

Se necesitan varias aclaraciones en relación con la notación. En ausencia de paréntesis, la operación AND es preferente a la operación OR. Además, cuando no hay ambigüedad, la operación AND se representa con una simple concatenación, en lugar de con el operador punto. Por tanto:

$$A + B \cdot C = A + (B \cdot C) = A + BC$$

lo que quiere decir: hacer AND con B y C, luego hacer la OR con el resultado y A.

Tabla A.1. Operaciones booleanas

| P | Q | NOT P | P AND Q | P OR Q | P XOR Q | P NAND Q | P NOR Q |
|---|---|-------|---------|--------|---------|----------|---------|
| 0 | 0 | 1     | 0       | 0      | 0       | 1        | 1       |
| 0 | 1 | 1     | 0       | 1      | 1       | 1        | 0       |
| 1 | 0 | 0     | 0       | 1      | 1       | 1        | 0       |
| 1 | 1 | 0     | 1       | 1      | 0       | 0        | 0       |

La Tabla A.1 define las operaciones lógicas básicas en una forma conocida como *tabla verdad*, que simplemente enumera el valor de una operación para cada combinación posible de los valores de los operandos. La tabla también enumera otros tres operadores útiles: XOR, NAND y NOR. La exclusive-or (XOR) de dos operandos lógicos es 1 si, y sólo si, sólo uno de los operandos vale 1. La función NAND es el complemento (NOT) de la función AND, y la NOR es el complemento de la OR:

$$A \text{ NAND } B = \text{NOT}(A \text{ AND } B) = \overline{AB}$$

$$A \text{ NOR } B = \text{NOT}(A \text{ OR } B) = \overline{A + B}$$

Como veremos, estas tres operaciones nuevas pueden ser útiles para implementar ciertos circuitos digitales.

La Tabla A.2 resume las identidades clave del álgebra de Boole. Las ecuaciones se han organizado en dos columnas para mostrar la complementariedad, o dualidad, propia de las operaciones AND y OR. Hay dos clases de identidades: las reglas básicas (*o postulados*) que se afirman sin demostración, y otras identidades que se pueden derivar de los postulados básicos. Los postulados definen la manera en la que expresiones booleanas se interpretan. Una de las dos leyes distributivas merece ser destacada ya que difiere de lo que encontraremos en un álgebra normal:

$$A + (B \cdot C) = (A + B) \cdot (A + C)$$

Las dos últimas expresiones, se denominan «teorema de DeMorgan». Se pueden reescribir de la siguiente forma:

$$A \text{ NOR } B = \overline{A} \text{ AND } \overline{B}$$

$$A \text{ NAND } B = \overline{A} \text{ OR } \overline{B}$$

Se invita al lector a verificar las expresiones de la Tabla A.2, sustituyendo las variables A, B y C, por valores reales (unos y ceros).

Tabla A.2. Identidades básicas del álgebra de Boole

| Postulados básicos                                   |                                                      |                         |
|------------------------------------------------------|------------------------------------------------------|-------------------------|
| $A \cdot B = B \cdot A$                              | $A + B = B + A$                                      | Ley comutativa          |
| $A \cdot (B + C) = (A \cdot B) + (A \cdot C)$        | $\cancel{A} + (B \cdot C) = (A + B) \cdot (A + C)$   | Ley distributiva        |
| $1 \cdot A = A$                                      | $0 + A = A$                                          | Elemento neutro         |
| $A \cdot \overline{A} = 0$                           | $A + \overline{A} = 1$                               | Elemento complementario |
| Otras identidades                                    |                                                      |                         |
| $0 \cdot A = 0$                                      | $1 + A = 1$                                          |                         |
| $A \cdot A = A$                                      | $A + A = A$                                          |                         |
| $A \cdot (B \cdot C) = (A \cdot B) \cdot C$          | $A + (B + C) = (A + B) + C$                          | Ley asociativa          |
| $\overline{A \cdot B} = \overline{A} + \overline{B}$ | $\overline{A + B} = \overline{A} \cdot \overline{B}$ | Teorema DeMorgan        |

**A.2. PUERTAS**

El bloque fundamental de construcción de todos los circuitos lógicos digitales son las puertas. Las funciones lógicas se implementan interconectando puertas.

Una puerta es un circuito electrónico que produce como señal de salida una operación booleana sencilla de las señales de entrada. Las puertas básicas usadas en lógica digital son AND, OR, NOT, NAND y NOR. La Figura A.1 muestra estas cinco puertas. Cada puerta se define de tres formas: símbolo gráfico, notación algebraica y tabla verdad. La simbología usada aquí y a lo largo del apéndice es el estándar IEEE, IEEE Std 91 [IEEE84]. Hay que destacar que la operación de inversión (NOT) se denota por un círculo.

Cada puerta tiene una o dos entradas y una salida. Cuando los valores de entrada cambian, la señal de salida correcta aparece casi instantáneamente, retrasada sólo por el tiempo de propagación de la señal a través de la puerta (conocido como *retardo de puerta*). El significado de esto se verá en la Sección A.3.

Además de las puertas indicadas en la Figura A.1, se pueden usar puertas con 3, 4 o más entradas. Por tanto, se puede implementar  $X + Y + Z$  con una simple puerta OR de tres entradas.

| Nombre | Símbolo | Función algebraica                      | Tabla verdad                                                                                                                                                                                                                                                           |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|--------|---------|-----------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AND    |         | $F = A \cdot B$<br>$\delta$<br>$F = AB$ | <table border="1"> <thead> <tr> <th>A</th><th>B</th><th>F</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </tbody> </table> | A | B | F | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| A      | B       | F                                       |                                                                                                                                                                                                                                                                        |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 0      | 0       | 0                                       |                                                                                                                                                                                                                                                                        |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 0      | 1       | 0                                       |                                                                                                                                                                                                                                                                        |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 1      | 0       | 0                                       |                                                                                                                                                                                                                                                                        |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 1      | 1       | 1                                       |                                                                                                                                                                                                                                                                        |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| OR     |         | $F = A + B$                             | <table border="1"> <thead> <tr> <th>A</th><th>B</th><th>F</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </tbody> </table> | A | B | F | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| A      | B       | F                                       |                                                                                                                                                                                                                                                                        |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 0      | 0       | 0                                       |                                                                                                                                                                                                                                                                        |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 0      | 1       | 1                                       |                                                                                                                                                                                                                                                                        |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 1      | 0       | 1                                       |                                                                                                                                                                                                                                                                        |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 1      | 1       | 1                                       |                                                                                                                                                                                                                                                                        |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| NOT    |         | $F = \bar{A}$<br>$\delta$<br>$F = A'$   | <table border="1"> <thead> <tr> <th>A</th><th>F</th></tr> </thead> <tbody> <tr><td>0</td><td>1</td></tr> <tr><td>1</td><td>0</td></tr> </tbody> </table>                                                                                                               | A | F | 0 | 1 | 1 | 0 |   |   |   |   |   |   |   |   |   |
| A      | F       |                                         |                                                                                                                                                                                                                                                                        |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 0      | 1       |                                         |                                                                                                                                                                                                                                                                        |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 1      | 0       |                                         |                                                                                                                                                                                                                                                                        |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| NAND   |         | $F = (\overline{AB})$                   | <table border="1"> <thead> <tr> <th>A</th><th>B</th><th>F</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>0</td></tr> </tbody> </table> | A | B | F | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |
| A      | B       | F                                       |                                                                                                                                                                                                                                                                        |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 0      | 0       | 1                                       |                                                                                                                                                                                                                                                                        |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 0      | 1       | 1                                       |                                                                                                                                                                                                                                                                        |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 1      | 0       | 1                                       |                                                                                                                                                                                                                                                                        |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 1      | 1       | 0                                       |                                                                                                                                                                                                                                                                        |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| NOR    |         | $F = (\overline{A} + \overline{B})$     | <table border="1"> <thead> <tr> <th>A</th><th>B</th><th>F</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>0</td></tr> </tbody> </table> | A | B | F | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 |
| A      | B       | F                                       |                                                                                                                                                                                                                                                                        |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 0      | 0       | 1                                       |                                                                                                                                                                                                                                                                        |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 0      | 1       | 0                                       |                                                                                                                                                                                                                                                                        |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 1      | 0       | 0                                       |                                                                                                                                                                                                                                                                        |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 1      | 1       | 0                                       |                                                                                                                                                                                                                                                                        |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |

Figura A.1. Puertas lógicas básicas.

Normalmente, no se usan todos los tipos de puertas en implementación. El diseño y la fabricación pueden ser más sencillos si sólo se usan uno o dos tipos de puertas. Por tanto, es importante identificar conjuntos de puertas *funcionalmente completos*. Esto significa que cualquier función booleana se puede implementar usando sólo las puertas del conjunto. Los siguientes conjuntos son funcionalmente completos:

- AND, OR, NOT
- AND, NOT
- OR, NOT
- NAND
- NOR

Debería quedar claro que las puertas AND, OR y NOT constituyen un conjunto funcionalmente completo, ya que representan tres operaciones del álgebra de Boole. Para que las puertas AND y NOT formen un conjunto completo, debe haber una forma de sintetizar la operación OR a partir de las operaciones AND y NOT. Esto se puede hacer aplicando el teorema de DeMorgan:

$$A + B = \overline{\overline{A} \cdot \overline{B}}$$

$$A \text{ OR } B = \text{NOT}(\text{NOT } A \text{ AND } \text{NOT } B)$$

De igual forma, las operaciones OR y NOT son funcionalmente completas, porque se pueden usar para sintetizar la operación AND.

La Figura A.2 muestra cómo se pueden implementar las funciones AND, OR y NOT únicamente con puertas NAND, y la Figura A.3 muestra lo mismo para NOR. Por esta razón, se pueden implementar circuitos digitales únicamente con puertas NAND o NOR, como frecuentemente se hace.

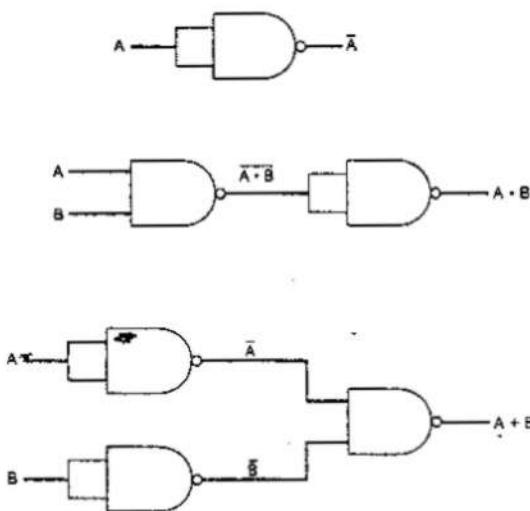


Figura A.2. Uso de las puertas NAND.

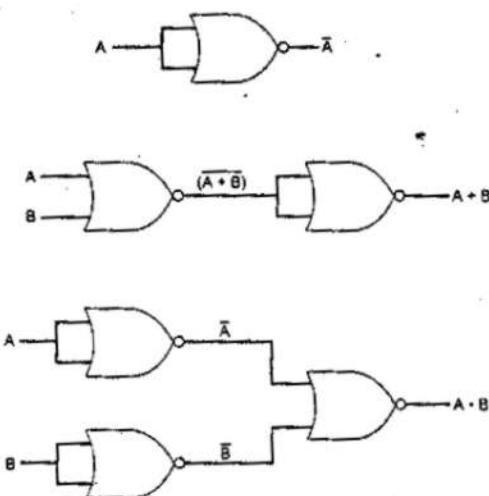


Figura A.3. Uso de las puertas NOR.

Con las puertas, se alcanza el nivel más primitivo de la ciencia e ingeniería de computadores. Un examen de las combinaciones de transistores usada para construir puertas, sale de este mundo para entrar en el mundo de la ingeniería electrónica. Para nuestros propósitos, sin embargo, nos es suficiente describir cómo se pueden usar las puertas como bloques de construcción, para implementar los circuitos lógicos esenciales de un computador digital.

### A.3. CIRCUITOS COMBINACIONALES

Un circuito combinacional es un conjunto de puertas interconectadas, cuya salida, en un momento dado, es función solamente de la entrada en ese instante. Como ocurre con una puerta sencilla, la aparición de la entrada viene seguida casi inmediatamente por la aparición de la salida, con sólo retardos de puerta.

En general, un circuito combinacional consiste en  $n$  entradas binarias y  $m$  salidas binarias. Como una puerta, un circuito combinacional puede definirse de tres formas:

- Tabla verdad: Para cada una de las  $2^n$  combinaciones posibles de las  $n$  señales de entrada, se enumera el valor binario de cada una de las  $m$  señales de salida.
- Símbolo gráfico: Describe la organización de las interconexiones entre puertas.
- Ecuaciones booleanas: Cada señal de salida se expresa como una función booleana de las señales de entrada.

### IMPLEMENTACIÓN DE LAS FUNCIONES BOOLEANAS

Cualquier función booleana se puede implementar en electrónica en forma de red de puertas. Para una función dada, hay una serie de realizaciones alternativas. Considérese la función

Tabla A.3. Función booleana de tres variables

| A | B | C | F |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

booleana representada por la tabla verdad de la Tabla A.3. Podemos expresar esta función sencillamente, detallando las combinaciones de los valores de A, B y C que hacen que F valga 1:

$$F = \overline{ABC} + \overline{AC} + \overline{AB}$$
 (A.1)

Hay tres combinaciones de los valores de entrada que hacen que F valga 1, y si se da cualquiera de estas tres combinaciones, el resultado será 1. Este tipo de expresión, por razones evidentes, se conoce como la forma *suma de productos* (SOP, «sum of products»). La Figura A.4, muestra una sencilla implementación con puertas AND, OR, y NOT.

Se puede obtener también otra forma de la tabla verdad. La forma SOP indica que la salida es 1, si cualquiera de las combinaciones de entrada que producen 1 es cierta. También se puede decir que la salida es 1, si ninguna de las combinaciones de entrada que producen 0 es cierta. Por tanto:

$$F = (\overline{ABC}) \cdot (\overline{ABC}) \cdot (\overline{ABC}) \cdot (\overline{ABC}) \cdot (\overline{ABC})$$

Esta expresión se puede reescribir usando una generalización del teorema de DeMorgan:

$$\overline{X \cdot Y \cdot Z} = \overline{X} + \overline{Y} + \overline{Z}$$

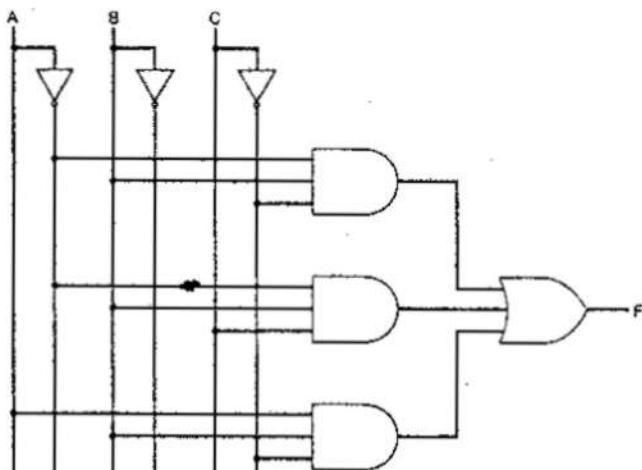


Figura A.4. Implementación de la suma de productos de la Tabla A.3.

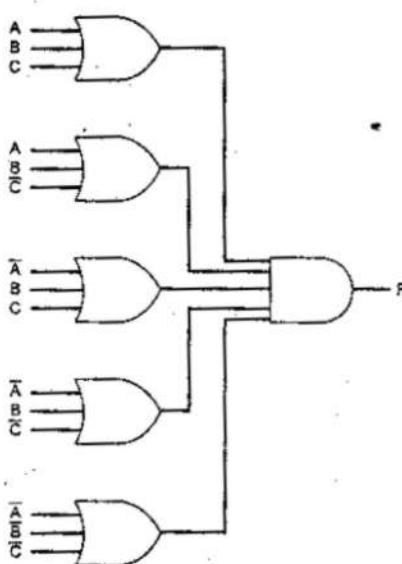


Figura A.5. Implementación del producto de sumas de la Tabla A.3.

Por tanto:

$$\begin{aligned} F &= (\bar{\bar{A}} + \bar{\bar{B}} + \bar{\bar{C}}) \cdot (\bar{\bar{A}} + \bar{\bar{B}} + \bar{\bar{C}}) \cdot (\bar{A} + \bar{\bar{B}} + \bar{\bar{C}}) \cdot (\bar{A} + \bar{\bar{B}} + \bar{\bar{C}}) \cdot (\bar{A} + \bar{\bar{B}} + \bar{\bar{C}}) \\ &= (A + B + C) \cdot (A + B + \bar{C}) \cdot (\bar{A} + B + C) \cdot (\bar{A} + B + \bar{C}) \cdot (\bar{A} + \bar{B} + \bar{C}) \end{aligned} \quad (A.2)$$

Esta última expresión está en la forma de *producto de sumas* (POS, «product of sums»), como se ilustra en la Figura A.5. Por claridad, no aparecen las puertas NOT. En su lugar, suponemos que se dispone de cada señal de entrada y de su complemento. Esto simplifica el diagrama lógico, y hace más legibles las entradas a las puertas.

Por tanto, se puede realizar una función booleana tanto en la forma SOP como en la forma POS. En este momento, podría parecer que la elección dependería de si la tabla verdad contiene más unos o ceros para la función de salida: La SOP tiene un término para cada 1, y la POS tiene un término para cada 0. Sin embargo, hay otras consideraciones:

- Generalmente es posible obtener una expresión booleana más sencilla de la tabla verdad que de las formas SOP o POS.
- Puede ser preferible implementar la función con puertas sencillas (NAND o NOR).

El significado del primer punto es que, con una expresión booleana más sencilla, se necesitan menos puertas para implementar la función. Para llevar a cabo esta simplificación se pueden usar tres métodos:

- Simplificación algebraica
- Mapas de Karnaugh
- Tablas de Quine-McKluskey

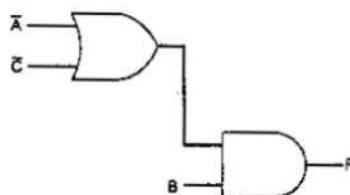


Figura A.6. Implementación simplificada.

### Simplificación algebraica

La simplificación algebraica supone la aplicación de las identidades de la Tabla A.2, que reduce la expresión booleana a otra con menos elementos. Por ejemplo, supongamos de nuevo la Ecuación (A.1). Un poco de razonamiento debería convencer al lector de que una expresión equivalente es:

$$F = \overline{A}B + B\overline{C} \quad (\text{A.3})$$

o, incluso más sencillamente:

$$F = B(\overline{A} + \overline{C})$$

Esta expresión se puede implementar como se indica en la Figura A.6. La simplificación de la Ecuación (A.1) se ha hecho esencialmente por observación. Para obtener una expresión más compleja, se necesita un procedimiento más sistemático.

### Mapas de Karnaugh

Si se quiere simplificar, los mapas de Karnaugh son una forma conveniente de representar una función booleana con pocas variables (de 4 a 6). El mapa es un conjunto de  $2^n$  cuadrículas, que representan las posibles combinaciones de los valores de  $n$  variables binarias. La Figura A.7a muestra el mapa de cuatro cuadrículas para una función de dos variables. Es conveniente, para futuros propósitos, enumerar las combinaciones en el orden 00, 01, 11, 10. Como las cuadrículas corresponden a combinaciones que se van a usar para escribir información, las combinaciones se escriben habitualmente externamente, en la parte superior de las cuadrículas. En el caso de tres variables, la representación es un conjunto de 8 cuadrículas (Figura A.7b), con los valores de una de las variables a la izquierda, y los de las otras dos variables encima de las cuadrículas. Para cuatro variables, se necesitan 16 cuadrículas, con la disposición indicada en la Figura A.7c.

El mapa se puede usar para representar cualquier función booleana de la siguiente forma: Cada cuadrícula corresponde a un único producto en la forma de suma de productos, con valor 1 correspondiente a la variable, y valor 0 correspondiente a la NOT de dicha variable. Por tanto, el producto  $AB$  corresponde a la cuarta cuadrícula de la Figura A.7a. Para cada uno de estos productos de la función, se coloca un 1 en la cuadrícula correspondiente. Por tanto, para el ejemplo de dos variables, el mapa corresponde a  $AB + \overline{A}\overline{B}$ . Dada la tabla verdad de una función booleana, es fácil construir el mapa: para cada combinación de los valores de las variables que dan como resultado 1 en la tabla verdad, se pone un 1 en la cuadrícula correspondiente. La Figura A.7b muestra el resultado para la tabla verdad de la Tabla A.3. Para pasar de una expresión booleana a un mapa, primero es necesario poner la expresión en

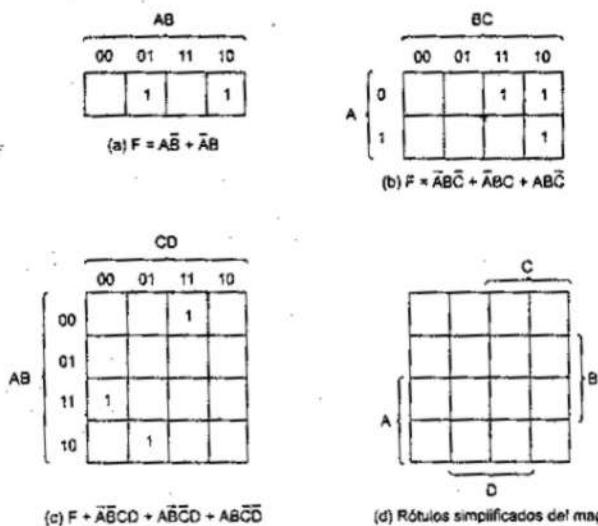


Figura A.7. La utilización de los mapas de Karnaugh para representar funciones booleanas.

lo que se denomina forma *canónica*: cada término de la expresión debe contener cada variable. Así, por ejemplo, si se tiene la Ecuación (A.3), debemos expandirla primero a la forma completa de la Ecuación (A.1) y, después, pasaría al mapa.

Los rótulos usados en la Figura A.7d enfatizan la relación entre las variables, y las filas y columnas del mapa. Aquí, las dos filas que abarca el símbolo A son aquellas en las que la variable A vale 1; las filas que no abarca el símbolo A son aquellas en las que A vale 0 (lo mismo ocurre para B, C y D).

Una vez que se ha creado el mapa de una función, podemos escribir, a menudo, una expresión algebraica sencilla anotando el conjunto de unos del mapa. El principio es el siguiente: dos casillas adyacentes cualesquiera difieren en sólo una de las variables; si dos casillas adyacentes contienen un 1, entonces los correspondientes términos producto difieren sólo en una variable; en tal caso, los dos términos se pueden fundir en uno, eliminando esta variable. Por ejemplo, en la Figura A.8a, las dos casillas adyacentes corresponden a los términos  $ABCD$  y  $ABC\bar{D}$ . Por tanto, la función se puede expresar así:

$$\bar{A}B\bar{C}D + \bar{A}BC\bar{D} = \bar{A}BD$$

Este proceso se puede ampliar de varias formas. En primer lugar, el concepto de adyacencia se puede ampliar para incluir el recubrimiento alrededor del borde del mapa. Por tanto, la casilla más alta de una columna es adyacente a la más baja, y la casilla más a la izquierda de la fila es adyacente a la que está más a la derecha. Estas condiciones se ilustran en las Figuras A.8b y A.8c. En segundo lugar, podemos agrupar no sólo 2 casillas, sino  $2^n$  casillas adyacentes; es decir, 4, 8, etc. Los tres siguientes ejemplos de la Figura A.8, muestran agrupaciones de 4 casillas. Hay que destacar que, en este caso, se pueden eliminar dos de las variables. Los tres últimos ejemplos muestran grupos de 8 cuadrículas, que permiten eliminar tres variables.

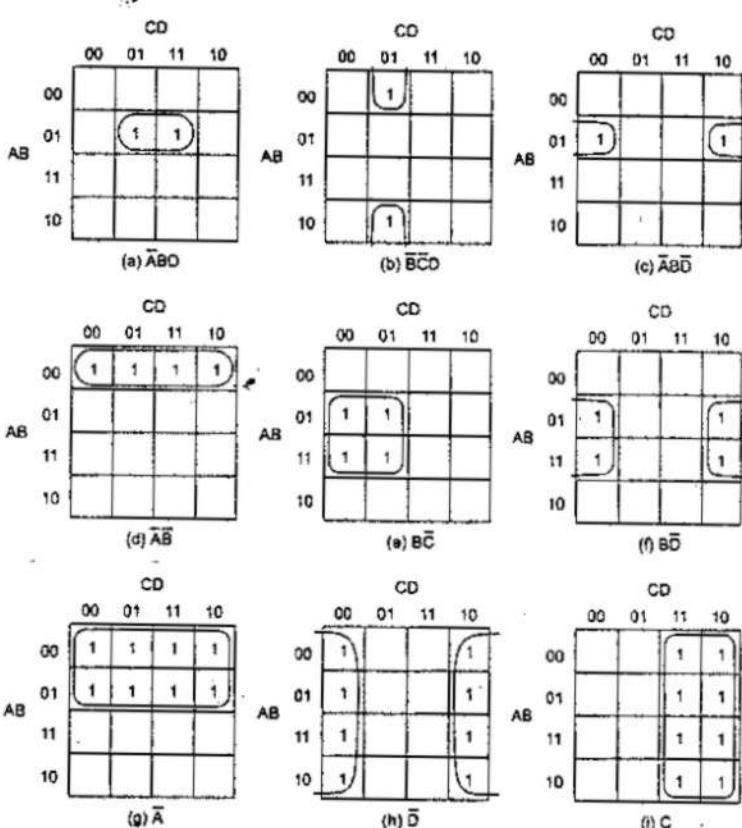


Figura A.8. Utilización de los mapas de Karnaugh.

Para simplificar, podemos resumir las reglas como sigue:

1. Entre las casillas marcadas (casillas con un 1), encontrar aquellas que pertenezcan a un único bloque, lo más grande posible ya sea de 1, 2, 4 u 8 casillas, y rodear el bloque con un círculo.
2. Seleccionar bloques adicionales de casillas marcadas, tan grandes y tan pocas como sea posible, pero que incluyan a cada casilla marcada al menos una vez. Los resultados pueden no ser únicos en algunos casos. Por ejemplo, si una casilla marcada se combina exactamente con otras dos, y no hay una cuarta para completar un grupo mayor, entonces se puede elegir entre dos agrupaciones. Cuando se seleccionan grupos, se permite usar el mismo 1 más de una vez.
3. Seguir dibujando círculos alrededor de las casillas aisladas marcadas, o de parejas de casillas marcadas adyacentes, o de grupos de cuatro, ocho, etc., de forma que cada cuadrado marcado pertenezca al menos a un círculo; luego, utilizar el menor número posible de bloques para incluir a todas las casillas marcadas.

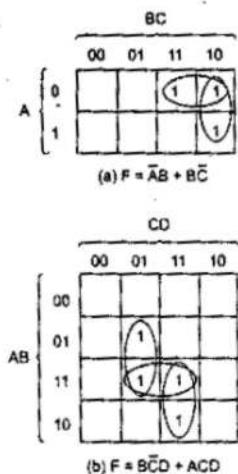


Figura A.9. Grupos solapados.

La Figura A.9a, basada en la Tabla A.3, ilustra el procedimiento. Si queda algún 1 aislado después de haber agrupado entonces, cada uno de ellos se rodea con un círculo, como si fuera un grupo de unos. Finalmente, antes de pasar el mapa a una expresión simplificada booleana, cualquier grupo de unos que esté completamente solapado por otros grupos, se puede eliminar. Es lo que se muestra en la Figura A.9b. En este caso, el grupo horizontal es redundante, y se puede ignorar a la hora de crear la expresión booleana.

Es necesario mencionar una característica adicional de los mapas de Karnaugh. En algunos casos, ciertas combinaciones de valores de las variables no se dan nunca y, por consiguiente, la salida correspondiente no se produce tampoco. Esto se denomina «condiciones de indiferencia». Para cada una de estas condiciones, se coloca la letra «d» en la casilla correspondiente del mapa. Cuando se hace la agrupación y simplificación, cada «d» puede tratarse como un 1 o un 0, eligiendo lo que conduzca a una expresión más sencilla.

Un ejemplo, presentado en [HAYE88], ilustra lo que hemos estado discutiendo. Nos gustaría desarrollar las expresiones booleanas para un circuito que suma un 1 a los dígitos decimales empaquetados. Recordemos de la Sección 9.2, que con decimales empaquetados, cada dígito decimal se representa con un código de 4 bits, de una forma obvia. Así, 0 = 0000, 1 = 0001, ..., 8 = 1000, y 9 = 1001. Las combinaciones de 4 bits restantes, de 1010 a 1111, no se usan. Este código también se denomina «decimal codificado en binario» (BCD, Binary Coded Decimal).

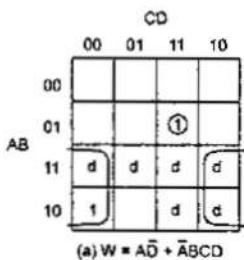
La Tabla A.4 muestra la tabla verdad para producir un resultado de 4 bits, que es la entrada BCD de 4 bits incrementada en 1. La suma es en módulo 10. Así,  $9 + 1 = 0$ . También, hay que notar que seis de los códigos de entrada producen «indiferencias» como resultado, ya que esas no son entradas BCD válidas. La Figura A.10 muestra el resultado de los mapas de Karnaugh para cada una de las variables de salida. Las casillas «d» se usan para lograr las mejores agrupaciones posibles.

Tabla A.4. Tabla verdad de un contador digital de un dígito

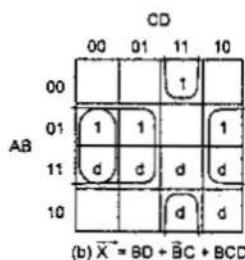
| Número        | Entrada |   |   |   | Número | Salida |   |   |   |
|---------------|---------|---|---|---|--------|--------|---|---|---|
|               | A       | B | C | D |        | W      | X | Y | Z |
| 0             | 0       | 0 | 0 | 0 | 1      | 0      | 0 | 0 | 1 |
| 1             | 0       | 0 | 0 | 1 | 2      | 0      | 0 | 1 | 0 |
| 2             | 0       | 0 | 1 | 0 | 3      | 0      | 0 | 1 | 1 |
| 3             | 0       | 0 | 1 | 1 | 4      | 0      | 1 | 0 | 0 |
| 4             | 0       | 1 | 0 | 0 | 5      | 0      | 1 | 0 | 1 |
| 5             | 0       | 1 | 0 | 1 | 6      | 0      | 1 | 1 | 0 |
| 6             | 0       | 1 | 1 | 0 | 7      | 0      | 1 | 1 | 1 |
| 7             | 0       | 1 | 1 | 1 | 8      | 1      | 0 | 0 | 0 |
| 8             | 1       | 0 | 0 | 0 | 9      | 1      | 0 | 0 | 1 |
| 9             | 1       | 0 | 0 | 1 | 0      | 0      | 0 | 0 | 0 |
| Indiferencias |         |   |   |   |        | d      | d | d | d |
|               | 1       | 1 | 0 | 0 |        | d      | d | d | d |
|               | 1       | 1 | 0 | 1 |        | d      | d | d | d |
|               | 1       | 1 | 1 | 0 |        | d      | d | d | d |
|               | 1       | 1 | 1 | 1 |        | d      | d | d | d |

### El método de Quine-McCluskey

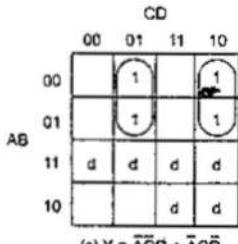
Cuando se incrementa en más de cuatro variables, el método del mapa de Karnaugh se va haciendo cada vez más incómodo. Con cinco variables, se necesitan dos mapas de  $16 \times 16$ , con un mapa situado encima del otro, en tres dimensiones, para conseguir la adyacencia. ¡Seis variables requieren cuatro tablas de  $16 \times 16$  en cuatro dimensiones! Un procedimiento



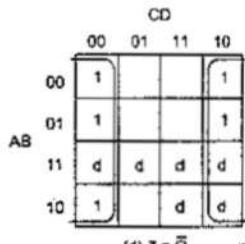
$$(a) W = \bar{A}\bar{D} + \bar{A}B\bar{C}D$$



$$(b) X = BD + \bar{B}C + BCD$$



$$(c) Y = \bar{A}\bar{C}D + \bar{A}C\bar{D}$$



$$(d) Z = \bar{C}$$

Figura A.10. Mapas de Karnaugh para el ejemplo del contador.

alternativo es una técnica tabular, denominada «método de Quine-McKluskey». Este método es adecuado para programar en un computador, y así tener una herramienta automática que produzca expresiones booleanas minimizadas.

Este método se explica mejor mediante un ejemplo. Consideremos la siguiente expresión:

$$ABCD + AB\bar{C}D + A\bar{B}CD + \bar{A}BCD + \bar{A}\bar{B}CD + \bar{A}\bar{B}\bar{C}D + \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}\bar{D}$$

Supongamos que esta expresión se ha obtenido de una tabla verdad. Nos gustaría conseguir una expresión mínima adecuada para implementarla con puertas.

El primer paso es construir una tabla en la que cada fila corresponda a un término producto de la expresión. Los términos se agrupan de acuerdo con el número de variables complementadas. Es decir, empezamos con el término sin complementos, si existe; luego, todos los términos con un complemento; etc. La Tabla A.5 muestra la lista para nuestra expresión ejemplo, indicando con las líneas horizontales las agrupaciones. Para más claridad, cada término se representa con un 1 para cada variable sin complementar, y con un 0 para cada variable complementada. Por tanto, agrupamos términos de acuerdo con el número de unos que contiene. La columna «índice» es simplemente el equivalente decimal, y es útil para lo que se explicará más adelante.

El siguiente paso es encontrar todas las parejas de términos que difieren en sólo una variable, es decir, todos los pares de términos que son iguales excepto en que una variable es 0 en uno de los términos, y 1 en el otro. Debido a la manera en la que se agrupan términos, podemos hacerlo empezando con el primer grupo, y comparar cada término del primer grupo con cada término del segundo grupo. Después, comparamos cada término del segundo grupo con todos los términos del tercer grupo, y así sucesivamente. Cuando se encuentra un emparejamiento, se coloca una marca en cada término, se combina el par eliminando la variable en la que difieren los dos términos, y se añade a la nueva lista. Así, por ejemplo, los términos  $ABCD$  y  $AB\bar{C}D$  se combinan para producir el término  $ABC$ . Este proceso continúa hasta que se haya analizado la tabla original entera. El resultado es una nueva tabla con los siguientes elementos:

|                                   |                   |                 |
|-----------------------------------|-------------------|-----------------|
| $\bar{A}\bar{C}D$                 | $A\bar{B}\bar{C}$ | $ABD\checkmark$ |
| $B\bar{C}D\checkmark$             | $ACD$             |                 |
| $\bar{A}\bar{B}C$                 | $BCD\checkmark$   |                 |
| $\bar{A}\bar{B}\bar{D}\checkmark$ |                   |                 |

Tabla A.5. Primer paso del método de Quine-McKluskey  
(para  $F = ABCD + AB\bar{C}D + A\bar{B}CD + \bar{A}BCD + \bar{A}\bar{B}CD - \bar{A}\bar{B}\bar{C}D + \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}\bar{D}$ )

| Término producto         | índice | A | B | C | D |   |
|--------------------------|--------|---|---|---|---|---|
| $ABCD$                   | 1      | 0 | 0 | 0 | 1 | ✓ |
| $\bar{A}B\bar{C}D$       | 5      | 0 | 1 | 1 | 1 | ✓ |
| $\bar{A}B\bar{C}\bar{D}$ | 6      | 0 | 1 | 1 | 0 | ✓ |
| $A\bar{B}CD$             | 12     | 1 | 1 | 0 | 0 | ✓ |
| $\bar{A}\bar{B}CD$       | 7      | 0 | 1 | 1 | 1 | ✓ |
| $\bar{A}\bar{B}C\bar{D}$ | 11     | 1 | 0 | 1 | 1 | ✓ |
| $A\bar{B}\bar{C}D$       | 13     | 1 | 1 | 0 | 1 | ✓ |
| $ABCD$                   | 15     | 1 | 1 | 1 | 1 | ✓ |

**Tabla A.6.** Último paso del método Quine-McKluskey  
(para  $F = ABCD + AB\bar{C}D + \bar{A}BC\bar{D} + \bar{A}\bar{B}CD + \bar{A}\bar{B}\bar{C}D + \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}\bar{D}$ )

|             | ABCD | $AB\bar{C}D$ | $A\bar{B}C\bar{D}$ | $\bar{A}BCD$ | $\bar{A}B\bar{C}D$ | $\bar{A}\bar{B}C\bar{D}$ | $\bar{A}\bar{B}\bar{C}D$ | $\bar{A}\bar{B}\bar{C}\bar{D}$ |
|-------------|------|--------------|--------------------|--------------|--------------------|--------------------------|--------------------------|--------------------------------|
| BD          | X    | X            |                    |              |                    | X                        |                          | X                              |
| ACD         |      |              |                    |              |                    |                          | X                        | (X)                            |
| ABC         |      |              |                    |              | X                  | (X)                      |                          |                                |
| $\bar{A}BC$ |      | X            | (X)                |              |                    |                          |                          |                                |
| ACD         | X    |              |                    | (X)          |                    |                          |                          |                                |

La nueva tabla se organiza en grupos, como indicamos antes, de la misma forma que la primera tabla. La segunda tabla se procesa después, de la misma manera que la primera. Es decir, se comprueban términos que difieren en sólo una variable y se produce un nuevo término para una tercera tabla. En este ejemplo, la tercera tabla que se hace contiene solamente un término: BD.

En general, el proceso continuaría, a través de sucesivas tablas, hasta una tabla en la que no haya emparejamientos. En este caso, hay implicadas tres tablas.

Una vez se haya completado el proceso descrito anteriormente, tenemos que eliminar muchos de los posibles términos de la expresión. Aquellos términos que no hayan sido eliminados se usan para construir una matriz, como se ilustra en la Tabla A.6. Cada fila de la matriz corresponde a uno de los términos que no se han eliminado (no tiene marca) en cualquiera de las tablas usadas anteriormente. Cada columna se corresponde con uno de los términos de la expresión original. Se coloca una X en cada intersección de una fila y una columna, tal que el elemento de la fila sea «compatible» con el elemento de la columna; es decir, las variables presentes en el elemento de la fila tienen el mismo valor que las variables presentes en el elemento de la columna. Después, se rodea con un círculo cada X que esté sola en una columna. Entonces, se sitúa un cuadrado alrededor de cada X en cualquier fila que tenga un círculo en la X. Si cada columna tiene ahora una X encerrada en un cuadrado o en un círculo, entonces ya se ha concluido, y los elementos de esa fila cuyas X estén marcadas, constituyen la expresión mínima. Por tanto, en nuestro ejemplo, la expresión final es:

$$ABC + ACD + \bar{A}BC + \bar{A}\bar{C}D$$

En los casos en los que algunas columnas no tengan ni un círculo ni un cuadrado, se necesita un proceso adicional. Esencialmente, seguimos añadiendo elementos a la expresión hasta que se cubran todas las columnas.

Resumamos el método Quine-McKluskey para intentar justificar intuitivamente cómo se realiza la minimización. La primera fase de la operación es razonablemente sencilla. El proceso elimina variables innecesarias en términos producto. Entonces, la expresión  $ABC + \bar{A}BC$  es equivalente a  $AB$ , dado que:

$$ABC + AB\bar{C} = AB(C + \bar{C}) = AB$$

Tras la eliminación de las variables, nos queda una expresión que es claramente equivalente a la expresión original. Sin embargo, puede haber términos redundantes en la expresión, como

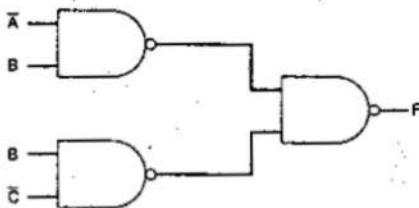


Figura A.11. Implementación con NAND de la Tabla A.3.

encontramos agrupaciones redundantes en los mapas de Karnaugh. La organización de la matriz asegura que se incluye (cubre) cada término de la expresión original, y lo hace de forma que minimiza el número de términos en la expresión final.

### Implementaciones NAND y NOR

Otra consideración en la implementación de funciones booleanas concierne a los tipos de puertas usados. Es a menudo deseable implementar una función booleana sólo con puertas NAND o sólo con puertas NOR. Aunque pueda no ser la implementación con un mínimo de puertas, tiene la ventaja de la regularidad, que puede simplificar el proceso de fabricación. Consideremos de nuevo la Ecuación (A.3):

$$F = B(\bar{A} + \bar{C})$$

Como el complemento del complemento es el valor original,

$$F = B(\bar{A} + \bar{C}) = (\bar{A}B) + (\bar{B}C)$$

aplicando el teorema de DeMorgan,

$$F = (\bar{A}B) \cdot (\bar{B}C)$$

que tiene tres operadores NAND, como se ilustra en la Figura A.11.

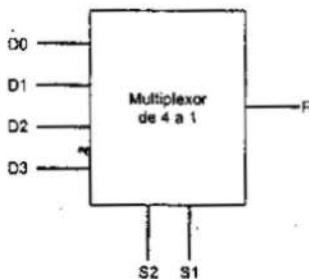


Figura A.12. Representación de multiplexor de 4 a 1.

Tabla A.7. Tabla verdad de un multiplexor 4 a 1

| S2 | S1 | F  |
|----|----|----|
| 0  | 0  | D0 |
| 0  | 1  | D1 |
| 1  | 0  | D2 |
| 1  | 1  | D3 |

## MULTIPLEXORES

El multiplexor conecta varias entradas con una única salida. En un momento dado, se selecciona una de las entradas para que pase a la salida. La Figura A.12 muestra una representación en diagrama de bloques general. Representa un multiplexor de 4 a 1. Hay cuatro líneas de entrada, llamadas D0, D1, D2 y D3. Se selecciona una de estas líneas para dar la señal de salida F. Para seleccionar una de las cuatro entradas posibles, se necesita un código de selección de 2 bits, que se implementa con dos líneas de selección, llamadas S1 y S2.

La tabla verdad de la Tabla A.7 define un ejemplo de un multiplexor de 4 a 1. Es una forma simplificada de una tabla verdad. En vez de mostrar todas las combinaciones posibles de las variables de entrada, muestra la salida como dato procedente de la línea D0, D1, D2 o D3. La Figura A.13 muestra una implementación usando puertas AND, OR y NOT. S1 y S2 se conectan a las puertas AND, de forma que, para cualquier combinación de S1 y S2, tres de las puertas AND tengan la salida a 0. La cuarta puerta, AND, sacará el valor de la línea

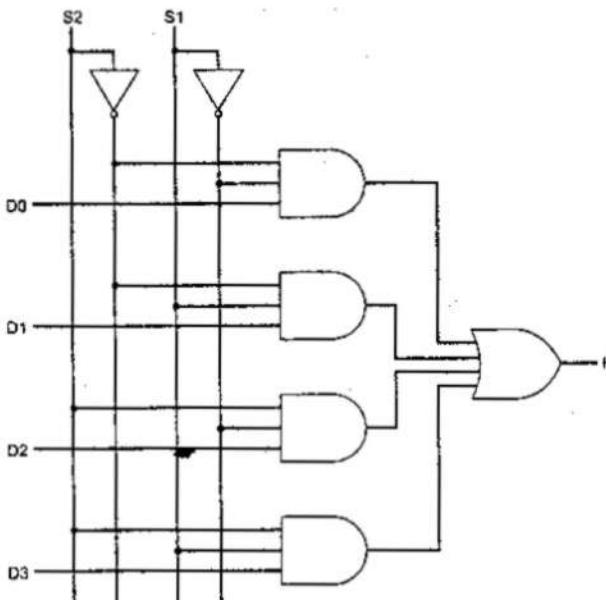


Figura A.13. Implementación de un multiplexor.

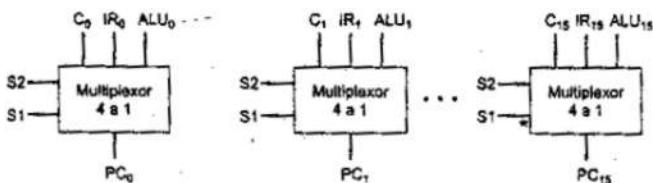


Figura A.14. Multiplexor de entrada al contador de programas.

seleccionada, que será 0 o 1. Por tanto, tres de las entradas de la puerta OR son siempre 0, y la salida de la puerta OR será igual al valor de la puerta de entrada seleccionada. Usando esta organización regular, es fácil construir multiplexores de 8 a 1, de 16 a 1, etc.

Los multiplexores se usan en circuitos digitales para controlar el enruteamiento de señales y datos. Un ejemplo es la carga del contador de programa (PC). El valor a cargar en el contador de programa puede venir de una o varias fuentes diferentes:

- De un contador binario, si el PC se va a incrementar para la siguiente instrucción.
- Del registro de instrucción, si se acaba de ejecutar una instrucción de salto usando direccionamiento directo.
- De la salida de la ALU, si la instrucción de salto especifica la dirección usando modo de desplazamiento.

Las distintas entradas se pueden conectar a las líneas de entrada de un multiplexor con el PC conectado a la línea de salida. Las líneas seleccionadas determinan cual es el valor a cargar en el PC. Como el PC contiene varios bits, se usan varios multiplexores, uno por bit. La Figura A.14 ilustra esto para direcciones de 16 bits.

## DECODIFICADORES

Un decodificador es un circuito combinacional con varias líneas de salida, con una sola de ellas seleccionada en un instante dado, dependiendo del patrón de líneas de entrada. En general, un decodificador tiene  $n$  entradas y  $2^n$  salidas. La Figura A.15 muestra un decodificador con tres entradas y ocho salidas.

Los decodificadores tienen muchos usos en computadores digitales. Un ejemplo es la decodificación de direcciones. Supongamos que queremos construir una memoria de 1 kbyte usando cuatro chips RAM de  $256 \times 8$  bits. Queremos un espacio de direcciones único y unificado, que se pueda descomponer como sigue:

| Address   | Chip |
|-----------|------|
| 0000-00FF | 0    |
| 0100-01FF | 1    |
| 0200-02FF | 2    |
| 0300-03FF | 3    |

Cada chip requiere 8 líneas de dirección, y estos se toman de los 8 bits menos significativos de la dirección. Los 2 bits más significativos de los 10 bits de dirección, se usan para seleccionar uno de los cuatro chips RAM. Para ello, se usa un decodificador de 2 a 4 cuya salida habilita uno de los cuatro chips, como se muestra en la Figura A.16.

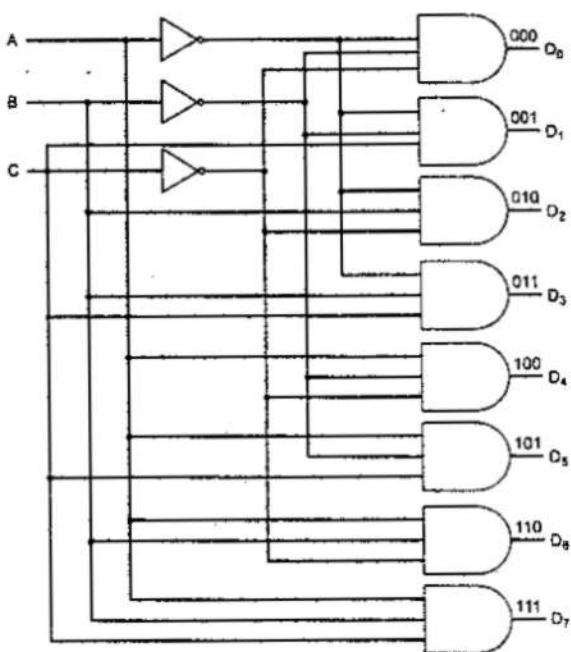


Figura A.15. Decodificador con 3 entradas y  $2^3 = 8$  salidas.

Con una línea adicional de entrada, se puede usar el decodificador como demultiplexor. El demultiplexor realiza la función inversa de un multiplexor: conecta una única entrada a una o varias salidas. Esto se ve en la Figura A.17. Como antes, se decodifican  $n$  entradas para producir una única salida de las  $2^n$ . Con todas las  $2^n$  líneas de salida, se hace la operación AND con un dato de la línea de entrada. Por tanto, las  $n$  entradas actúan como una direc-

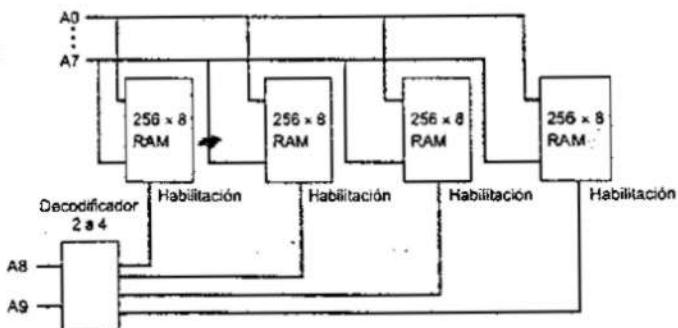


Figura A.16. Decodificación de una dirección.

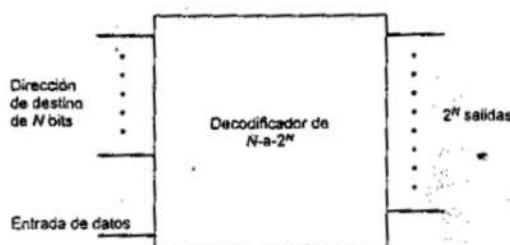


Figura A.17. Implementación de un demultiplexor usando un decodificador.

ción para seleccionar una línea de salida concreta, y el valor del dato de la línea de entrada (0 o 1) se encamina a dicha línea de salida.

La configuración de la Figura A.17 se puede ver de otra forma. Cambiar el nombre de la nueva línea, *entrada de datos*, por *habilitar*. Esto permite el control de la temporización del decodificador. La salida decodificada aparece sólo cuando está presente la entrada codificada y la línea de habilitación vale 1.

#### ARRAY LOGICO PROGRAMABLE (PLA, PROGRAMMABLE LOGIC ARRAY)

Hasta ahora, hemos tratado las puertas individuales como bloques de construcción para realizar funciones arbitrarias. El diseñador podría seguir una estrategia de minimización del número de puertas que van a usarse, manipulando las expresiones booleanas correspondientes.

Como el nivel de integración de los circuitos integrados aumenta, se tienen en cuenta otras consideraciones. Los primeros circuitos integrados, usando una escala de integración pequeña (SSI), contenían de una a diez puertas en un chip. Cada puerta se trataba independientemente, en su uso como bloques de construcción descrito hasta ahora. La Figura A.18 es un ejemplo de algunos chips SSI. Para construir una función lógica, se disponen varios de estos chips en una tarjeta de circuito impreso y se hacen las interconexiones adecuadas entre los terminales de los chips.

El incremento del nivel de integración hace posible añadir más puertas en un chip y hacer interconexiones dentro del chip también. Esto tiene como ventajas la disminución del coste y del tamaño, y el incremento de velocidad (puesto que los retardos dentro del chip son menores que los retardos fuera del chip). Sin embargo, surge un problema de diseño. Para cada función lógica particular o conjunto de funciones, hay que diseñar la organización de las puertas e interconexiones en el chip. El coste y el tiempo implicados en el diseño del chip a medida son altos. Entonces, se convierte en algo atractivo el desarrollo de un chip de uso general que esté listo para adaptarse a usos específicos. Esta es la intención de los *arrays lógicos programables* (PLA, «programmable logic array»).

Los PLA se basan en el hecho de que cualquier función booleana (tabla verdad) se puede expresar en forma de suma de productos (SOP), como hemos visto. Un PLA consiste en una disposición regular de puertas NOT, AND y OR, en un chip. Cada entrada al chip pasa a través de una puerta NOT, así que cada entrada y su complemento están disponibles para cada puerta AND. La salida de cada puerta AND está disponible para cada puerta OR, y la salida de cada puerta OR es una salida del chip. Haciendo las conexiones adecuadas, se pueden implementar expresiones SOP arbitrarias.

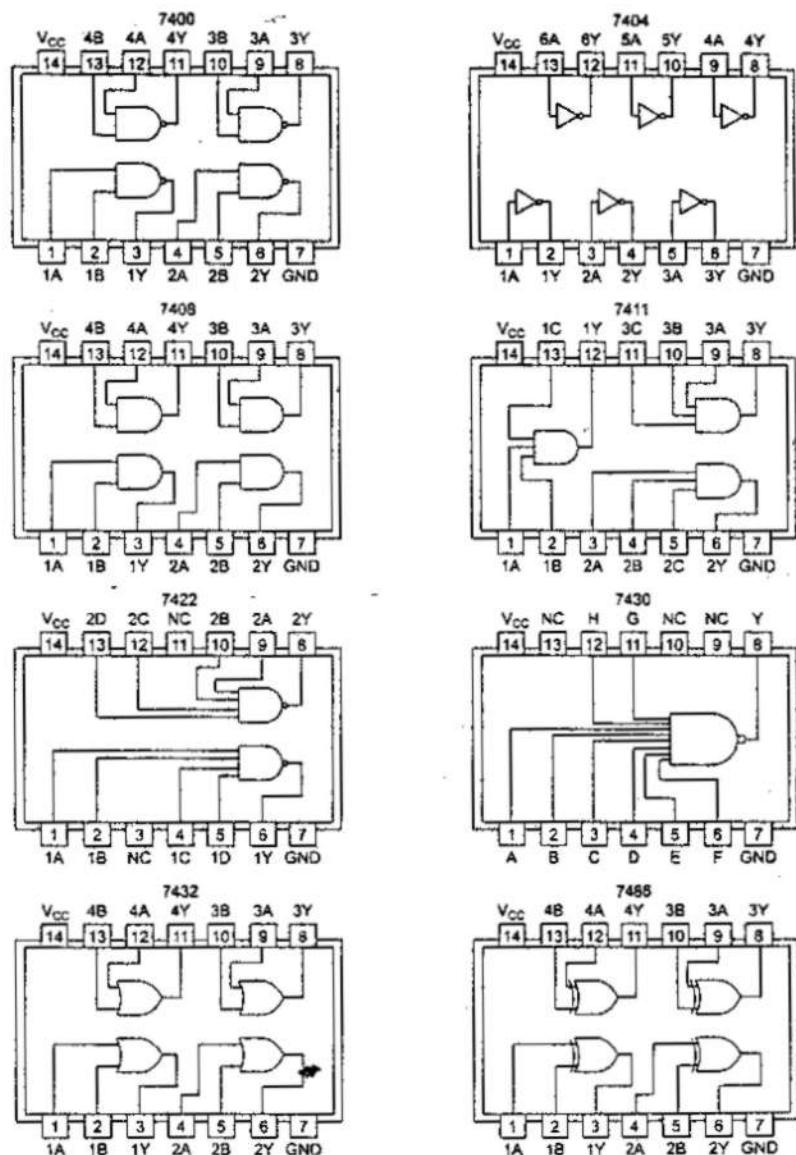


Figura A.18. Algunos chips SSI. Organización de los pines de *The TTL data book for design engineers*, copyright © 1976 Texas Instruments Incorporated.

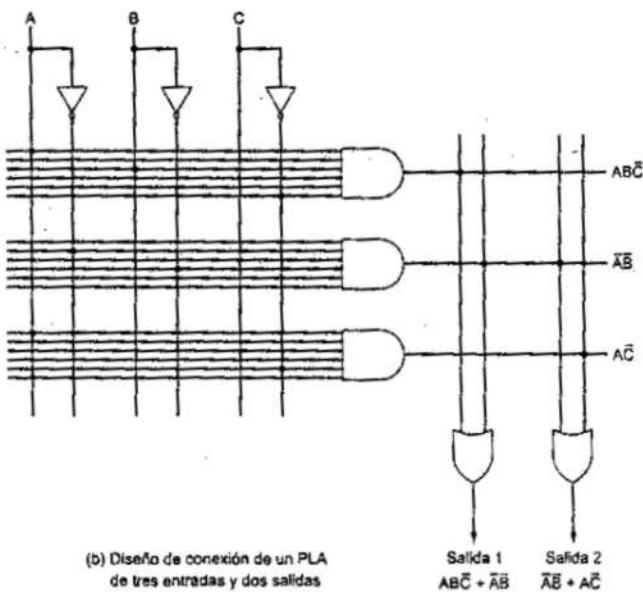
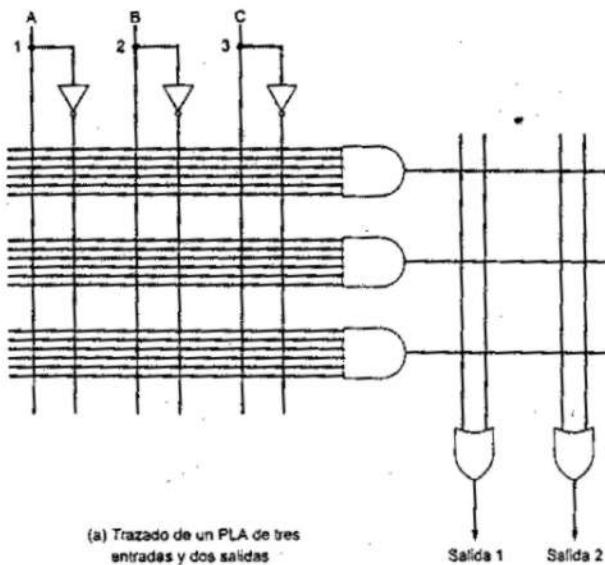


Figura A.19. Ejemplo de un conjunto lógico programable.

La Figura A.19 muestra un PLA con tres entradas, ocho puertas y dos salidas. Los PLA mayores contienen varios cientos de puertas, de 15 a 25 entradas, y de 5 a 15 salidas. Las conexiones de las entradas a las puertas AND, y de las puertas AND a las puertas OR, no están especificadas.

Los PLA se fabrican de dos formas diferentes para permitir una programación más fácil (hacer las conexiones). En la primera, cada conexión posible se hace a través de fusible en cada punto de intersección. Este tipo de PLA se denomina *array lógico programable de campo*. Alternativamente, las propias conexiones se pueden hacer durante la fabricación del chip usando una máscara adecuada dada para un patrón de interconexión particular. En cualquiera de los casos, el PLA proporciona una forma de implementación de funciones lógicas digitales flexible y barata.

En la Figura A.19b se muestra un diseño que sintetiza dos expresiones booleanas.

### MEMORIA DE SOLO LECTURA (ROM, READ ONLY MEMORY)

A los circuitos combinacionales se les llama a veces circuitos «sin memoria», ya que su salida depende sólo de la entrada actual y no retiene la historia de las entradas anteriores. Sin embargo, hay un tipo de memoria que se implementa con circuitos combinacionales, llamada *memoria de solo lectura* (ROM).

Recordemos que una memoria ROM es una unidad de memoria en la que sólo se realiza la operación de lectura. Esto implica que la información binaria almacenada en una ROM es permanente, y se crea en el proceso de fabricación. Entonces, una entrada dada a la ROM (líneas de direcciones) siempre produce la misma salida (líneas de datos). Como las salidas son función sólo de las entradas presentes, la ROM es, de hecho, un circuito combinacional.

Se puede implementar una ROM con un decodificador y un conjunto de puertas OR. Como ejemplo, consideremos la Tabla A.8. Esta se puede ver como una tabla verdad con cuatro entradas y cuatro salidas. Para cada uno de los 16 posibles valores de entrada, se muestra el conjunto correspondiente de valores de salida. También se puede ver como la definición del contenido de una ROM de 64 bits de 16 palabras de 4 bits cada una. Las cuatro entradas determinan una dirección, y las cuatro salidas especifican el contenido de posición indicada.

Tabla A.8. Tabla verdad de una ROM

| Entrada |   |   |   | Salida |   |   |   |
|---------|---|---|---|--------|---|---|---|
| 0       | 0 | 0 | 0 | 0      | 0 | 0 | 0 |
| 0       | 0 | 0 | 1 | 0      | 0 | 0 | 1 |
| 0       | 0 | 1 | 0 | 0      | 0 | 1 | 1 |
| 0       | 0 | 1 | 1 | 0      | 0 | 1 | 0 |
| 0       | 1 | 0 | 0 | 0      | 1 | 1 | 0 |
| 0       | 1 | 0 | 1 | 0      | 1 | 1 | 1 |
| 0       | 1 | 1 | 0 | 0      | 1 | 0 | 1 |
| 0       | 1 | 1 | 1 | 0      | 1 | 0 | 0 |
| 1       | 0 | 0 | 0 | 1      | 1 | 0 | 0 |
| 1       | 0 | 0 | 1 | 1      | 1 | 0 | 1 |
| 1       | 0 | 1 | 0 | 1      | 1 | 1 | 1 |
| 1       | 0 | 1 | 1 | 1      | 1 | 1 | 0 |
| 1       | 1 | 0 | 0 | 1      | 0 | 1 | 0 |
| 1       | 1 | 0 | 1 | 1      | 0 | 1 | 1 |
| 1       | 1 | 1 | 0 | 1      | 0 | 0 | 1 |
| 1       | 1 | 1 | 1 | 1      | 0 | 0 | 0 |

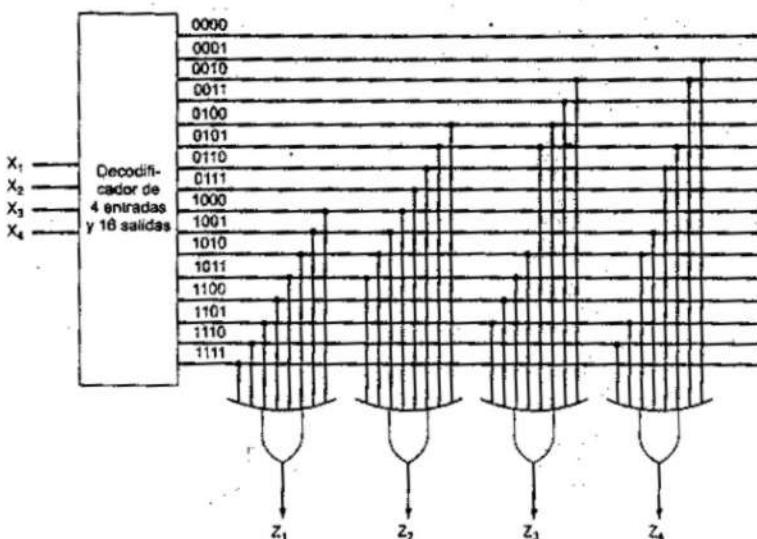


Figura A.20. ROM de 64 bit.

en la dirección. En la Figura A.20 se muestra cómo podría implementarse esta memoria usando un decodificador de 4 a 16 y cuatro puertas OR. Como en un PLA, se usa una organización regular, y las interconexiones se hacen de forma que reflejen el resultado deseado.

### SUMADORES

Hasta ahora hemos visto cómo se pueden usar puertas interconectadas para implementar funciones como enrutamiento de señales, decodificación y ROM. Un área esencial, a la que no nos hemos dirigido todavía, es la aritmética. En esta breve visión general, nos contentaremos con ver la función de suma.

La suma binaria difiere de la del álgebra booleana en que el resultado incluye un término de acarreo. Así:

$$\begin{array}{r}
 0 & 0 & 1 & 1 \\
 + 0 & + 1 & + 0 & + 1 \\
 \hline
 0 & 1 & 1 & 10
 \end{array}$$

No obstante, la suma se puede implementar con términos booleanos. En la Tabla A.9 puede verse la lógica para sumar dos bits de entrada para producir un bit de suma y un bit de acarreo. Esta tabla verdad podría implementarse fácilmente en lógica digital. Sin embargo, no estamos interesados en realizar la suma con sólo un par de bits. Más bien, queremos sumar dos números de  $n$  bits. Esto se puede hacer poniendo juntos un conjunto de sumadores, de forma que el acarreo de un sumador sea la entrada del siguiente. En la Figura A.21 se muestra un sumador de 4 bits.

Tabla A.9. Tabla verdad de la suma binaria

| (a) Suma de 1 bit aislado |   |      |        | (b) Suma con acarreo de entrada |   |   |      |                  |
|---------------------------|---|------|--------|---------------------------------|---|---|------|------------------|
| A                         | B | Suma | Acceso | C <sub>in</sub>                 | A | B | Suma | C <sub>out</sub> |
| 0                         | 0 | 0    | 0      | 0                               | 0 | 0 | 0    | 0                |
| 0                         | 1 | 1    | 0      | 0                               | 0 | 1 | 1    | 0                |
| 1                         | 0 | 1    | 0      | 0                               | 1 | 0 | 0    | 0                |
| 1                         | 1 | 0    | 1      | 0                               | 1 | 1 | 0    | 1                |
|                           |   |      |        | 1                               | 0 | 0 | 1    | 0                |
|                           |   |      |        | 1                               | 0 | 1 | 0    | 1                |
|                           |   |      |        | 1                               | 1 | 0 | 0    | 1                |
|                           |   |      |        | 1                               | 1 | 1 | 1    | 1                |

Para que funcione un sumador de varios bits, cada uno de los sumadores de un bit debe tener tres entradas, incluyendo el acarreo del sumador inmediato inferior. La tabla verdad revisada aparece en la Tabla A.9b. Las dos salidas se pueden expresar:

$$\text{Sum} = \overline{ABC} + \overline{ABC} + ABC + \overline{ABC}$$

$$\text{Carry} = AB + AC + BC$$

La Figura A.22 es una implementación usando puertas AND, OR y NOT.

Entonces, tenemos la lógica necesaria para implementar un sumador de varios bits como se muestra en la Figura A.23. Hay que notar que, como la salida de cada sumador depende del acarreo del sumador previo, hay un retardo que crece del bit menos significativo al más significativo. Cada sumador de un bit experimenta cierta cantidad del retardo de puerta, y este retardo de puerta se acumula. Para sumadores grandes, el retardo acumulado puede hacerse inaceptablemente alto.

Si los valores de acarreo se pudieran determinar sin tener que pasar a través de todas las etapas previas, entonces cada sumador de un bit podría funcionar independientemente, y el retardo no se acumularía. Éste se puede lograr con un procedimiento conocido como *acarreo anticipado*. Veamos de nuevo el sumador de 4 bits para explicar este método.

Nos gustaría proponer una expresión que especificara el acarreo de entrada a cualquier etapa del sumador, sin referirnos a los valores de acarreo previos. Tenemos:

$$C_0 = A_0 B_0 \quad (A.4)$$

$$C_1 = A_1 B_1 + A_1 A_0 B_0 + B_1 A_0 B_0 \quad (A.5)$$

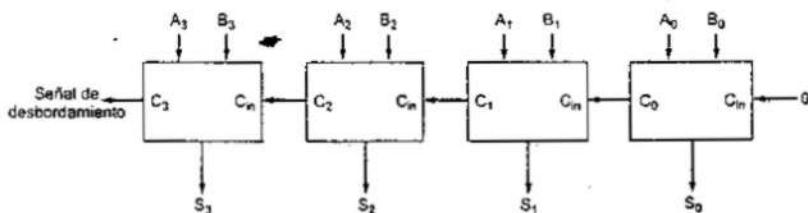


Figura A.21. Sumador de 4 bits.

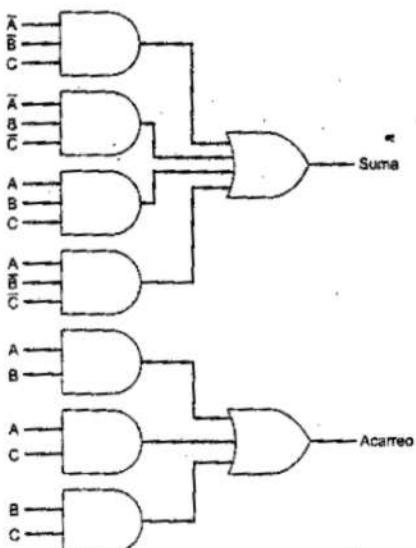


Figura A.22. Implementación de un sumador.

Siguiendo el mismo procedimiento, tenemos:

$$C_2 = A_3B_3 + A_2A_1B_1 + A_2A_1A_0B_0 + A_2B_1A_0B_0 + B_2A_1B_1 + B_2A_1A_0B_0 + B_2B_1A_0B_0$$

Este proceso se puede repetir para sumadores arbitrariamente grandes. Cada término de acarreo se puede expresar en forma SOP como función de sólo las entradas originales, sin dependencia de los acarreos. Por tanto, sólo se dan dos niveles de retardo de puerta, a pesar del tamaño del sumador.

Para números grandes, este procedimiento se vuelve demasiado complicado. Evaluando la expresión para el bit más significante de un sumador de  $n$  bits, se requiere una puerta OR con  $n - 1$  entradas, y  $n$  puertas AND de 2 a  $n + 1$  entradas. Por consiguiente, el acarreo anticipado se hace normalmente con sólo 4 u 8 bits a la vez. La Figura A.23 muestra cómo se puede construir un sumador de 32 bits a partir de cuatro sumadores de 8 bits. En este caso, el acarreo debe pasar a través de los cuatro sumadores de 8 bits, pero puede ser sustancialmente más rápido que pasar a través de 32 sumadores de 1 bit.

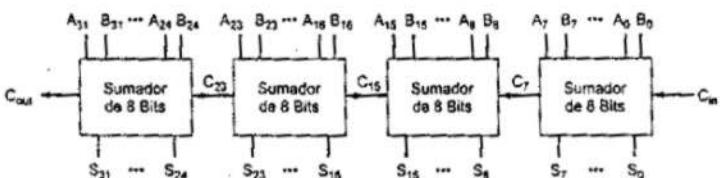


Figura A.23. Construcción de un sumador de 32 bits usando sumadores de 8 bits.

## A.4. CIRCUITOS SECUENCIALES

Los circuitos combinacionales implementan las funciones esenciales de un computador digital. Sin embargo, excepto para el caso especial de ROM, no proporcionan memoria o información de estado, elementos también esenciales para el funcionamiento de un computador digital. Para estos últimos fines, se usa una forma de circuitos lógicos digitales más compleja: los circuitos secuenciales. La salida actual de un circuito secuencial depende no sólo de la entrada actual, sino también de la historia pasada de las entradas. Otra manera más general y útil de ver esto es que la salida actual de un circuito secuencial depende de la entrada actual y del estado actual del circuito.

En esta sección, examinaremos algunos ejemplos sencillos, pero útiles, de circuitos secuenciales. Como veremos, los circuitos secuenciales se implementan con circuitos combinacionales.

### BIESTABLES

La forma más sencilla de un circuito secuencial es un biestable. Hay varios tipos de biestables, y todos ellos comparten dos propiedades:

- El biestable es un dispositivo con dos estados. Está en uno de dos estados, en ausencia de entrada, recordando el último estado. Entonces, el biestable puede funcionar como una memoria de 1 bit.
- El biestable tiene dos salidas, que son siempre complementarias. Normalmente se denominan  $Q$  y  $\bar{Q}$ .

### El cerrojo (latch) S-R

La Figura A.24 muestra una configuración común conocida como biestable S-R o *cerrojo S-R*. El circuito tiene dos entradas,  $S$  (Set) y  $R$  (Reset), y dos salidas,  $Q$  y  $\bar{Q}$ , y consiste en dos puertas NOR conectadas por realimentación.

Primero, mostremos que el circuito es biestable. Supongamos que  $S$  y  $R$  valen 0, y que  $Q$  es 0. Las entradas a la puerta NOR inferior son  $Q = 0$  y  $S = 0$ . Entonces, la salida  $Q = 1$  significa que las entradas a la puerta NOR superior son  $Q = 1$  y  $R = 0$ , con salida  $Q = 0$ . Por tanto, el estado del circuito es internamente consistente y permanece estable, mientras  $S = R = 0$ . Con un razonamiento similar se llega a que el estado  $Q = 1$ ,  $\bar{Q} = 0$ , es también estable para  $R = S = 0$ .

Por tanto, estos circuitos pueden funcionar como una memoria de 1bit. Podemos ver la salida  $Q$  como el «valor» del bit. Las entradas  $S$  y  $R$  sirven para escribir los valores, 1 y 0

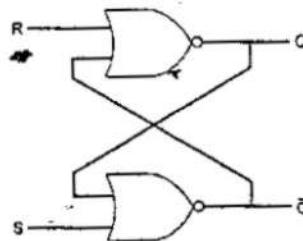


Figura A.24. Implementación del biestable S-R con puertas NOR

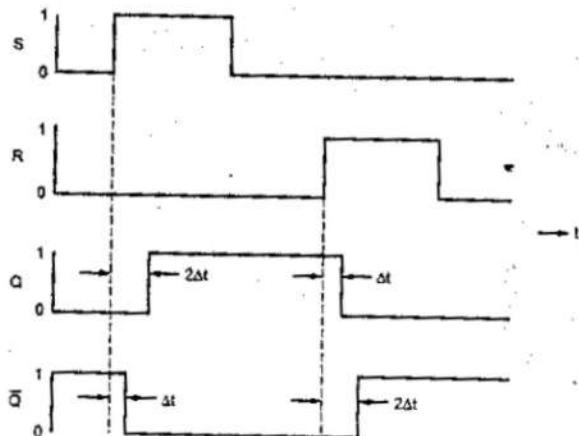


Figura A.25. Diagrama de tiempo del biestable S-R con NOR.

respectivamente, en la memoria. Para ver esto, consideremos el estado  $Q = 0$ ,  $\bar{Q} = 1$ ,  $S = 0$ ,  $R = 0$ . Supongamos que  $S$  cambia al valor 1. Ahora, las entradas a la puerta NOR inferior son  $S = 1$ ,  $Q = 0$ . Después de cierto tiempo de retraso  $\Delta t$ , la salida de la puerta NOR inferior será  $Q = 0$  (ver Figura A.25). Así que, en este momento, las entradas a la puerta NOR superior pasan a ser  $R = 0$ ,  $\bar{Q} = 0$ . Después de otro retraso de puerta de  $\Delta t$ , la salida  $Q$  pasa a 1. Este es, de nuevo, un estado estable. Las entradas de la puerta inferior son ahora  $S = 1$ ,  $Q = 1$ , que mantienen la salida  $Q = 0$ . Mientras  $S = 1$  y  $R = 0$ , las salidas seguirán siendo  $Q = 1$ ,  $\bar{Q} = 0$ . Además, si  $S$  vuelve a 0, las salidas permanecerán sin cambiar.

La salida  $R$  realiza la función contraria. Cuando  $R$  vale 1, fuerza  $Q = 0$ ,  $\bar{Q} = 1$ , sin importar el estado previo de  $Q$  y  $\bar{Q}$ . De nuevo, hay un tiempo de retraso de  $2\Delta t$  antes de que se restablezca la estabilidad (Figura A.25).

El biestable S-R se puede definir con una tabla parecida a la tabla verdad, llamada *tabla característica*, que muestra el siguiente estado o estados de un circuito secuencial, en función de los estados y entradas actuales. En el caso del biestable S-R, el estado se puede definir por el valor de  $Q$ . La Tabla A.10a muestra la tabla característica resultante. Se observa que las entradas  $S = 1$ ,  $R = 1$  no están permitidas, ya que producirían una salida inconsistente ( $Q$  y  $\bar{Q}$  igual a 0). La tabla se puede expresar de forma más compacta, como en la Tabla A.10b. En la Tabla A.10c se muestra una ilustración del comportamiento del biestable S-R.

### Biestable S-R sincrónico

La salida del cerrojo S-R cambia, tras un breve tiempo de retraso, en respuesta a un cambio en la entrada. Esto se denomina *operación asincrónica*. La mayoría de los acontecimientos en computadores digitales están sincronizados por un pulso de reloj, así que los cambios ocurren sólo en un pulso de reloj. La Figura A.26 muestra esta disposición. Este dispositivo se denomina *biestable S-R sincrónico*. Nótese que las entradas  $S$  y  $R$  se aplican a las entradas de las puertas NOR sólo durante el pulso de reloj.

Tabla A.10. El cerrojo S-R

| (a) Tabla de características |               |                  | (b) Tabla característica simplificada |   |           |
|------------------------------|---------------|------------------|---------------------------------------|---|-----------|
| Entrada actual               | Estado actual | Estado siguiente | S                                     | R | $Q_{n+1}$ |
| SR                           | $Q_n$         | $Q_{n+1}$        |                                       |   |           |
| 00                           | 0             | 0                | 0                                     | 0 | $Q_n$     |
| 00                           | 1             | 1                | 0                                     | 1 | 0         |
| 01                           | 0             | 0                | 1                                     | 0 | 1         |
| 01                           | 1             | —                |                                       | 1 | —         |
| 10                           | 0             | 1                |                                       |   |           |
| 10                           | 1             | 1                |                                       |   |           |
| 11                           | 0             | —                |                                       |   |           |
| 11                           | 1             | —                |                                       |   |           |

| (c) Respuesta a una serie de entradas |   |   |   |   |   |   |   |   |   |   |
|---------------------------------------|---|---|---|---|---|---|---|---|---|---|
| t                                     | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| S                                     | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| R                                     | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| $Q_{n+1}$                             | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

### Biestable D

Un problema con los biestables S-R es que la condición  $R = 1, S = 1$  debe ser evitada. Una manera de hacerlo es permitir sólo una única entrada. El biestable D lo cumple. La Figura A.27 muestra una implementación con puertas y la tabla característica del biestable D. Usando un inversor, las entradas de no reloj de las dos puertas AND garantizan ser una la opuesta de la otra.

El biestable D se denomina a veces «biestable de datos» porque es, en efecto, almacén para un bit de datos. La salida del biestable D es siempre igual al valor más reciente aplicado a la entrada. Por tanto, recuerda y produce la última entrada. También se le llama «biestable de retraso», porque retraza un 0 o un 1 aplicado a la entrada durante un pulso de reloj.

### Biestable J-K

Otro biestable útil, es el biestable J-K. Como el biestable S-R, tiene dos entradas. Sin embargo, en este caso, todas las combinaciones posibles de los valores de entrada son válidos. La Figura A.28 muestra una implementación con puertas del biestable J-K, y la Figura A.29

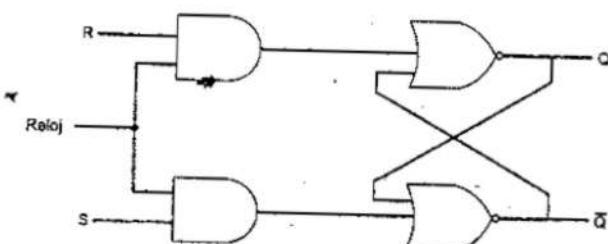


Figura A.26. Biestable S-R sincronizador.

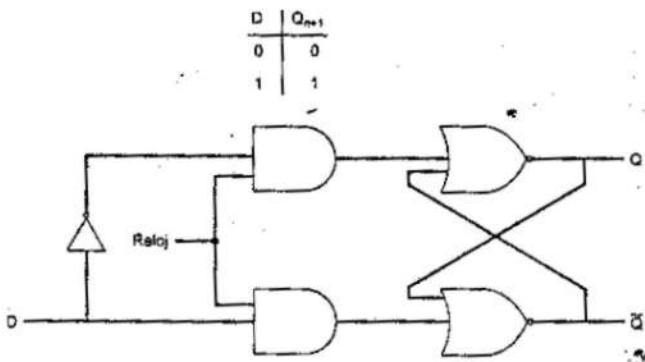


Figura A.27. Biestable D.

muestra su tabla característica (junto con las de los biestables S-R y D). Nótese que las tres primeras combinaciones son las mismas que para el biestable S-R. Sin entrada, la salida es estable. La entrada J sola, realiza la función de puesta a 1, causando que la salida sea 1; la entrada K sola, realiza la función de puesta a cero, provocando que la salida sea 0. Cuando J y K son 1, la función realizada se denomina función de *comutación*: la salida se invierte. Si Q vale 1 y se aplica un 1 a J y K, entonces Q se hace 1. El lector debería verificar que la implementación de la Figura A.28 produce esta función característica.

### REGISTROS

Como ejemplo del uso de los biestables, examinemos primero uno de los elementos esenciales de la CPU: el registro. Como sabemos, un registro es un circuito digital usado en la CPU para almacenar uno o más bits de datos. Normalmente, se usan dos tipos básicos de registros: registros paralelos y de desplazamiento.

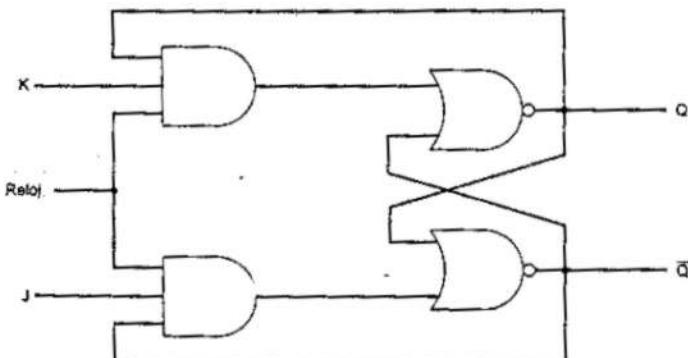


Figura A.28. Biestable J-K.

| Nombre | Simbolo gráfico | Tabla característica                                                                                                                                                                                                                                                                                                              |   |           |           |   |   |       |   |   |   |   |   |   |   |   |             |
|--------|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---|-----------|-----------|---|---|-------|---|---|---|---|---|---|---|---|-------------|
| S-R    |                 | <table border="1"> <thead> <tr> <th>S</th><th>R</th><th><math>Q_{n+1}</math></th></tr> </thead> <tbody> <tr> <td>0</td><td>0</td><td><math>Q_n</math></td></tr> <tr> <td>0</td><td>1</td><td>0</td></tr> <tr> <td>1</td><td>0</td><td>1</td></tr> <tr> <td>1</td><td>1</td><td>-</td></tr> </tbody> </table>                      | S | R         | $Q_{n+1}$ | 0 | 0 | $Q_n$ | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | -           |
| S      | R               | $Q_{n+1}$                                                                                                                                                                                                                                                                                                                         |   |           |           |   |   |       |   |   |   |   |   |   |   |   |             |
| 0      | 0               | $Q_n$                                                                                                                                                                                                                                                                                                                             |   |           |           |   |   |       |   |   |   |   |   |   |   |   |             |
| 0      | 1               | 0                                                                                                                                                                                                                                                                                                                                 |   |           |           |   |   |       |   |   |   |   |   |   |   |   |             |
| 1      | 0               | 1                                                                                                                                                                                                                                                                                                                                 |   |           |           |   |   |       |   |   |   |   |   |   |   |   |             |
| 1      | 1               | -                                                                                                                                                                                                                                                                                                                                 |   |           |           |   |   |       |   |   |   |   |   |   |   |   |             |
| J-K    |                 | <table border="1"> <thead> <tr> <th>J</th><th>K</th><th><math>Q_{n+1}</math></th></tr> </thead> <tbody> <tr> <td>0</td><td>0</td><td><math>Q_n</math></td></tr> <tr> <td>0</td><td>1</td><td>0</td></tr> <tr> <td>1</td><td>0</td><td>1</td></tr> <tr> <td>1</td><td>1</td><td><math>\bar{Q}_n</math></td></tr> </tbody> </table> | J | K         | $Q_{n+1}$ | 0 | 0 | $Q_n$ | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | $\bar{Q}_n$ |
| J      | K               | $Q_{n+1}$                                                                                                                                                                                                                                                                                                                         |   |           |           |   |   |       |   |   |   |   |   |   |   |   |             |
| 0      | 0               | $Q_n$                                                                                                                                                                                                                                                                                                                             |   |           |           |   |   |       |   |   |   |   |   |   |   |   |             |
| 0      | 1               | 0                                                                                                                                                                                                                                                                                                                                 |   |           |           |   |   |       |   |   |   |   |   |   |   |   |             |
| 1      | 0               | 1                                                                                                                                                                                                                                                                                                                                 |   |           |           |   |   |       |   |   |   |   |   |   |   |   |             |
| 1      | 1               | $\bar{Q}_n$                                                                                                                                                                                                                                                                                                                       |   |           |           |   |   |       |   |   |   |   |   |   |   |   |             |
| D      |                 | <table border="1"> <thead> <tr> <th>D</th><th><math>Q_{n+1}</math></th></tr> </thead> <tbody> <tr> <td>0</td><td>0</td></tr> <tr> <td>1</td><td>1</td></tr> </tbody> </table>                                                                                                                                                     | D | $Q_{n+1}$ | 0         | 0 | 1 | 1     |   |   |   |   |   |   |   |   |             |
| D      | $Q_{n+1}$       |                                                                                                                                                                                                                                                                                                                                   |   |           |           |   |   |       |   |   |   |   |   |   |   |   |             |
| 0      | 0               |                                                                                                                                                                                                                                                                                                                                   |   |           |           |   |   |       |   |   |   |   |   |   |   |   |             |
| 1      | 1               |                                                                                                                                                                                                                                                                                                                                   |   |           |           |   |   |       |   |   |   |   |   |   |   |   |             |

Figura A.29. Biestables básicos.

### Registros paralelos

Un registro paralelo consiste en un conjunto de memorias de 1 bit que se pueden leer o escribir simultáneamente. Se usa para almacenar datos. Los registros de los que hemos hablado a lo largo de este libro son registros paralelos.

El registro paralelo de 8 bits de la Figura A.30 ilustra el funcionamiento de un registro paralelo. Se usan cerrojos S-R. Una señal de control, llamada *validación de dato de entrada*, controla la escritura en los registros de los valores provenientes de las líneas de señales, de la D11 a la D18. Estas líneas pueden ser salidas de multiplexores, ya que los datos que se pueden cargar en un registro pueden provenir de una gran variedad de fuentes. La salida se controla de una forma similar. Como una característica extra, hay disponible una línea de puesta a cero (reset), que permite poner a 0 fácilmente el registro. Nótese que esto no podría realizarse tan fácilmente con un registro construido con biestables D.

### Registro de desplazamiento

Un registro de desplazamiento acepta y/o transfiere información vía serie. Consideremos, por ejemplo, la Figura A.31, que muestra un registro de desplazamiento de 5 bits construido a

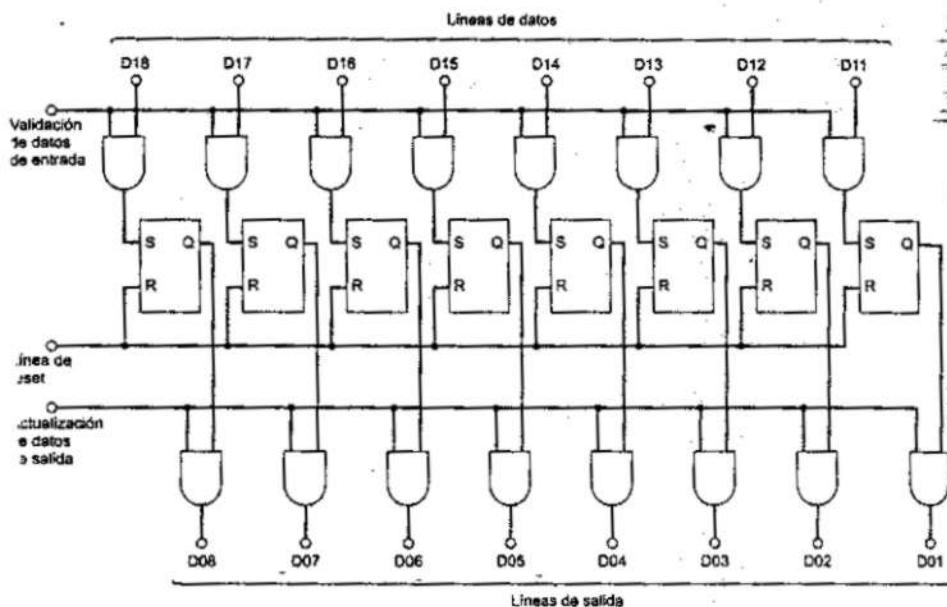


Figura A.30. Registro paralelo de 8 bits.

partir de biestables S-R síncronos. Los datos se introducen únicamente a través del biestable que está más a la izquierda. Con cada pulso de reloj, los datos se desplazan a la derecha una posición, y el bit más a la derecha se transfiere fuera.

Los registros de desplazamiento se pueden usar como interfaz de dispositivos serie de E/S. Además, pueden usarse en la ALU para realizar desplazamiento lógicos y funciones de rotación. En este último uso, necesitan equiparse con circuitería de lectura/escritura, tanto paralela como serié.

## CONTADORES

Otra categoría útil de circuitos secuenciales es la de los contadores. Un contador es un registro cuyo valor se puede incrementar fácilmente en 1 módulo la capacidad de ese registro. Así,

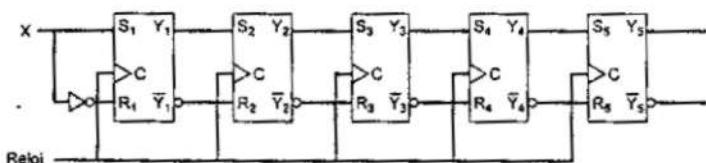


Figura A.31. Registro de desplazamiento de 5 bits.

un registro hecho con  $n$  biestables puede contar hasta  $2^n - 1$ . Cuando el contador se incrementa más allá de su valor máximo, se pone a 0. Un ejemplo de un contador en la CPU es el contador de programa.

Los contadores pueden ser asíncronos o síncronos, dependiendo de la forma en que operen. Los contadores asíncronos son relativamente lentos, ya que la salida de un biestable produce un cambio en el estado del siguiente biestable. En un contador síncrono, todos los biestables cambian de estado a la vez. Como el último tipo es mucho más rápido, es el tipo que se usa en las CPU. Sin embargo, es útil empezar la discusión con una descripción del contador asíncrono.

### Contador asíncrono

Un contador asíncrono se denomina también «contador de onda» (ripple), ya que el cambio que se produce para incrementar el contador empieza en un extremo y se transfiere como una «onda» hasta el otro extremo. La Figura A.32 muestra una implementación de un contador de 4 bits usando biestables J-K, junto con un diagrama de tiempo que ilustra su comportamiento. El diagrama de tiempo está idealizado, ya que no muestra el retardo de propagación que se produce cuando las señales bajan en la serie de biestables. La salida del biestable más a la izquierda ( $Q_0$ ) es el bit menos significante. El diseño se podría ampliar fácilmente a un número arbitrario de bits añadiendo en cascada más biestables.

En la implementación mostrada, el contador se incrementa con cada pulso de reloj. Las entradas J y K de cada biestable se mantienen a 1 constante. Esto quiere decir que, cuando hay un pulso de reloj, la salida en Q se invierte (de 1 a 0; de 0 a 1). Nótese que el cambio de

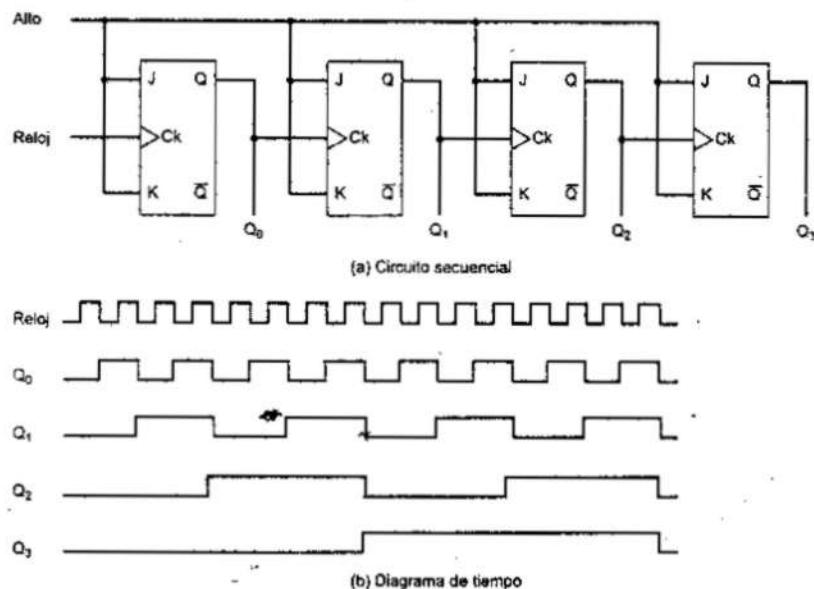


Figura A.32. Contador ondulado.

estado se produce cuando cae el flanco del pulso de reloj; esto se conoce como «biestable disparado por flanco». Usando biestables que responden a la transición en un pulso de reloj en vez de en el pulso mismo, proporciona un control del tiempo mejor en circuitos complejos. Si se analizan los patrones de salida de este contador, se puede ver el ciclo 0000, 0001, ..., 1110, 1111, 0000, etc.

### Contadores síncronos

El contador asíncrono tiene la desventaja del retraso asociado al cambio de valor, que es proporcional al tamaño del contador. Para superar esta desventaja, las CPU usan contadores síncronos, en los que todos los biestables del contador cambian al mismo tiempo. En esta subsección, presentamos el diseño de un contador síncrono de 3 bits. Al mismo tiempo, ilustraremos algunos conceptos básicos en el diseño de circuitos síncronos.

Para un contador de 3 bits, necesitamos tres biestables. Vamos a usar biestables J-K. Llamemos a las salidas sin complementar de los tres biestables A, B y C, respectivamente, donde C representa el bit menos significativo. El primer paso es construir la tabla verdad que relaciona las entradas J-K con las salidas, para poder diseñar el resto del circuito. Dicha tabla verdad se muestra en la Figura A.33a. Las primeras tres columnas muestran las combinaciones posibles de las salidas A, B y C. Están listadas en el orden en que aparecen cuando se incrementa el contador. Cada fila indica el valor actual de A, B, C y las entradas a los tres biestables que se necesitarán para obtener el próximo valor de A, B y C.

Para comprender la forma en la que se ha hecho la tabla verdad de la Figura A.33a, puede ser útil reestructurar la tabla característica del biestable J-K. Recordemos que esta tabla verdad se presentaba como sigue:

| J | K | $Q_{n+1}$       |
|---|---|-----------------|
| 0 | 0 | $Q_n$           |
| 0 | 1 | 0               |
| 1 | 0 | 1               |
| 1 | 1 | $\bar{Q}_{n+1}$ |

En esta forma, la tabla muestra el efecto que las entradas J y K tienen en la salida. Ahora consideraremos la siguiente reorganización de la misma información:

| $Q_n$ | J | K | $Q_{n+1}$ |
|-------|---|---|-----------|
| 0     | 0 | d | 0         |
| 0     | 1 | d | 1         |
| 1     | d | 1 | 0         |
| 1     | d | 0 | 1         |

En esta forma, la tabla proporciona el valor de la siguiente salida cuando se conocen las entradas y la salida actual. Esta es exactamente la información que se necesita para diseñar el contador o, de hecho, cualquier circuito secuencial. En esta forma, la tabla se denomina *tabla de excitaciones*.

Volvamos a la Figura A.33a. Consideremos la primera fila. Queremos que el valor de A permanezca a 0, que el valor de B permanezca a 0, y que el valor de C cambie de 0 a 1, con la siguiente aplicación del pulso de reloj. La tabla de excitaciones muestra que, para mantener

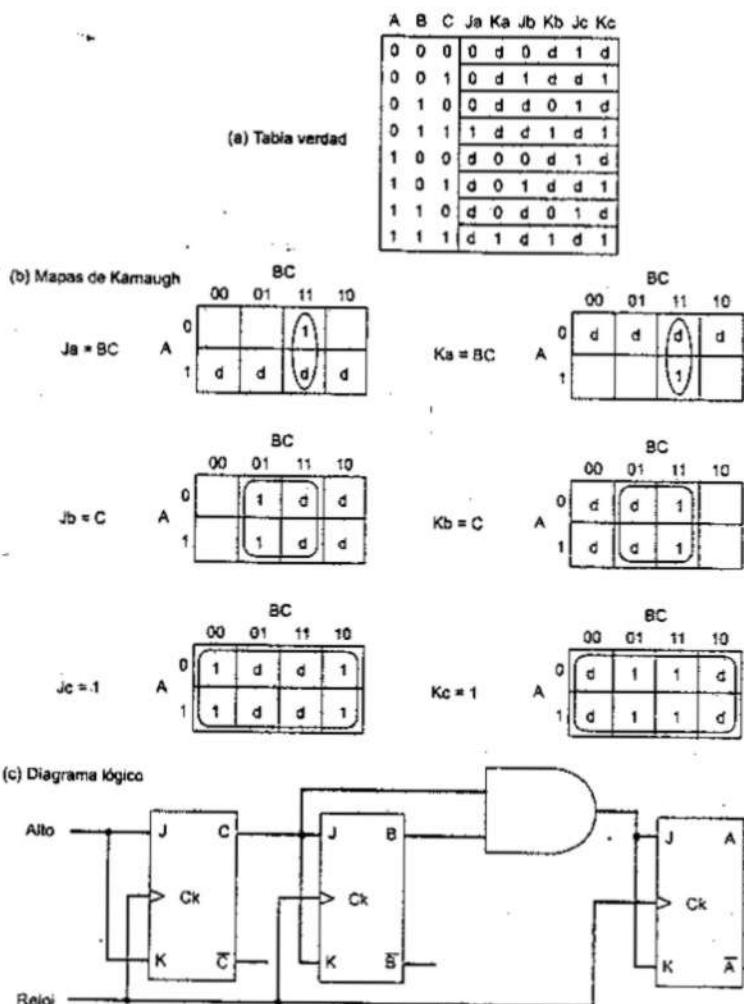


Figura A.33. Diseño de un contador sincrónico.

una salida a 0, debemos tener las entradas  $J = 0$  e indiferencia para  $K$ . Para que haya una transición de 0 a 1, las entradas deben ser  $J = 1$  y  $K = d$ . Estos valores se encuentran en la primera fila de la tabla. Con un razonamiento similar, se puede llenar el resto de la tabla.

Una vez construida la tabla verdad de la Figura A.33a, vemos que la tabla muestra los valores requeridos para todas las entradas  $J$  y  $K$  como funciones de los valores actuales de  $A$ ,  $B$  y  $C$ . Con la ayuda de mapas de Karnaugh, podemos obtener las expresiones booleanas para estas seis funciones. Esto se muestra en la parte (b) de la figura. Por ejemplo, el mapa de

Karnaugh de la variable Ja (la entrada J del biestable que produce la salida A), da lugar a la expresión  $Ja = BC$ . Cuando se derivan las seis expresiones, es un problema sencillo diseñar el circuito real, como se muestra en la parte c de la figura.

### APLICACIONES

A.1. Construir la tabla verdad de las siguientes expresiones booleanas:

- $ABC + \overline{ABC}$
- $ABC + \overline{ABC} + \overline{ABC}$
- $A(\overline{B}C + \overline{B}C)$
- $(A + B)(A + C)(\overline{A} + \overline{B})$

A.2. Simplificar las siguientes expresiones aplicando la ley conmutativa:

- $A \cdot \overline{B} + \overline{B} \cdot A + C \cdot D \cdot E + \overline{C} \cdot D \cdot E + E \cdot \overline{C} \cdot D$
- $A \cdot B + A \cdot C + B \cdot A$
- $(L \cdot M \cdot N)(A \cdot B)(C \cdot D \cdot E)(M \cdot N \cdot L)$
- $F \cdot (K + R) + S \cdot V + W \cdot \overline{X} + V \cdot S + \overline{X} \cdot W + (R + K) \cdot F$

A.3. Aplicar el teorema de DeMorgan a las siguientes ecuaciones:

- $F = \overline{V + A + L}$
- $F = \overline{A} + \overline{B} + \overline{C} + \overline{D}$

A.4. Simplificar las siguientes expresiones:

- $A = S \cdot T + V \cdot W + R \cdot S \cdot T$
- $A = T \cdot U \cdot V + X \cdot Y + Y$
- $A = F \cdot (E + F + G)$
- $A = (P \cdot Q + R + S \cdot T)T \cdot S$
- $A = \overline{\overline{D} \cdot \overline{D} \cdot E}$
- $A = Y \cdot (W + X + \overline{Y} + \overline{Z}) \cdot Z$
- $A = (B \cdot E + C + F) \cdot C$

A.5. Obtener la operación XOR a partir de las operaciones booleanas básicas AND, NOR y NOT.

A.6. Dada una puerta NOR y varias NOT, dibujar un diagrama lógico que realice una función AND de tres entradas.

A.7. Escribir la expresión Booleana de una puerta NAND de cuatro entradas.

A.8. Se usa un circuito combinacional para controlar un visualizador de dígitos decimales de 7 segmentos, como se muestra en la Figura A.34. El circuito tiene cuatro entradas, que proporcionan el código de 4 bits usado en representación decimal compacta ( $0_{10} = 0000, \dots, 9_{10} = 1001$ ). Las siete salidas definen qué segmentos se van a activar

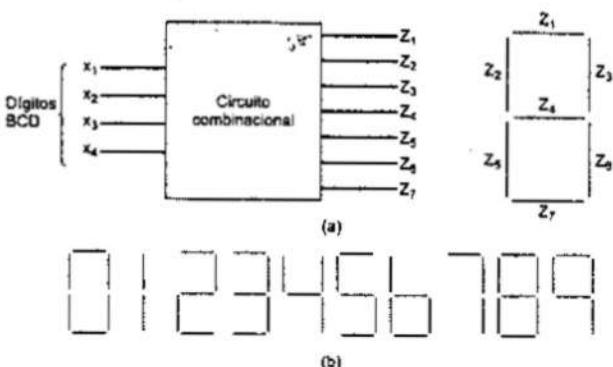


Figura A.34. Ejemplo de un visualizador LED de siete segmentos.

para visualizar un dígito decimal dado. Nótese que no se necesitan algunas combinaciones de entrada ni sus salidas.

- Desarrollar la tabla verdad para este circuito.
- Expresar la tabla verdad en la forma SOP.
- Expresar la tabla verdad en la forma POS.

A.9. Diseñar un multiplexor de 8 a 1.

- A.10. Añadir una línea adicional a la Figura A.15 para que funcione como un demultiplexor.
- A.11. El código Gray es un código binario para enteros. Difiere de la representación binaria ordinaria en que sólo cambia un único bit entre la representación de dos números cualesquiera. Esto es útil en aplicaciones como contadores o conversores analógico-digitales, donde se genera una secuencia de números. Como sólo cambia un bit a la vez, no hay nunca ambigüedad debido a las ligeras diferencias de tiempo. Los primeros ocho elementos del código son:

| Código binario | Código Gray |
|----------------|-------------|
| 000            | 000         |
| 001            | 001         |
| 010            | 011         |
| 011            | 010         |
| 100            | 110         |
| 101            | 111         |
| 110            | 101         |
| 111            | 100         |

Diseñar un circuito que convierta un código binario en un código Gray.

- A.12. Diseñar un decodificador de  $5 \times 32$  usando cuatro decodificadores de  $3 \times 8$  (con entradas de habilitación) y un decodificador de  $2 \times 4$ .
- A.13. Implementar el sumador completo de la Figura A.22 con sólo cinco puertas (*Ayuda:* algunas de las puertas son XOR).
- A.14. Considerar la Figura A.22. Suponer que cada puerta produce un retardo de 10 ns. Entonces, la salida de suma es válida tras 30 ns, y la salida de acarreo tras 0 ns. ¿Cuál es el tiempo total de suma de un sumador de 32 bits
  - a) implementado sin acarreo anticipado, como en la Figura A.21?
  - b) implementado con acarreo anticipado y usando sumadores de 8 bits, como en la Figura A.23?

## **APÉNDICE B**

---

# **Proyectos para enseñar arquitectura y organización de computadores**

B.1. Proyectos de investigación

B.2. Proyectos de simulación

B.3. Lecturas/informes

Muchos profesores creen que la investigación o la implementación de proyectos son cruciales para entender claramente los conceptos de la arquitectura y organización de computadores. Sin proyectos, puede ser difícil para los estudiantes comprender algunos conceptos e interacciones básicos entre componentes. Los proyectos refuerzan los conceptos introducidos en el libro, dan a los estudiantes una mejor apreciación del funcionamiento interno del procesador, y pueden motivar a los estudiantes y darles confianza en que dominan el material.

En este texto se ha tratado de presentar los conceptos lo más claramente posible, y de proporcionar más de 200 ejercicios para reforzar estos conceptos. Muchos profesores desearían complementar este material con proyectos. Este apéndice constituye una guía en este sentido, y describe el material de ayuda disponible en el manual del profesor. El material de ayuda cubre tres tipos de proyectos:

- Proyectos de investigación
- Proyectos de simulación
- Lecturas/informes

#### PROYECTOS DE INVESTIGACIÓN

Una forma efectiva de reforzar los conceptos básicos del curso y enseñar a los estudiantes habilidades para investigar, es asignar proyectos de investigación. Un proyecto de este tipo puede implicar búsqueda de literatura así como productos en venta en la Web, tareas de investigación en laboratorio, y esfuerzos de normalización. Los proyectos podrían asignarse a grupos de alumnos, y los pequeños proyectos a individuos. En cualquier caso, es mejor exigir algún tipo de propuesta de proyecto al principio, dando tiempo al profesor para evaluarla, y para ajustar el tema y el nivel de esfuerzo. El anuncio a los estudiantes de los proyectos de investigación debe incluir:

- Un formato para la propuesta
- Un formato para el informe final
- Un esquema con las fechas tope intermedias y finales
- Una lista de posibles temas de proyectos

Los estudiantes pueden seleccionar uno de los temas de la lista o idear su propio proyecto. El manual del profesor incluye un posible formato para la propuesta y el informe final, así como una lista de posibles temas de investigación.

#### PROYECTOS DE SIMULACIÓN

Una forma excelente de conseguir una comprensión del comportamiento interno de un procesador, y de estudiar y apreciar algunos de los compromisos de diseño e implicaciones de las prestaciones, es simulando los elementos clave del procesador. Una herramienta muy útil para este propósito es SimpleScalar.

Comparado con la implementación real del hardware, la simulación proporciona ventajas, tanto en investigación como en educación:

- Con la simulación, es fácil modificar varios elementos de una organización, variar las características de funcionamiento de varios componentes y, luego, analizar los efectos de dicha modificación.
- La simulación posibilita la obtención de una serie de estadísticas de funcionamiento detallado, que se puede usar para comprender los compromisos de funcionamiento.

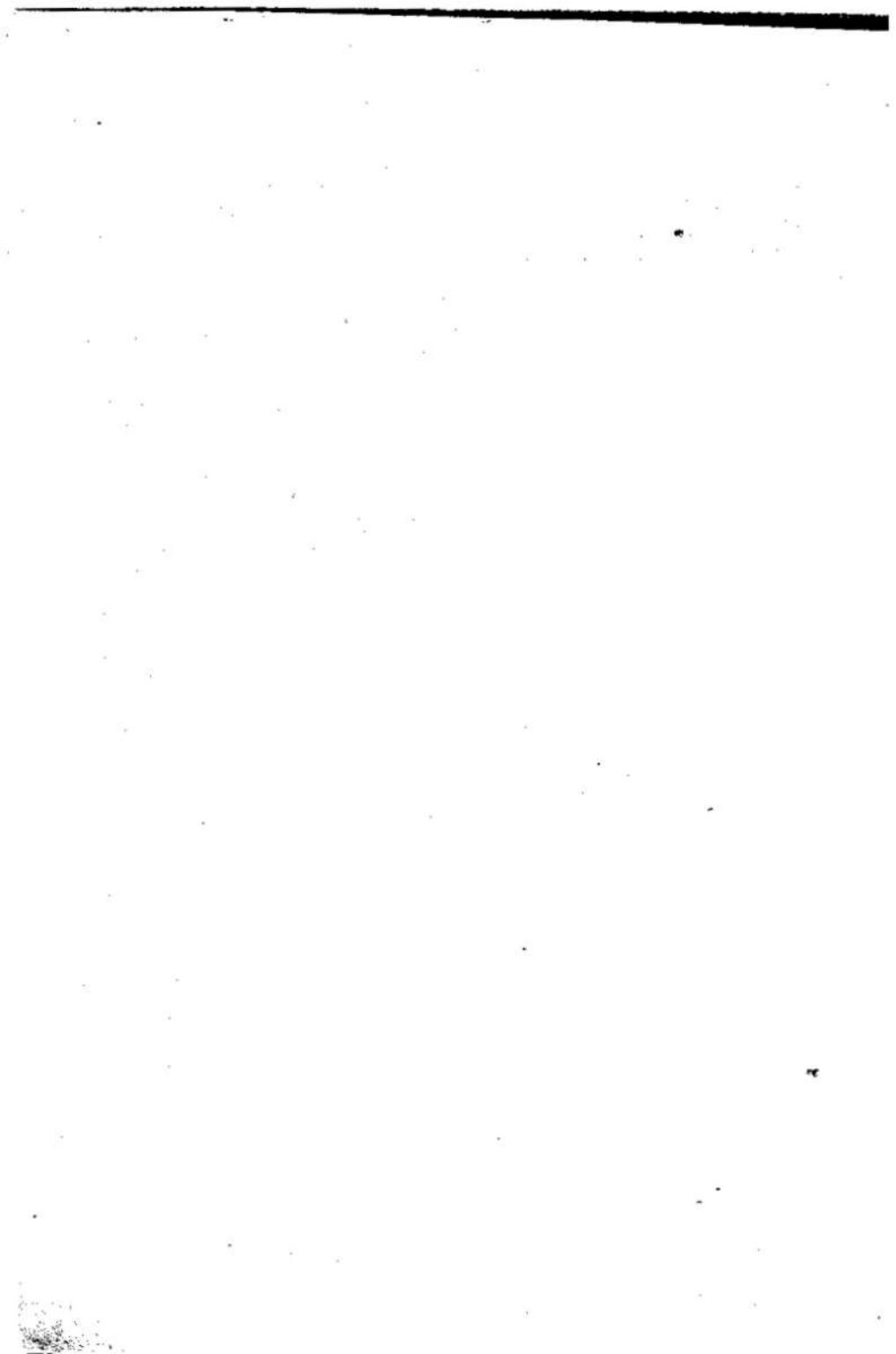
SimpleScalar [BURG97] es un conjunto de herramientas para simular programas reales dentro de un gran rango de procesadores y sistemas modernos. La herramienta incluye un compilador, un ensamblador, un enlazador, y otras herramientas de simulación y visualización. SimpleScalar suministra simuladores de procesadores con una gama que va desde un simulador funcional extremadamente rápido, hasta un simulador de procesador superescalar, con emisión fuera de orden, que soporta cachés no bloqueantes y ejecución especulativa. La arquitectura del conjunto de instrucciones y los parámetros de organización se pueden modificar para crear distintos experimentos.

El manual del profesor para este libro incluye una introducción concisa a SimpleScalar para estudiantes, con instrucciones de cómo cargar e iniciar SimpleScalar. El manual también sugiere algunos proyectos.

SimpleScalar es un paquete software transportable, que vale para Windows NT y UNIX. El software SimpleScalar se puede cargar desde el sitio Web SimpleScalar. Está disponible gratis para uso no comercial.

#### **8.3.3. LECTURAS/INFORMES**

Otra forma excelente para reforzar los conceptos del curso y dar a los estudiantes experiencia en investigación es asignarles artículos para leer y analizar. El manual del profesor incluye una lista de artículos recomendados, uno o dos por capítulo. Todos los artículos están disponibles a través de Internet o en la biblioteca técnica de alguna facultad. El manual también incluye la terminología recomendada.



## **APÉNDICE C**

---

# **Pentium III**

Julio Ortega  
Alberto Prieto

*Departamento de Arquitectura y Tecnología de Computadores  
Universidad de Granada*

C.1. Incidencia de las instrucciones SIMD en las aplicaciones

C.2. El repertorio de instrucciones SSE

C.3. Interconexión del Pentium III a su entorno

C.4. Lecturas y sitios Web recomendados

**E**l microprocesador Pentium III de Intel se comercializó a principios de 1999. Los microprocesadores previos de Intel seguían aproximadamente la ley de Moore, que establece que la velocidad de los procesadores se dobla cada 18 meses. Las primeras versiones del Pentium III no doblaban la velocidad de sus predecesores (Pentium II y Pentium II Xeon), ya que utilizaban frecuencias de reloj de 450 a 550 MHz, mientras que estos últimos funcionaban a frecuencias de 333 a 400 MHz. Además del incremento de velocidad (relativamente pequeño), el Pentium III ofrece algunas características funcionales novedosas.

Básicamente, el Pentium III es un Pentium II, con una arquitectura totalmente compatible con la arquitectura de Intel IA-32 (Intel Architecture-32), y con los elementos descritos en el Capítulo 13 para posibilitar la ejecución dinámica superescalar de instrucciones. Además, la mejora de la tecnología de integración utilizada ha permitido que el Pentium III funcione a mayores frecuencias de reloj, y se han incluido unas 70 nuevas instrucciones, que se agrupan bajo la denominación SSE (Streaming SIMD Extensions). El conjunto SSE hace posible una mejora notable de prestaciones del procesador en aplicaciones avanzadas de imágenes, audio, video 3D, y reconocimiento del habla.

También, el Pentium III incluye el número de serie del procesador (processor serial number), un código de identificación que permite a las aplicaciones y a los entornos de red identificar a la CPU que incluye el microcomputador. El objetivo de este número es aumentar la seguridad de funcionamiento de los sistemas actuales, ampliamente interconectados a través de redes, y está siendo objeto de cierta controversia, ya que muchos consideran que vulnera la privacidad. No obstante, Intel asegura que el número de serie del procesador se puede desactivar a través de la BIOS de la placa base del computador, o bien utilizando la utilidad de control del número de serie del procesador proporcionada por Intel.

A continuación, en la Sección C.1, se analizan las características fundamentales de las aplicaciones que en un futuro próximo constituirán presumiblemente el mayor volumen de las cargas de trabajo, y la forma en que un repertorio SIMD puede contribuir a mejorar las prestaciones de un microprocesador. Despues, en la Sección C.2, se describen las características del repertorio SSE. Finalmente, la Sección C.3 incluye algunas cuestiones relativas a la interconexión del Pentium III con su entorno, las cachés, etc.

#### 14. INCIDENCIA DE LAS INSTRUCCIONES SIMD EN LAS APLICACIONES

Las instrucciones del conjunto SSE, anteriormente denominado *Katmai*, son instrucciones de tipo SIMD (Single Instruction Multiple Data). Tal y como se indicó en el Capítulo 16, una instrucción SIMD codifica una operación que se aplica al mismo tiempo a varios operandos distintos. De esta forma, se puede conseguir una reducción importante del número de instrucciones a ejecutar en determinadas aplicaciones.

Para ilustrar el concepto anteriormente indicado, considérese un procesador en el que sólo existan instrucciones de tipo SISD (cada instrucción codifica una operación que se realiza una sola vez con el operando o los operandos correspondientes). En este caso, el bucle:

$$\text{for } i = 1 \text{ to } 4 \text{ do } a[i] := a[i] + b[i],$$

al traducirse al lenguaje máquina del procesador, da lugar a que éste ejecute cuatro instrucciones de suma, una para cada pareja de operandos  $a[i]$ ,  $b[i]$ ,  $i = 1, 2, 3, 4$ , más cuatro instrucciones de comparación y salto condicional para controlar el final del bucle. En cambio, si se dispone de instrucciones de tipo SIMD que, por ejemplo, codifiquen una operación (en este caso la suma) que se repite simultáneamente varias veces (en este caso cuatro) con los corres-

pondientes operandos, el bucle anterior daría lugar únicamente a una instrucción SIMD. Se produce de esta forma una importante mejora en las prestaciones del procesador, ya que por una parte se reduce el número de instrucciones a procesar, y por otra se evitan instrucciones de salto, que, como se ha visto en el Capítulo 11, pueden dar lugar a cierta penalización en las arquitecturas con segmentación de cauce, como es el caso de los microprocesadores superscalares actuales.

Por lo tanto, en aplicaciones en las que los códigos estén estructurados en términos de bucles similares al considerado en el ejemplo, se pueden obtener mejoras importantes en el tiempo de ejecución al utilizar instrucciones del tipo SIMD. Precisamente, las *aplicaciones multimedia* presentan estas características, y parece ser que a ellas van a corresponder cada vez mayores porcentajes de las cargas de trabajo de los microprocesadores a medida que las tecnologías multimedia dinámicas (videoconferencia, procesamiento y compresión de imágenes, animación, gráficos 3D, reconocimiento del habla, encriptado de datos, comunicaciones en banda ancha, etc.) se vayan introduciendo con mayor frecuencia en las aplicaciones.

A diferencia de otras, las aplicaciones multimedia implican un procesamiento en tiempo real de secuencias continuas de datos organizados como vectores de enteros de 8, 16 y 32 bits, y números en coma flotante. Además, la evaluación del resultado de una aplicación multimedia es más una cuestión de percepción cualitativa en tiempo real que una comparación cuantitativa del valor numérico obtenido, a diferencia de lo que ocurre con la mayoría de las aplicaciones científicas. Esto tiene una consecuencia importante en el aprovechamiento de los recursos de un computador para este tipo de aplicaciones. Teniendo en cuenta el rango de discriminación de los sentidos humanos, 8 ó 16 bits pueden ser suficientes para los tipos de datos enteros utilizados en estas aplicaciones. En consecuencia, en estos casos el ancho del camino de datos (32 ó 64 bits) de la mayoría de los microprocesadores actuales resulta sobredimensionado.

En las aplicaciones multimedia es usual utilizar varios hilos (threads) dedicados a distintas tareas (imagen, sonido, etc.) que se puedan procesar de forma prácticamente independiente. Esto facilita el aprovechamiento eficaz del paralelismo funcional de grano grueso por parte de los procesadores. Por otra parte, en las aplicaciones de gráficos y procesamiento de señales existe un paralelismo de datos inherente, ya que las secuencias de datos a procesar están constituidas por grandes cantidades de datos uniformes, tales como puntos de imagen (pixels), vértices, valores de frecuencia o amplitud, etc., que se procesan todos de forma idéntica. Esto permite aprovechar eficazmente el procesamiento SIMD, con las consiguientes ventajas en cuanto a la reducción de los riesgos de control y a la posibilidad de mejorar el ancho de banda de acceso a la memoria principal. Hay que tener en cuenta que, dado el gran volumen de datos que involucran estas aplicaciones, los procesadores deben disponer de un ancho de banda elevado, y ser capaces de tolerar los retardos de memoria elevados. Por otra parte, las prestaciones de las cachés se degradan debido a la pobre localidad temporal de los datos, pero, por el contrario, se logra un gran beneficio de otras técnicas, como la precaptación de datos y de los esquemas que evitan el uso de la cache de datos (cache bypass).

Los bucles de pequeño tamaño consumen la mayoría del tiempo de procesamiento en las aplicaciones de procesamiento de señales e imágenes. Esto significa que estas aplicaciones presentan una gran localidad espacial y temporal en el acceso a las instrucciones, y que existe una gran correlación entre la reducción de tiempo que se consigue en la ejecución de estos bucles y la reducción en la aplicación global. De ahí los beneficios que se derivan de utilizar instrucciones SIMD para aumentar la velocidad de ejecución de los bucles, tal y como se indicó con anterioridad.

Puede afirmarse que las aplicaciones multimedia han pasado a ser una de las consideraciones determinantes en el diseño de los microprocesadores y de la arquitectura del computador. Prueba de ello es que prácticamente todos los fabricantes han propuesto la inclusión de repertorios de instrucciones SIMD específicos, y un soporte adecuado para los mismos por parte de la arquitectura del procesador, consiguiéndose así mejorar notablemente el rendimiento de estos microprocesadores cuando ejecutan aplicaciones multimedia. Entre estos repertorios de instrucciones cabe destacar el VIS para el SPARC de Sun Microsystems, el MDMX para el MIPS V de Silicon Graphics, el MVI para el Alpha de Digital, el MAX2 para el PA-RISC de Hewlett-Packard, el 3D Now! para el Athlon de AMD, y el MMX y el SSE de Intel para el Pentium. En la siguiente sección se comparan estos dos últimos repertorios de instrucciones SIMD.

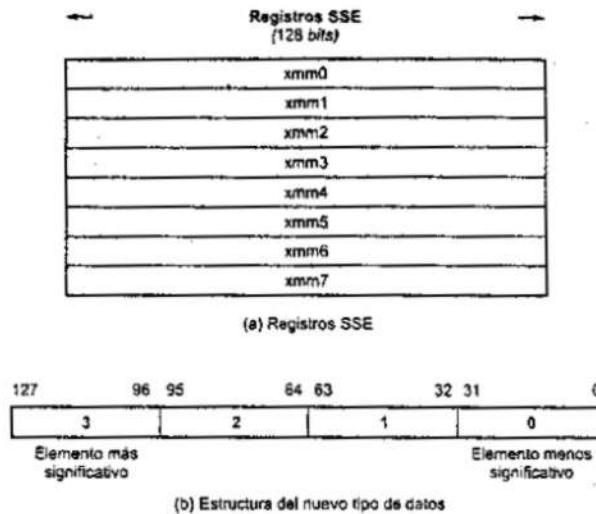
### El repertorio de instrucciones SSE

Las instrucciones SSE que se introducen en el Pentium III son instrucciones SIMD, igual que las instrucciones del conjunto MMX que se añadieron a la arquitectura inicial del Pentium en 1996. Una que resaltar que la implementación de las instrucciones SIMD en los repertorios MMX y SSE es distinta de las de las instrucciones SIMD propias de los procesadores vectoriales, estudiados en el Capítulo 16. En el caso de una instrucción SIMD de un procesador vectorial, el procesamiento de los operandos vectoriales se realiza mediante una unidad funcional encuadrada en la que se van introduciendo los componentes de los vectores a procesar. Se aprovecha la segmentación de cauce para reducir el tiempo de ejecución, ya que los componentes de los operandos vectoriales se procesan concurrentemente. En las instrucciones SIMD de los repertorios MMX y SSE, los componentes de los operandos se procesan simultáneamente, utilizando varias unidades funcionales que implementan la misma operación sincronamente. Se corresponde al modelo propio de un procesador matricial.

A parte de por el conjunto de instrucciones que incluyen, los repertorios MMX y SSE se diferencian entre sí por los tipos de datos que utilizan y por los recursos que el procesador pone a disposición de las instrucciones de cada repertorio. Para las instrucciones MMX no se incluyen registros adicionales en la arquitectura del procesador. Los operandos de las instrucciones MMX se almacenan en los 8 registros para datos en coma flotante. Cada uno de estos registros tiene 80 bits, de los cuales, los 64 menos significativos se utilizan para las ins-



Figura C.1. Registros de coma flotante y MMX.



**Figura C.2.** Registros y distribución en el nuevo tipo de datos SSE.

trucciones MMX, y reciben el nombre *mm0.....mm7* (Figura C.1). Esto da lugar a una limitación importante, ya que, de este modo, las aplicaciones no pueden utilizar simultáneamente instrucciones MMX y operaciones en coma flotante. Además, se necesita un número de ciclos elevado para conmutar desde el modo de ejecución correspondiente a las instrucciones MMX al de operaciones en coma flotante, y viceversa.

Por el contrario, para las instrucciones SSE se han incluido en el procesador ocho nuevos registros, denominados `xmm0`...`xmm7`. Estos registros son de 128 bits, y pueden almacenar

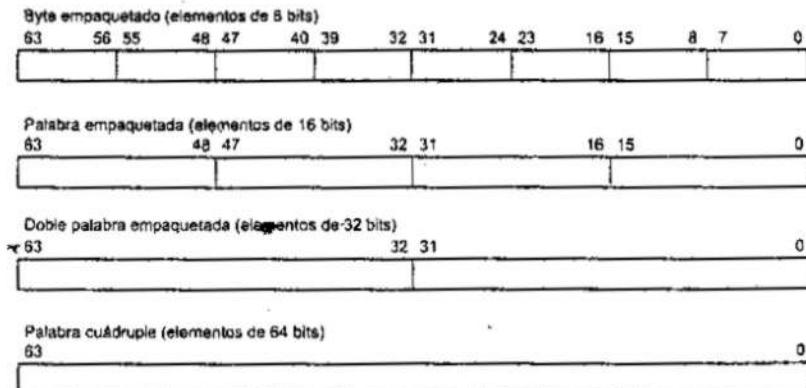


Figura C.3. Tipos de datos incluibles en los registros de 64 bits.

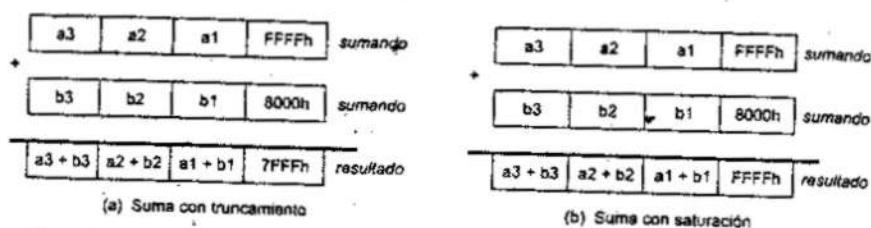


Figura C.4. Ejemplo de la diferencia entre operación con truncamiento y con saturación.

cuatro números de 32 bits en coma flotante, tal y como se ilustra en la Figura C.2. De esta forma, las aplicaciones pueden ejecutar tanto las instrucciones SIMD enteras correspondientes al repertorio MMX, como las instrucciones SIMD en coma flotante del SSE. También se pueden ejecutar simultáneamente las instrucciones con datos en coma flotante de tipo SIMD y las instrucciones con datos en coma flotante que no son del repertorio SSE (instrucciones no SIMD).

En el Capítulo 9 se proporcionan algunos detalles del repertorio MMX. Concretamente, en la Tabla 9.10 aparecen los mnemotécnicos de las 57 instrucciones MMX y se describe el significado de cada instrucción. También se presentan en el Capítulo 9 los tipos de datos de 64 bits utilizados en el repertorio MMX. Como se vio, los tipos son: *byte empaquetado*, constituido por ocho bytes; *palabra empaquetada*, constituida por cuatro palabras de 16 bits; *palabra doble empaquetada*, constituida por 2 palabras de 32 bits; y *palabra cuádruple*, que ocupa los 64 bits del dato empaquetado. La estructura de estos tipos de datos se ilustra en la Figura C.3.

En la Figura C.4 se muestra la diferencia entre los resultados de una instrucción con truncamiento y con saturación, conceptos que se explicaron en el Capítulo 9. Tal y como se indicó, cuando la instrucción utiliza truncamiento se pierde el bit de acarreo si el resultado excede el valor representable con el tipo de datos utilizado (8 en el byte empaquetado, 16 en la palabra empaquetada, etc.). Si se utiliza saturación, en el caso de que se produzca un desbordamiento en la suma o un desbordamiento a cero (underflow) en la resta, el resultado se hace igual, respectivamente, al mayor o al menor valor representable con el número de bits del tipo de dato utilizado. Otro ejemplo de utilización de instrucciones MMX se da en la Figura C.5, donde se ilustra el funcionamiento de la instrucción PMADDWD, que realiza cuatro productos de 32 bits seguidos de dos acumulaciones. Parte de operandos del tipo palabra empaquetada y genera resultados del tipo doble palabra empaquetada.

En el repertorio MMX existen instrucciones de empaquetamiento y desempaquetamiento, que permiten la conversión entre distintos tipos de datos empaquetados. Como ejemplo de

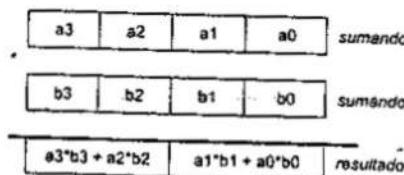


Figura C.5. Efecto de la ejecución de una instrucción PMADDWD.

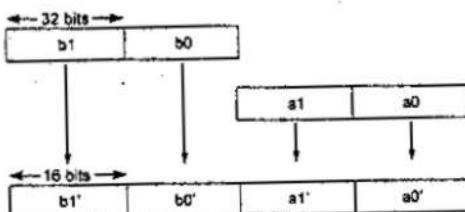


Figura C.6. Efecto de la ejecución de la instrucción de empaquetamiento PACKSSDW.

instrucción de empaquetamiento. en la Figura C.6 se muestra el efecto de la instrucción PACKSSDW. Esta instrucción toma cuatro datos de 32 bits y los transforma en cuatro datos de 16 bits con saturación si alguno de los datos no puede expresarse con 16 bits. Existe una instrucción de desempaquetamiento que realiza la transformación contraria.

## EL REPERTORIO SSE

Las instrucciones SSE pueden operar simultáneamente con cuatro datos de 32 bits en coma flotante. No todas las instrucciones del conjunto SSE son instrucciones SIMD con datos de coma flotante. De las instrucciones SSE, unas 50 son instrucciones SIMD con datos de coma flotante. 12 son SIMD con enteros, y las 8 restantes permiten modificar las opciones de almacenamiento de los datos en cache.

Si se tiene en cuenta la forma de almacenar los datos a procesar por las instrucciones, se distingue entre instrucciones con datos empaquetados (*instrucciones SSE empaquetadas*) e instrucciones con datos escalares (*instrucciones SSE escalares*). Para distinguir entre ambos tipos de instrucciones se utiliza el sufijo 'ps' en las instrucciones empaquetadas, y 'ss' en las escalares. Por otra parte, las instrucciones SSE permiten definir un nuevo tipo de datos, que hace referencia a cuatro números en coma flotante de precisión simple. La Figura C.7 ilustra el uso de instrucciones empaquetadas e instrucciones escalares con este tipo de datos. En la figura,  $a0$ ,  $a1$ ,  $a2$ , y  $a3$  hacen referencia a cuatro números en coma flotante de precisión simple.

La Tabla C.1 proporciona los mnemónicos de las instrucciones del repertorio SSE, agrupadas según los distintos tipos de operaciones que realizan. A continuación, se proporcionan algunos ejemplos de uso de estas instrucciones.

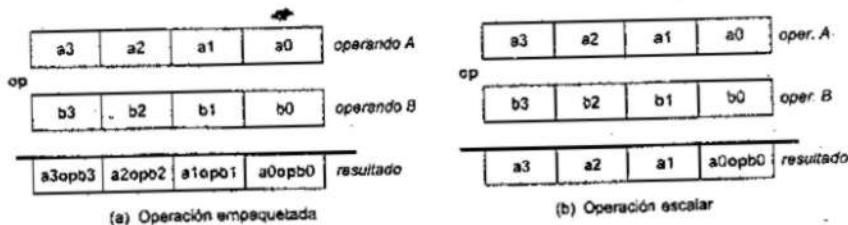


Figura C.7. Diferencias entre instrucciones empaquetadas y escalares.

Tabla C.1. Instrucciones del conjunto SSE

| Nomotécnica        | Codops | Descripción                                                                                                                                                                                                                                                                    |
|--------------------|--------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Aritméticas</b> |        |                                                                                                                                                                                                                                                                                |
| ADD[PS;SS]         | 2      | Sumar [los cuatro valores de los dos operandos; los valores menos significativos de los dos operandos y tomar los restantes tres valores del primero de los operandos] (los datos son números en coma flotante de simple precisión de 32 bits).                                |
| MUL[PS;SS]         | 2      | Multiplicar [los cuatro valores de los dos operandos; los valores menos significativos de los dos operandos y tomar los restantes tres valores del primero de los operandos] (los datos son números en coma flotante de simple precisión de 32 bits).                          |
| DIV[PS;SS]         | 2      | Dividir [los cuatro valores de los dos operandos; los valores menos significativos de los dos operandos y tomar los restantes tres valores del primero de los operandos] (los datos son números en coma flotante de simple precisión de 32 bits).                              |
| SQRT[PS;SS]        | 2      | Raíz cuadrada [los cuatro valores de los dos operandos; los valores menos significativos de los dos operandos y tomar los restantes tres valores del primero de los operandos] (los datos son números en coma flotante de simple precisión de 32 bits).                        |
| MAX[PS;SS]         | 2      | Máximos de [los cuatro valores de los dos operandos; los valores menos significativos de los dos operandos y tomar los restantes tres valores del primero de los operandos] (los datos son números en coma flotante de simple precisión de 32 bits).                           |
| MIN[PS;SS]         | 2      | Mínimos de [los cuatro valores de los dos operandos; los valores menos significativos de los dos operandos y tomar los restantes tres valores del primero de los operandos] (los datos son números en coma flotante de simple precisión de 32 bits).                           |
| 1/P[PS;SS]         | 2      | Obtiene la aproximación de los reciprocos de [los cuatro valores en coma flotante y simple precisión del operando, el valor en coma flotante y simple precisión menos significativo del operando manteniendo los tres valores más significativos iguales].                     |
| 1/SQRT[PS;SS]      | 2      | Obtiene la aproximación de los reciprocos de la raíz cuadrada de [los cuatro valores en coma flotante y simple precisión del operando; el valor en coma flotante y simple precisión menos significativo del operando manteniendo los tres valores más significativos iguales]. |
| <b>lógicas</b>     |        |                                                                                                                                                                                                                                                                                |
| IDPS               | 1      | OR bit a bit de los cuatro valores empaquetados (128 bits) en coma flotante y precisión simple de los dos operandos.                                                                                                                                                           |
| DNPS               | 1      | Invertir los cuatro valores empaquetados (128 bits) del primero de los operandos y realizar la operación AND bit a bit con los cuatro valores empaquetados del segundo.                                                                                                        |
| PS                 | 1      | -OR bit a bit de los cuatro valores empaquetados (128 bits) en coma flotante y precisión simple de los dos operandos.                                                                                                                                                          |
| EPS                | 1      | EXOR (or-exclusiva) bit a bit de los cuatro valores empaquetados (128 bits) en coma flotante y precisión simple de los dos operandos.                                                                                                                                          |

Tabla C.1. Instrucciones del conjunto SSE (continuación)

| Nemotécnico        | Codops. | Descripción                                                                                                                                                                                                                                                                                                               |
|--------------------|---------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>Comparación</i> |         |                                                                                                                                                                                                                                                                                                                           |
| CMP[PS]SS          | 2       | Comparar [los cuatro valores de los dos operandos; los valores menos significativos de los dos operandos y tomar los restantes tres valores del primero de los operandos]. El tipo de comparación se indica en un dato inmediato de 8 bits (0 corresponde a «igual que», 1 a «menor que», 2 a «menor o igual que», etc.). |
| COMISS             | 1       | Comparar el valor en coma flotante y precisión simple menos significativo del dato empaquetado en uno de los operandos con el correspondiente del dato empaquetado en el otro operando, y actualizar los bits de estado correspondientes. Da lugar a una excepción de número no válido cuando el operando es sNaN o qNaN. |
| UCOMISS            | 1       | Igual que COMISS, pero sólo genera una excepción de número no válido cuando el operando es sNaN.                                                                                                                                                                                                                          |
| <i>Baraje</i>      |         |                                                                                                                                                                                                                                                                                                                           |
| SHUFPS             | 1       | Selecciona cuatro valores de coma flotante y simple precisión (32 bits cada uno) de los operandos empaquetados según una máscara que se indica a través de un dato inmediato. Los dos valores menos significativos se toman de uno de los operandos, y los dos más significativos del otro.                               |
| UNPCKHPS           | 1       | Selecciona y entrelaza los dos valores de coma flotante y simple precisión (32 bits cada uno) más significativos de cada uno de los dos operandos. Los dos valores menos significativos de los operandos no se tienen en cuenta.                                                                                          |
| UNPCKLPS           | 1       | Selecciona y entrelaza los dos valores de coma flotante y simple precisión (32 bits cada uno) menos significativos de cada uno de los dos operandos. Los dos valores menos significativos de los operandos no se tienen en cuenta.                                                                                        |
| <i>Conversión</i>  |         |                                                                                                                                                                                                                                                                                                                           |
| CVTPI2PS           | 1       | Convierte los dos valores enteros de 32 bits en forma empaquetada de uno de los operandos en valores de coma flotante y simple precisión. Los dos valores en coma flotante y simple precisión más significativos del dato empaquetado resultante se toman directamente.                                                   |
| CVTPS2PI           | 1       | Convierte los dos valores en coma flotante y simple precisión menos significativos de uno de los operandos en dos enteros de 32 bits, según el modo de redondeo actual.                                                                                                                                                   |
| CVTSI2SS           | 1       | Convierte un entero de 32 bits en un valor en coma flotante de simple precisión; los tres valores en coma flotante y simple precisión más significativos se toman directamente del otro operando (constituido por cuatro valores en coma flotante y simple precisión empaquetados).                                       |
| CVTSS2SI           | 1       | Convierte el valor en coma flotante y simple precisión menos significativo de un operando (constituido por cuatro valores en coma flotante y simple precisión empaquetados) en un entero de 32 bits.                                                                                                                      |
| CVTTPS2PI          | 1       | Convierte los dos valores en coma flotante y simple precisión menos significativos del operando en dos enteros con signo de 32 bits con truncamiento.                                                                                                                                                                     |

Tabla C.1. Instrucciones del conjunto SSE (continuación)

| Nomotécnico                                           | Codops | Descripción                                                                                                                                                                                                                                                                                                                                                                                   |
|-------------------------------------------------------|--------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CVTTSS2SI                                             | 1      | Convierte el valor en coma flotante y simple precisión menos significativo de un operando (constituido por cuatro valores en coma flotante y simple precisión empaquetados) en un entero con signo de 32 bits, según el modo de redondeo con truncamiento. El resultado se introduce en un registro para enteros.                                                                             |
| <i>Transferencia de datos</i>                         |        |                                                                                                                                                                                                                                                                                                                                                                                               |
| MOVAPS                                                | 2      | Carga un registro de datos empaquetados desde memoria con cuatro valores en coma flotante de simple precisión (o viceversa) / Mueve datos empaquetados desde un registro a otro. La dirección de memoria debe estar alineada con las palabras de 16 bits.                                                                                                                                     |
| MOVHLPS                                               | 1      | Transfiere los dos valores en coma flotante de simple precisión más significativos de uno de los operandos a los dos valores menos significativos del resultado. Los dos valores más significativos se toman directamente del otro operando empaquetado (los dos valores más significativos).                                                                                                 |
| MOVHPS                                                | 2      | Mueve 64 bits, que representan dos valores en coma flotante de simple precisión, desde la dirección de memoria indicada en la instrucción a los dos valores más significativos de un registro de datos empaquetados / Mueve los dos valores en coma flotante de simple precisión más significativos de un registro de datos empaquetados a la posición de memoria indicada en la instrucción. |
| MOVLHPS                                               | 1      | Transfiere los dos valores en coma flotante de simple precisión menos significativos de uno de los operandos a los dos valores más significativos del resultado. Los dos valores menos significativos se toman directamente del otro operando empaquetado (los dos valores menos significativos).                                                                                             |
| MOVLPS                                                | 2      | Mueve 64 bits, que representan dos valores en coma flotante de simple precisión, desde la dirección de memoria indicada en la instrucción a los dos valores más significativos de un registro de datos empaquetados / Mueve los dos valores en coma flotante de simple precisión más significativos de un registro de datos empaquetados a la posición de memoria indicada en la instrucción. |
| MOVMSKPS                                              | 2      | Crea una máscara de 4 bits a partir de los bits más significativos de los cuatro valores en coma flotante del operando empaquetado de 128 bits.                                                                                                                                                                                                                                               |
| OVSS                                                  | 2      | Carga un valor en coma flotante de simple precisión en los 32 bits menos significativos de un registro de datos empaquetados, y borra los tres valores en coma flotante de simple precisión más significativos / Almacena en memoria el valor en coma flotante y simple precisión (32 bits) menos significativo de un registro de datos empaquetados (128 bits).                              |
| OVUPS                                                 | 2      | Carga desde memoria, o almacena en memoria, un registro de datos empaquetados (cuatro valores en coma flotante de simple precisión). La dirección de memoria no tiene que estar alineada con las palabras de 16 bits.                                                                                                                                                                         |
| <i>Control de opciones de almacenamiento en cache</i> |        |                                                                                                                                                                                                                                                                                                                                                                                               |
| ISKMOVQ                                               | 1      | Mueve condicionalmente los bytes de uno de los operandos en la dirección indicada. La condición de almacenamiento viene determinada por el bit más significativo de cada byte del operando selector.                                                                                                                                                                                          |

Tabla C.1. Instrucciones del conjunto SSE (continuación)

| Nomotécnico                                       | Codops | Descripción                                                                                                                                                                                                                                                                                                                                           |
|---------------------------------------------------|--------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| MOVNTPS                                           | 1      | Mueve un dato empaquetado (128 bits) a la dirección de memoria indicada, sin modificar la cache.                                                                                                                                                                                                                                                      |
| MOVVNTQ                                           | 1      | Mueve 64 bits, que representan enteros (8 bits, 16 bits, 32 bits), desde un registro a la dirección de memoria indicada, sin modificar la cache.                                                                                                                                                                                                      |
| PREFETCHT0                                        | 1      | Precarga una línea en las caches de todos los niveles, con datos tomados a partir de la dirección de memoria indicada.                                                                                                                                                                                                                                |
| PREFETCHT1                                        | 1      | Precarga una línea en las caches de todos los niveles, excepto el nivel 0, con datos tomados a partir de la dirección de memoria indicada.                                                                                                                                                                                                            |
| PREFETCHT2                                        | 1      | Precarga una línea en las caches de todos los niveles, excepto en los niveles 0 y 1, con datos tomados a partir de la dirección de memoria indicada.                                                                                                                                                                                                  |
| PREFETCHNTA                                       | 1      | Precarga una línea en la cache del nivel 1 (L1) con datos tomados a partir de la dirección de memoria indicada.                                                                                                                                                                                                                                       |
| SFENCE                                            | 1      | Garantiza que todo almacenamiento previo sea visible antes de cualquier operación de almacenamiento posterior.                                                                                                                                                                                                                                        |
| <i>Gestión del estado</i>                         |        |                                                                                                                                                                                                                                                                                                                                                       |
| FXRSTOR                                           | 1      | Recupera el estado correspondiente a las operaciones en coma flotante, MMX y SSE, desde una posición de memoria indicada en la instrucción. Este estado debe haberse almacenado previamente en esa posición mediante FXSAVE.                                                                                                                          |
| FXSAVE                                            | 1      | Almacena el estado correspondiente a las operaciones en coma flotante, MMX y SSE, a partir de una posición de memoria indicada en la instrucción.                                                                                                                                                                                                     |
| LDMXCSR                                           | 1      | Carga el registro de 32 bits de control/estado correspondiente a las instrucciones SSE (registro MXCSR). El registro de control/estado MXCSR se utiliza para acceder a los bits de estado de las excepciones, activar las excepciones enmascaradas/no enmascaradas, fijar los modos de redondeo, y activar el modo de borrado a cero (flush-to-zero). |
| STMXSCR                                           | 1      | Almacena el registro de 32 bits de control/estado correspondiente a las instrucciones SSE (registro MXCSR). Los bits reservados se almacenan como ceros.                                                                                                                                                                                              |
| <i>Instrucciones SIMD adicionales con enteros</i> |        |                                                                                                                                                                                                                                                                                                                                                       |
| PAVG[B;W]                                         | 2      | Obtiene un dato empaquetado con las medias de los [bytes; palabras] de los operandos empaquetados.                                                                                                                                                                                                                                                    |
| PEXTRW                                            | 1      | Selecciona una de las palabras del dato empaquetado y la almacena en la mitad menos significativa de un registro de 32 bits. La selección se indica mediante los dos bits menos significativos de un dato inmediato.                                                                                                                                  |
| PINSRW                                            | 1      | Inserta la palabra menos significativa de un registro de 32 bits en un dato empaquetado. La selección de la palabra en el dato se hace mediante los dos bits menos significativos de un dato inmediato. El resto de las palabras del dato empaquetado permanecen inalteradas.                                                                         |
| PMAX(SW;UB)                                       | 2      | Dados dos operandos empaquetados, determina el máximo [de las palabras de dichos operandos, de los bytes sin signo de dichos operandos].                                                                                                                                                                                                              |

Tabla C.1. Instrucciones del conjunto SSE (continuación)

| Nemotécnico | Codops | Descripción                                                                                                                                                                                                                             |
|-------------|--------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| PMIN(SW;UB) | 2      | Dados dos operandos empaquetados, determina el máximo [de las palabras de dichos operandos; de los bytes sin signo de dichos operandos].                                                                                                |
| PMOVMSKB    | 1      | Crea una máscara a partir de los bits más significativos de los bytes de un dato empaquetado.                                                                                                                                           |
| PMULHUW     | 1      | Multiplica las palabras sin signo de dos operandos empaquetados, proporcionando los 16 bits más significativos de los resultados intermedios de 32 bits.                                                                                |
| PSADBW      | 1      | Calcula el valor absoluto de las diferencias de los bytes sin signo de los operandos. Estas diferencias se suman y se almacenan en los 16 bits menos significativos del resultado. Los restantes bits del resultado se hacen igual a 0. |
| PSHUFW      | 1      | Proporciona una combinación de las cuatro palabras de un dato empaquetado. La selección se indica mediante un dato inmediato.                                                                                                           |

*Ejemplo 1. Multiplicación empaquetada.*

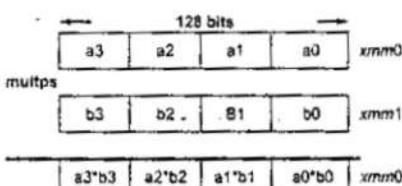
Se pueden realizar cuatro multiplicaciones simultáneas si se almacenan los cuatro pares de números a multiplicar en dos registros específicos para las instrucciones SSE y luego se utiliza la instrucción *mulps*. Por ejemplo, si en el registro de 128 bits *xmm0* se almacenan los cuatro números en coma flotante de 32 bits  $a0 = 8.0, a1 = 4.0, a2 = 2.0, a3 = 1.0$ , y en el registro *xmm1* se almacenan  $b0 = 4.0, b1 = 3.0, b2 = 2.0, b3 = 1.0$ , al ejecutar

*mulps xmm0, xmm1*

se obtendrán en el registro *xmm0* los productos  $c0 = a0 * b0 = 32.0, c1 = a1 * b1 = 12.0, c2 = a2 * b2 = 4.0, c3 = a3 * b3 = 1.0$  tal y como indica la Figura C.8.

*Ejemplo 2. Operación de comparación.*

Es posible comparar varios números en coma flotante utilizando una única instrucción del repertorio SSE. Por ejemplo, la instrucción *cmpps* permite comparar cuatro pares de números de 32 bits en coma flotante empaquetados, almacenados en dos registros SSE de 128 bits. La condición de comparación («mayor que», «mayor o igual que», «menor que», «menor o igual que», etc.) se codifica mediante un número, que constituye el tercer operando de la instrucción. Así por ejemplo, el programa:

Figura C.8. Efecto de la ejecución de una instrucción *mulps*.

```

float a[4],b[4],c[4];
integer i;
for (i = 0;i<4;i++)
{
 if (a[i]< b[i]) then c[i] = a[i] else c[i] = b[i];
}

```

se puede sustituir por la siguiente secuencia de instrucciones SSE, asumiendo que el registro *xmm1* contiene los componentes de *a*, y el *xmm2* los componentes de *b*:

|                           |                                                                                                           |
|---------------------------|-----------------------------------------------------------------------------------------------------------|
| <i>movaps xmm0,xmm1</i>   | :Copiar <i>xmm1</i> en <i>xmm0</i>                                                                        |
| <i>cmpps xmm0,xmm2, 1</i> | :Comprobar si los operandos en <i>xmm0</i><br>son menores que los operandos en <i>xmm2</i>                |
| <i>movaps xmm3,xmm0</i>   | :Copiar el resultado de la comparación en <i>xmm3</i>                                                     |
| <i>andps xmm0,xmm1</i>    | : <i>AND</i> bit a bit de <i>xmm0</i> y <i>xmm1</i> . Resultado en <i>xmm0</i>                            |
| <i>andnps xmm3,xmm2</i>   | : <i>AND</i> bit a bit de <i>xmm3</i> complementado bit a bit<br>y <i>xmm2</i> . Resultado en <i>xmm3</i> |
| <i>orps xmm0,xmm3</i>     | : <i>OR</i> de <i>xmm0</i> y <i>xmm3</i> . El resultado está en <i>xmm0</i>                               |

Para entender el efecto de la secuencia de instrucciones SSE anterior es fundamental tener en cuenta el funcionamiento de las instrucción de comparación, *cmpps*. El tercer operando de esta instrucción (igual a 1) indica el tipo de opción de comparación («menor que», en este caso). La comparación se realiza entre cada uno de los cuatro operandos de coma flotante de 32 bits componentes de *xmm0* y de *xmm1*. El resultado de la comparación se guarda en el registro que aparece como primer operando (*xmm0*, en este caso), de forma que se hacen igual a 1 los 32 bits correspondientes al componente del registro para el que se cumple la condición de comparación, y a 0 en caso contrario. El comportamiento de esta instrucción de comparación se ilustra en la Figura C.9.

Este último ejemplo muestra cómo el uso de las instrucciones SIMD del repertorio SSE no sólo tiene el efecto de reducir el número de instrucciones a ejecutar para completar una serie de operaciones sobre un conjunto de operandos, sino que además permite reducir el número de instrucciones de salto condicional, evitando riesgos de control que disminuyen el rendimiento del cauce del procesador.

El número de instrucciones a ejecutar para realizar una operación con un conjunto de N datos se reduce a N/4 si se utilizan instrucciones SSE, puesto que cada instrucción SSE codifica una operación con cuatro datos. No obstante, hay que tener en cuenta que el número de instrucciones cuando se utiliza el repertorio SSE no es exactamente la cuarta parte del número de instrucciones no SIMD que se utilizarían, sino algo mayor. Esta situación se debe a que se necesitan ciertas instrucciones adicionales para situar los datos, de forma que puedan ser utilizados por las instrucciones SIMD. Además, no siempre se pueden expresar todas las operaciones a realizar mediante instrucciones SSE actuando con cuatro operandos.

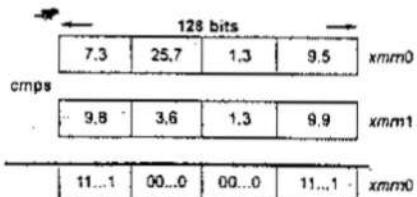


Figura C.9. Efecto de la ejecución de una instrucción *cmpps* («menor que»).

La conversión de los datos al formato correspondiente al tipo de datos que necesitan las instrucciones SSE recibe el nombre de *mezcla de datos* (data swizzling). Esta conversión consume ciclos máquina y, por lo tanto, si el número de conversiones que se necesita es muy elevado, el tiempo que se invierte en ellas puede contrarrestar el tiempo que se gana al utilizar las instrucciones SSE.

### C3. INTERCONEXIÓN DEL PENTIUM III A SU ENTORNO

Actualmente existen distintas versiones del Pentium III que pueden funcionar con frecuencias de reloj que van desde los 450 a los 800 MHz. La CPU tiene elementos que funcionan a 2 V, consumiendo entre 14 y 18 amperios, según la frecuencia, y otros que funcionan a 3,3 V. Incorpora una memoria cache interna L1 de datos de 16 Kbytes, y otra L1 para instrucciones, también de 16 Kbytes.

El Pentium III es un procesador de 32 bits pero, igual que los procesadores Pentium anteriores, tiene un bus externo de 64 bits para comunicarse más eficazmente con el sistema de memoria. Aunque el bus externo de 64 bits permite incrementar el ancho de banda entre el procesador y la memoria, el Pentium III utiliza internamente registros y direcciones de memoria de 32 bits. Puede direccionar, por tanto, hasta 4 Gbytes de memoria principal.

El Pentium III utiliza SLOT1 con encapsulado SECC 2 (Single Edge Contact Cartridge) para mejorar las características de disipación de calor, e incorpora una cache externa L2 para datos e instrucciones de 512 Kbytes con código de corrección de errores (ECC, Error Correcting Code). La cache externa L2 funciona a una frecuencia igual a la mitad de la frecuencia del procesador (a 250 MHz para el caso del Pentium III de 500 Mhz); en cambio, el bus externo que utiliza el procesador es de 100 MHz, independientemente de la velocidad interna del procesador. Debido a la frecuencia de reloj del bus externo, la memoria que se suele utilizar es SDRAM PC-100. No obstante, existen versiones del Pentium III (versiones «B») que pueden utilizar un bus externo de 133 MHz.

El Pentium III tiene una arquitectura con bus dual independiente (Dual Independent Bus, DIB), que significa la inclusión de dos buses independientes que se puede utilizar simultáneamente. Uno es el bus que comunica el procesador con la cache L2, y el otro es el que lo comunica con el sistema de memoria principal. La utilización simultánea de ambos buses y las posibilidades de uso segmentado del bus de acceso a la memoria, contribuyen a aumentar de forma importante el ancho de banda efectivo del procesador. Se estima que este ancho de banda es del orden de tres veces superior al de una arquitectura de un sólo bus. Así, en un Pentium III a 500 MHz, el ancho de banda efectivo sería del orden de 2800 Mbytes/s: 2000 MB/s correspondientes al bus de la cache L2, y 800 MB/s para el bus de memoria principal.

### C4. LECTURAS Y SITIOS WEB RECOMENDADOS

[DIEF97] Dieffendorff, K.; Dubey, P. K.: «How Multimedia workloads will change processor design». IEEE Computer, pp. 43-45. Septiembre, 1997.



#### SITIOS WEB RECOMENDADOS:

- Patwardham, B.: «Introduction to the Streaming SIMD Extensions in the Pentium III». <http://x86.ddj.com/articles/sse-ptl/simd1.htm>



AUGUST 19

---

# Glosario

**A**lgunos de los términos de este glosario vienen del *Diccionario nacional americano para sistemas de información* (1990). Dichos términos se indican con un asterisco.

**Acceso directo a memoria (DMA)** Forma de E/S que utiliza un módulo especial llamado *módulo DMA*, que controla los intercambios de datos entre la memoria principal y un módulo de E/S. La CPU envía una petición de transferencia de un bloque de datos al módulo DMA, y es interrumpido sólo después de que el bloque entero haya sido transferido.

**Acceso directo\*** Capacidad de extraer o introducir datos de/en un dispositivo de almacenamiento en una secuencia independiente de sus posiciones relativas; es decir, de las direcciones que indican la posición física del dato.

**Acumulador** Nombre del registro de la CPU en formato de instrucción de dirección única. El acumulador, o AC, es, implícitamente, uno de los dos operandos de la instrucción.

**Arbitraje del bus** Proceso para determinar a cuál de los controladores del bus que están intentando acceder se le permitirá el acceso al bus.

**Array lógico programable (PLA)\*** Array de puertas, cuyas interconexiones pueden programarse para realizar una función lógica específica.

**Array redundante de discos independientes (RAID)** Discos múltiples en los que parte de la capacidad de almacenamiento se usa para guardar información redundante sobre datos almacenados por el usuario en el resto de la memoria. La información redundante posibilita la regeneración de los datos del usuario en el caso de que falle uno de los discos del array o el camino de acceso a ellos.

**ASCII (American Standard Code for Information Interchange)** Código Estándar Americano para Intercambio de Información. El ASCII es un código de 7 bits, usado para representar caracteres imprimibles numéricos, alfábéticos y especiales. También incluye códigos para *caracteres de control*, que no se pueden imprimir o visualizar, pero que especifican alguna función de control.

## Glosario

**Autoindexación** Una forma de direccionamiento indexado en el que se incrementa o decremente automáticamente el registro índice con cada referencia a memoria.

**Base\*** En el sistema de numeración usado comúnmente en documentos científicos, es el número que se eleva a la potencia indicada por el exponente y se multiplica por la mantisa para determinar el número real representado. (Por ejemplo, el número 6,25 en la expresión  $2,7 \times 6,25^{1,5} = 42,1875$ ).

**Bistable (flip-flop)\*** Circuito o dispositivo que contiene elementos activos, capaz de entrar en uno de dos estados estables posibles en un momento dado. Sinónimo de *circuito bies-table, báscula*.

**Bit de paridad\*** Dígito binario añadido a un grupo de dígitos binarios para hacer la suma de todos los dígitos (unos) impares (paridad impar) o pares (paridad par).

**Bit\*** En el sistema de numeración binario puro, es 0 ó 1.

**Bloque de control de un proceso** Manifestación de un proceso en un sistema operativo. Es una estructura de datos que contiene información sobre las características y estado del proceso.

**Buffer\*** Memoria usada para compensar una diferencia de velocidad en el flujo de datos o de tiempo de aparición de eventos, cuando se transfieren datos de un dispositivo a otro.

**Bus** Camino de comunicación compartido, consistente en una o varias líneas. En algunos computadores, la CPU, la memoria, y los componentes de E/S se conectan a un bus común. Como las líneas son compartidas por varios componentes, sólo uno puede transmitir a la vez.

**Bus de control** Parte del bus para transferir señales de control.

**Bus de datos** Parte de un bus usada para transferir datos.

**Bus de direcciones** Porción de un bus del sistema usada para transferir una dirección. Típicamente, la dirección identifica una posición de la memoria principal o un dispositivo de E/S.

**Bus del sistema** Bus que se usa para interconectar varios componentes (CPU, memoria, E/S).

**Byte** Ocho bits. A veces se denomina *octeto*.

**Canal de E/S** Módulo, relativamente complejo, que releva a la CPU de los detalles de las operaciones de E/S. Un canal de E/S ejecuta una secuencia de órdenes de E/S de la memoria principal sin necesidad de la implicación de la CPU.

**Canal multiplexor** Canal diseñado para operar con varios dispositivos de E/S simultáneamente. Varios dispositivos de E/S pueden transferir registros al mismo tiempo, intercalando campos de datos. Ver también *canal multiplexor de bytes* y *canal multiplexor de bloques*.

**Canal multiplexor de bloques** Canal multiplexor que intercala bloques de datos. Ver también *canal multiplexor de bytes*. Contrastar con *canal selector*.

**Canal multiplexor de bytes\*** Canal multiplexor que intercala bytes de datos. Ver también *canal multiplexor de bloques*. Contrastar con *canal selector*.

**Canal selector** Canal de E/S diseñado para operar sólo con un dispositivo de E/S a la vez. Una vez se haya seleccionado el dispositivo de E/S, se transfiere un registro completo byte a byte. Contrastar con *canal multiplexor de bloques* y *canal multiplexor*.

**CD-ROM (Compact Disk Read-Only Memory)** Disco compacto de sólo lectura, usado como memoria de datos. El sistema estándar usa discos de 12 cm, que pueden almacenar más de 650 Mbytes.

**Ciclo de captación** Parte de un ciclo de instrucción en la que la CPU capta de la memoria la instrucción que se va a ejecutar.

**Ciclo de ejecución** Parte de un ciclo de instrucción en la que la CPU ejecuta la operación especificada por el código de operación de la instrucción.

**Ciclo de instrucción** Proceso realizado por la CPU para ejecutar una instrucción.

**Ciclo de interrupción** Parte del ciclo de instrucción durante la cual la CPU comprueba las interrupciones. Si hay pendiente una interrupción habilitada, la CPU salva el estado actual del programa y procesa una rutina que gestiona la interrupción.

**Ciclo indirecto** Parte del ciclo de instrucción durante la cual la CPU realiza un acceso a memoria para convertir una dirección indirecta en directa.

**Cinta magnética\*** Cinta con una superficie magnetizable donde se pueden almacenar datos grabándolos magnéticamente.

**Circuito combinacional\*** Dispositivo lógico, cuyos valores de salida, en cualquier instante, dependen sólo de los valores de entrada en ese instante. Un circuito combinacional es un caso especial de circuito secuencial, que no tiene capacidad de almacenamiento.

**Circuito integrado (IC)** Pequeño trozo de material sólido (por ejemplo, silicio) sobre el que se graban o imprimen una serie de componentes electrónicos y sus interconexiones.

**Circuito secuencial** Circuito lógico cuya salida depende de la entrada actual y del estado del circuito. Los circuitos secuenciales poseen, por tanto, el atributo de memoria.

**Cluster** Grupo de computadores completos interconectados, trabajando a la vez como un recurso unificado que puede crear la ilusión de ser una única máquina. El término *computador completo* se refiere a un sistema que puede funcionar por sí solo, sin conexión al «cluster».

**Código de condición** Bit que refleja el resultado de una operación (por ejemplo aritmética). La CPU puede incluir uno o más bits de condición, que se pueden almacenar de forma separada en la CPU o como parte de un registro de control. También se le llama *indicador*.

**Código de corrección de errores\*** Código en el que cada carácter o señal se establece conforme a unas reglas específicas de construcción, de manera que las desviaciones de estas reglas indican la presencia de un error; algunos o todos los errores detectados pueden corregirse automáticamente.

**Código de detección de errores\*** Código en el que cada carácter o señal se establece conforme a unas reglas específicas de construcción, de forma que las desviaciones de estas reglas indican la presencia de un error.

**Código de operación\*** Código usado para representar las operaciones de un computador. Se denomina abreviadamente «codop».

**Codop** Abreviatura de *código de operación*.

**Componente de estado sólido\*** Componente cuyo funcionamiento depende del control de un fenómeno eléctrico o magnético en un sólido (por ejemplo, transistor, diodo de cristal o núcleo de ferrita).

**Comunicación de datos** Transferencia de datos entre dispositivos. El término generalmente excluye E/S.

**Contador de programa** Registro que contiene la dirección de la instrucción a ejecutar.

**Controlador de E/S** Módulo de E/S relativamente sencillo, que requiere un control de la CPU o de un canal de E/S. Es sinónimo de *controlador de dispositivo*.

**Controlador del bus** Dispositivo asociado al bus que es capaz de iniciar y controlar la comunicación en el bus.

**CPU microprogramada** CPU cuya unidad de control se implementa usando microprogramación.

**Decodificador\*** Dispositivo en el que las líneas de entrada representan un código, y sólo se activa una de las líneas de salida, existiendo una correspondencia una a una entre las líneas de salida y los códigos de entrada.

**Demandas de página\*** Transferencia de una página desde una memoria auxiliar a una memoria real en el momento en que se necesite.

**Dirección absoluta\*** Dirección, en lenguaje de los computadores, que identifica una posición de almacenamiento o un dispositivo sin usar ninguna referencia intermedia.

**Dirección base\*** Valor numérico que se usa como referencia en el cálculo de direcciones en la ejecución de un programa.

**Dirección directa\*** Dirección que designa la posición de almacenamiento de un dato que va a ser tratado como operando. Sinónimo de dirección de un nivel.

**Dirección indexada\*** Dirección que es modificada por el contenido de un registro índice antes de, o durante, la ejecución de una instrucción.

**Dirección indirecta\*** Dirección de una posición de memoria que contiene a su vez otra dirección.

**Dirección inmediata\*** Contenido de una parte de una dirección que contiene el valor de un operando en lugar de una dirección. Es sinónimo de dirección de nivel cero.

**Disco compacto (CD)** Disco no borrible que almacena información de audio digitalizada.

**Disco en tiras de datos (Disk stripping)** Método de asignación en discos múltiples, en el cual se asignan bloques lógicamente contiguos de datos o tiras, cíclicamente, a discos consecutivos. Un conjunto de tiras lógicamente consecutivas que se asignan exactamente a una tira en cada disco se denomina «barra».

**Disco magnético\*** Sustrato circular (plato) plano con una superficie magnetizable en una o ambas caras donde se pueden almacenar datos.

**Disco óptico borrable** Disco que usa una tecnología óptica en la que una información puede ser fácilmente borrada y reescrita. Hay dos tipos en uso: el de 3,25 y el de 5,25 pulgadas. Una capacidad típica es 650 Mbytes.

**Disquete\*** Disco magnético flexible encerrado en un contenedor protector. Sinónimo de disco flexible.

**E/S aislada** Método de direccionamiento de módulos de E/S y dispositivos externos. El espacio de direcciones de E/S se trata separadamente del espacio de direcciones de la memoria principal. Se usan instrucciones máquina específicas de E/S. Comparar con E/S asignada en memoria.

**E/S asignada en memoria** Método de direccionamiento de módulos de E/S y dispositivos externos. Se usa un único espacio de direcciones, tanto para las direcciones de memoria principal como para E/S, y las mismas instrucciones máquina se utilizan tanto para lectura/escritura en memoria como para E/S.

**E/S dirigida por interrupciones** Tipo de E/S. La CPU da una orden de E/S, continua ejecutando las instrucciones siguientes, y es interrumpida por el módulo de E/S cuando ha terminado su trabajo.

**E/S programada** Una forma de E/S en la que la CPU envía una orden de E/S a un módulo de E/S y debe, por tanto, esperar a que la operación termine antes de proceder a concluir su ejecución.

**Ejecución de predicados** Mecanismo que permite la ejecución condicional de instrucciones individuales. Esto hace posible ejecutar especulativamente las distintas ramificaciones de una instrucción de ramificación y retener los resultados de la ramificación que definitivamente se seleccionó.

**Ejecución especulativa** Ejecución de instrucciones a lo largo de uno de los caminos de una bifurcación. Si, al final, resulta que no se ha tomado esta rama, entonces los resultados de la ejecución especulativa se descartan.

**Emulación\*** Imitación de todo o parte de un sistema por otro, usualmente hardware, de forma que el sistema de imitación acepta los mismos datos, ejecuta los mismos programas, y produce los mismos resultados que el sistema imitado.

**Entrada/salida (E/S)** Se refiere a una entrada o una salida o a ambos. Trata de la transferencia de datos entre un computador y un periférico unido directamente a él.

**Equipo periférico (IBM)** En un computador, con respecto a una unidad de procesamiento particular, es cualquier equipo que proporcione a dicha unidad de procesamiento una comunicación externa. Sinónimo de *periférico*.

**Escalar\*** Cantidad caracterizada por un único valor.

**Espacio de direcciones** Rango de direcciones (memoria, E/S) que se pueden referenciar.

**Fallo de página** Se produce cuando se referencia una palabra que está en una página que no se encuentra en la memoria principal. Esto causa una interrupción, y requiere que el sistema operativo proporcione la página que se necesita.

**Firmware\*** Microcódigo almacenado en memorias de sólo lectura.

**Formato de una instrucción** Estructura de una instrucción como secuencia de bits. El formato divide la instrucción en campos, que corresponden a los elementos que constituyen la instrucción (por ejemplo: código de operación, operandos).

**G** Prefijo que indica *mil millones*.

**Indexación** Técnica de modificación de direcciones mediante registros índice.

**Instrucción de un computador\*** Instrucción que puede ser reconocida por la unidad de procesamiento del computador para la que fue diseñada. Sinónimo de *instrucción máquina*.

**Interrupción deshabilitada** Condición, normalmente creada por la CPU, durante la cual la CPU ignora las señales de petición de interrupción de un tipo especificado.

**Interrupción habilitada** Condición, normalmente determinada por la CPU, mediante la cual la CPU responde a las señales de petición de interrupción de una clase predeterminada.

**Interrupción\*** Suspensión de un proceso (ejecución de un programa), causada por un evento externo, y realizada de tal forma que el proceso se puede reanudar.

**K** Prefijo que significa  $2^{10} = 1.024$ . Entonces,  $2 \text{ Kb} = 2 \times 220 \text{ bits}$ .

**Lenguaje de microprogramación\*** Conjunto de instrucciones usado para especificar microprogramas.

**Lenguaje ensamblador\*** Lenguaje orientado al computador cuyas instrucciones se suelen corresponder una a una con las instrucciones del computador, y que posibilita poder definir, por ejemplo, macroinstrucciones. Es sinónimo de *lenguaje dependiente del computador*.

**Línea de cache** Bloque de datos asociado a una etiqueta de la cache y a la unidad de transferencia entre cache y memoria.

**Localidad de referencia** Tendencia de un procesador a acceder al mismo conjunto de lugares de memoria de forma repetida en periodo de tiempo corto.

**M** prefijo que significa  $2^{20} = 1.048.576$ . Por tanto, 2 Mb =  $2 \times 2^{20}$  bits.

**Marco de página** Zona de la memoria principal para guardar una página.

**Maya en cadena\*** Método de interconexión de dispositivos para determinar prioridades de interrupción conectando las fuentes de interrupción vía serie.

**Memoria asociativa\*** Memoria cuyas posiciones de almacenamiento se identifican por su contenido o por parte de éste, en vez de por sus nombres o posiciones.

**Memoria cache\*** Buffer especial de almacenamiento, menor y más rápido que la memoria principal, que es usado para guardar una copia de las instrucciones y datos de la memoria principal que el procesador va a necesitar, y que obtiene automáticamente de la memoria principal.

**Memoria de acceso aleatorio (RAM)** Memoria en la que cada posición direccionable tiene un único mecanismo de direccionamiento. El tiempo de acceso a una posición dada es independiente de la secuencia de acceso previa.

**Memoria de control** Porción de la memoria que contiene microcódigo.

**Memoria de sólo lectura (ROM)** Memoria semiconductora cuyo contenido no se puede cambiar, salvo destruyéndola. Es una memoria no borrable.

**Memoria no volátil** Memoria cuyo contenido es estable y no requiere una fuente de alimentación permanente.

**Memoria principal\*** Memoria direccionable por programa en la que se pueden cargar, en posiciones, instrucciones y otros datos para su ejecución o procesamiento posterior.

**Memoria programable de sólo lectura (PROM)** Memoria semiconductora cuyo contenido se puede establecer una sola vez. El proceso de escritura se lleva a cabo eléctricamente, y lo puede hacer el usuario después de la fabricación del chip.

**Memoria secundaria** Memoria situada fuera del computador, como disco o cinta.

**Memoria virtual\*** Memoria que permite ser vista por el usuario como una memoria principal direccionable de un computador, en el que las direcciones virtuales se transforman en direcciones reales. El tamaño de la memoria virtual está limitado por el modo de direccionamiento del computador y por la cantidad de memoria exterior disponible, y no por el número real de posiciones de la memoria principal.

**Memoria volátil** Memoria que requiere permanentemente alimentación eléctrica para mantener su contenido. Si la alimentación se interrumpe, la información se pierde.

**Microcomputador\*** Computador cuya unidad de procesamiento es un microprocesador. Un microcomputador básico incluye un microprocesador, una memoria, y entradas/salidas que pueden o no estar en un solo chip.

**Microinstrucción\*** Instrucción que controla el flujo de datos y el secuenciamiento en un procesador, a un nivel más básico que las instrucciones máquina. Las instrucciones máquina individuales y quizás otras funciones, se pueden implementar en microprogramas.

**Microoperación** Operación elemental de la CPU, ejecutada durante un pulso de reloj.

**Microprocesador\*** Un procesador cuyos elementos se han miniaturizado en uno o varios circuitos integrados.

**Microprograma\*** Secuencia de microinstrucciones que están en un memoria especial, donde se puede acceder a ellas dinámicamente para realizar funciones diversas.

**Módulo de E/S** Uno de los principales componentes del computador. Es responsable del control de uno o más dispositivos externos (periféricos), y del intercambio de datos entre estos y la memoria principal y/o los registros de la CPU.

**Monoprocesamiento** Ejecución secuencial de instrucciones bajo la dirección de una unidad de procesamiento, o uso independiente de la unidad de procesamiento en un sistema multiprocesador.

**Multiplexor** Circuito combinacional que conecta una de las varias entradas a una sola salida. En un momento dado, sólo una de las entradas puede estar seleccionada para pasar a la salida.

**Multiprocesador** Computador que tiene uno o más procesadores con acceso común a la memoria principal.

**Multiprocesador con acceso a memoria no uniforme (NUMA)** Multiprocesador con memoria compartida en el que el tiempo de acceso a memoria desde un procesador dado a una palabra de memoria, varía con la posición de la palabra en la memoria.

**Multiprocesamiento simétrico (SMP)** Forma de multiprocesamiento que permite a un sistema operativo ejecutar en cualquier procesador disponible o en un conjunto de procesadores disponibles, simultáneamente.

**Multiprogramación\*** Forma de funcionamiento que posibilita la ejecución intercalada de dos o más programas, con un solo procesador.

**Núcleo** Parte del sistema operativo que contiene las funciones básicas y más usadas. A menudo, los núcleos permanecen residentes en la memoria principal.

**Operador binario\*** Operador que representa una operación con dos, y solamente dos, operandos.

**Operador unitario\*** Operador que representa una operación con un sólo operando.

**Operando\*** Entidad en la que se realiza una operación.

**Ortogonalidad** Principio por el que dos variables o dimensiones son independientes una de la otra. En el contexto de un conjunto de instrucciones, el término se usa generalmente para indicar que los elementos de una instrucción (modo de direccionamiento, número de operandos y longitud del operando) son independientes de, o no están determinados por, el código de operación.

**Página\*** En un sistema de memoria virtual, un bloque con una longitud fija que tiene una dirección virtual, y que se transfiere como una unidad entre la memoria principal y la memoria auxiliar.

**Palabra de estado de un programa (PSW)** Un área en la memoria usada para indicar el orden en que se deben ejecutar las instrucciones, y que mantiene e indica el estado del computador. Sinónimo de *palabro de estado del procesador*.

**Palabra de instrucción muy larga (VLIW, Very Long Instruction Word)** Se refiere al uso de instrucciones que contienen operaciones múltiples. En efecto, las instrucciones múltiples están contenidas en una sola palabra. Usualmente, el compilador construye VLIW, situando las operaciones que se pueden ejecutar en paralelo en la misma palabra.

**Paquete de discos\*** Conjunto de discos magnéticos que se pueden sacar como un todo de la unidad de disco, junto con un contenedor desde el que el conjunto debe ser sacado cuando se utilice.

**Pila\*** Lista que se forma y mantiene, de forma que el elemento próximo a extraer es el que se ha almacenado más recientemente: último en entrar, primero en salir (LIFO, Last In First Out).

**Predicción de saltos** Mecanismo usado por el procesador para predecir el inicio de una rama de un programa previamente a su ejecución.

**Procesador con supersegmentación de cauce** Procesador en el que el cauce de segmentación de las instrucciones está formado por etapas muy pequeñas, de forma que se puede ejecutar más de una etapa en un ciclo de reloj y, por tanto, puede haber simultáneamente muchas instrucciones en el cauce.

**Procesador de E/S** Módulo de E/S con procesador propio, capaz de ejecutar instrucciones especializadas de E/S o, en algunos casos, instrucciones máquina de uso general.

**Procesador superscalar** Procesador que incluye cauces segmentados para múltiples instrucciones, de forma que se pueden ejecutar simultáneamente más de una instrucción en la misma etapa.

**Procesador\*** Unidad funcional que interpreta y ejecuta instrucciones en un computador. Un procesador consta al menos de una unidad de control y una unidad aritmética.

**Proceso** Programa en ejecución. Un proceso se controla y temporiza con el sistema operativo.

**Protocolo de coherencia de cache** Mecanismo para mantener la validez de los datos entre varias caches, de forma que cada acceso a los datos adquirirá siempre la versión más reciente del contenido de una palabra de la memoria principal.

**Puerta** Circuito electrónico que produce una señal de salida que es una simple operación booleana de sus señales de entrada.

**RAM dinámica** RAM cuyas celdas se implementan usando condensadores. Una RAM dinámica pierde gradualmente sus datos, salvo si se refresca periódicamente.

**RAM estática** RAM cuyas celdas se implementan con biestables. Una RAM estática mantendrá sus datos mientras se le suministre corriente; no se necesita un refresco periódico.

**Registro búffer de memoria (MBR)** Registro que contiene datos leídos de la memoria o datos a escribir en la memoria.

**Registro de dirección de instrucción\*** Registro de uso especial utilizado para guardar la dirección de la siguiente instrucción que se va a ejecutar.

**Registro de dirección de memoria (MAR)\*** Registro de la unidad de procesamiento que contiene la dirección de la posición de memoria a la que se va a acceder.

**Registro de uso general\*** Registro, dentro de un conjunto de ellos, normalmente direccionable explícitamente, que se puede usar para distintos objetivos; por ejemplo, como acumulador, registro de indexación, o manipulador especial de datos.

**Registro índice\*** Registro cuyo contenido puede ser usado para modificar una dirección de un operando durante la ejecución de instrucciones; también puede ser usado como un contador. Un registro índice puede usarse para controlar la ejecución de un bucle, controlar el uso de un array, como conmutador, como índice de una tabla de consulta, o como un puntero.

**Registro instrucción\*** Registro que se usa para guardar e interpretar la instrucción a ejecutar.

**Registros de control** Registros de la CPU empleados para controlar operaciones de la CPU. La mayoría de estos registros no son visibles al usuario.

**Registros** Memoria muy rápida, interna a la CPU. Algunos registros son visibles al usuario; es decir, se pueden programar a través del conjunto de instrucciones máquina. Otros registros sólo los puede usar la CPU, con fines de control.

**Registros visibles al usuario** Registros de la CPU que pueden ser referenciados por el programador. El formato del conjunto de instrucciones permite especificar uno o más registros como operandos o direcciones de operandos.

**Repertorio de instrucciones de un computador\*** Conjunto completo de instrucciones de un computador junto con una descripción de los significados que se pueden atribuir a las mismas. Sinónimo de *conjunto de instrucciones máquina*.

**Representación en complemento a dos** Se usa para representar enteros binarios. Un entero positivo se representa en signo y magnitud. Un número negativo se representa sumando uno a la representación en complemento a uno del mismo número.

**Representación en complemento a uno** Usada para representar enteros binarios. Un entero positivo se representa en signo y magnitud. Un entero negativo se representa invirtiendo cada bit en la representación de un entero positivo de la misma magnitud.

**Representación en signo y magnitud** Usada para representar enteros binarios. En una palabra de  $N$  bits, el bit que está más a la izquierda es el bit de signo (0 = positivo, 1 = negativo), y el resto, los otros  $N - 1$  bits, son la magnitud del número.

**Salto condicional\*** Salto que se produce sólo cuando la instrucción se ejecuta y, además, se satisfacen unas condiciones especiales. Contrastar con *salto incondicional*.

**Salto incondicional\*** Salto que se produce siempre que la instrucción que lo contiene se ejecute.

**Segmentación de cauce** Organización de un procesador en la que el procesador consta de una serie de etapas, permitiendo la ejecución concurrente de instrucciones múltiples.

**Semiconductor** Material semicristalino, como silicio o germanio, cuya conductividad eléctrica está entre aislante y buen conductor. Se usa para fabricar transistores y componentes de estado sólido.

**Sistema de representación en coma fija\*** Sistema de numeración en el que la posición de la coma que separa la parte entera de la decimal se ha fijado implícitamente en la serie de dígitos por alguna convención previamente establecida.

**Sistema de representación en coma flotante\*** Sistema de numeración en el que un número real es representado por un par de números, siendo el valor del número real el producto de uno de los números (que está representado en coma fija) por la base implícita elevada a la potencia indicada por el segundo número (que es el exponente).

**Sistema operativo\*** Software que controla la ejecución de programas y ofrece servicios como reserva de recursos, planificación, control de entradas/salidas y gestión de datos.

**Tabla de verdad\*** Tabla que describe una función lógica mostrando todas las combinaciones posibles de las entradas, e indicando, para cada combinación, la salida.

**Temporización asíncrona** Técnica en la que la aparición de un evento en el bus sigue a, y depende de, la aparición de un evento previo.

**Temporización síncrona** Técnica en la que la aparición de un evento en el bus viene determinada por un reloj. El reloj define intervalos de tiempo de igual duración, y los eventos sólo pueden empezar al principio de un intervalo.

**Tiempo de ciclo de memoria** Inversa de la velocidad a la que se puede acceder a la memoria. Es el tiempo mínimo entre la respuesta a una petición de acceso (lectura o escritura) y la respuesta a la próxima petición de acceso.

**Tiempo de ciclo del procesador** Tiempo requerido para realizar la microoperación más corta de la CPU. Es la unidad básica de tiempo para medir todas las acciones de la CPU. Sinónimo de *tiempo de ciclo máquina*.

**Unidad aritmético lógica (ALU)\*** Parte del computador que realiza las operaciones aritméticas, lógicas y de relación.

**Unidad central de procesamiento (CPU)** Parte del computador que capta y ejecuta instrucciones. Está formada por la unidad aritmético lógica (ALU), la unidad de control y los registros. A menudo se le suele llamar *procesador*.

**Unidad de control** Parte de la CPU que controla las operaciones de la CPU, incluyendo las operaciones de la ALU, las transferencias de datos en la CPU, y el intercambio de datos y señales de control a través de las interfaces externas (por ejemplo, a través del bus).

**Variable global** Variable definida en una parte de un programa que es usada al menos en alguna otra parte del programa.

**Variable local** Variable que se define y se usa sólo en una parte de un programa.

**Vector\*** Cantidad que se caracteriza por un conjunto ordenado de escalares.

**WORM (Write Once, Read Many, «escribir una vez, leer muchas»)** Disco en el que se escribe más fácilmente que en un CD-ROM, siendo comercialmente factible hacer copias. Como un CD-ROM, después de escribir en él ya sólo se puede leer. El tamaño más normal es de 5,25 pulgadas, pudiendo almacenar entre 200 y 800 MBytes de datos.

# Bibliografía

- ACOS86 Acosta, R.; Kjelstrup, J.; and Torgn, H. «An Instruction Issuing Approach to Enhancing Performance in Multiple Functional Unit Processors.» *IEEE Transactions on Computers*, September 1986.
- ADAM91 Adamek, J. *Foundations of Coding*. New York: Wiley, 1991.
- AGAR89a Agarwal, A. *Analysis of Cache Performance for Operating Systems and Multiprogramming*. Boston: Kluwer Academic Publishers, 1989.
- AGER87 Agerwala, T., and Cocke, J. *High Performance Reduced Instruction Set Processors*. Technical Report RC12434 (#55845). Yorktown, NY: IBM Thomas J. Watson Research Center, January 1987.
- ALEX93 Alexandridis, N. *Design of Microprocessor-Based Systems*. Englewood Cliffs, NJ: Prentice Hall, 1993.
- ANDE67 Anderson, D.; Sparacio, F.; and Tomasulo, F. «The IBM System/360 Model 91: Machine Philosophy and Instruction Handling.» *IBM Journal of Research and Development*, January 1967.
- ANDE98a Anderson, D., and Shanley, T. *Pentium Pro and Pentium II System Architecture*. Reading, MA: Addison-Wesley, 1998.
- ANDE98b Anderson, D. *FireWire System Architecture*. Reading, MA: Addison-Wesley, 1998.
- ATKI96 Atkins, M. «PC Software Performance Tuning.» *IEEE Computer*, August 1996.
- AZIM92 Azimi, M.; Prasad, B.; and Bhat, K. «Two Level Cache Architectures.» *Proceedings COMPCON '92*. February 1992.
- BAEN97 Baentsch, M., et al. «Enhancing the Web's Infrastructure: From Caching to Replication.» *Internet Computing*, March/April 1997.
- BASH81 Bashe, C.; Bucholtz, W.; Hawkins, G.; Ingram, J.; and Rochester, N. «The Architecture of IBM's Early Computers.» *IBM Journal of Research and Development*, September 1981.
- BASH91 Bassteen, A.; Lui, I.; and Mullan, J. «A Superpipeline Approach to the MIPS Architecture.» *Proceedings, COMPCON Spring '91*. February 1991.
- BELL70 Bell, C.; Cady, R.; McFarland, H.; Delagi, B.; O'Loughlin, J.; and Noonan, R. «A New Architecture for Minicomputers—The DEC PDP-11.» *Proceedings, Spring Joint Computer Conference*, 1970.

- BELL71** Bell, C., and Newell, A. *Computer Structures: Readings and Examples*. New York: McGraw-Hill, 1971.
- BELL78a** Bell, C.; Mudge, J.; and McNamara, J. *Computer Engineering: A DEC View of Hardware Systems Design*. Bedford, MA: Digital Press, 1978.
- BELL78b** Bell, C.; Newell, A.; and Siewiorek, D. «Structural Levels of the PDP-8.» In [BELL78a].
- BELL78c** Bell, C.; Kotok, A.; Hastings, T.; and Hill, R. «The Evolution of the DEC System-10.» *Communications of the ACM*, January 1978.
- BENH92** Benham, J. «A Geometric Approach to Presenting Computer Representations of Integers.» *SIGCSE Bulletin*, December 1992.
- BETK97** Betker, M.; Fernando, J.; and Whalen, S. «The History of the Microprocessor.» *Bell Labs Technical Journal*, Autumn 1997.
- BLAA97** Blaauw, G., and Brooks, F. *Computer Architecture: Concepts and Evolution*. Reading, MA: Addison-Wesley, 1997.
- BLAH83** Blahut, R. *Theory and Practice of Error Control Codes*. Reading, MA: Addison-Wesley, 1983.
- BOHR98** Bohr, M. «Silicon Trends and Limits for Advanced Microprocessors.» *Communications of the ACM*, March 1998.
- BOND94** Bondurant, D. «Low Latency EDRAM Main Memory Subsystem for 66 MHz Bus Operation.» *Proceedings. COMPON '94*. March 1994.
- BRAD91a** Bradlee, D.; Eggers, S.; and Henry, R. «The Effect on RISC Performance of Register Set Size and Structure Versus Code Generation Strategy.» *Proceedings. 18th Annual International Symposium on Computer Architecture*. May 1991.
- BRAD91b** Bradlee, D.; Eggers, S.; and Henry, R. «Integrating Register Allocation and Instruction Scheduling for RISCs.» *Proceedings. Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. April 1991.
- BREW97** Brewer, E. «Clustering: Multiply and Conquer.» *Data Communications*, July 1997.
- BREY97** Brey, B. *The Intel Microprocessors: 8086/8066, 80186/80188, 80286, 80386, 80486, Pentium, and Pentium Processor*. Upper Saddle River, NJ: Prentice Hall, 1997.
- BURG97** Burger, D., and Austin, T. «The SimpleScalar Tool Set, Version 2.0.» *Computer Architecture News*, June 1997.
- BURK46** Burks, A.; Goldstine, H.; and von Neumann, J. *Preliminary Discussion of the Logical Design of an Electronic Computer Instrument*. Report prepared for U.S. Army Ordnance Dept., 1946, reprinted in [BELL71a].
- CART96** Carter, J. *Microprocessor Architecture and Microprogramming*. Upper Saddle River, NJ: Prentice Hall, 1996.
- CATA94** Catanzaro, B. *Multiprocessor System Architectures*. Mountain View, CA: Sunsoft Press, 1994.
- CHAI82** Chaitin, G. «Register Allocation and Spilling via Graph Coloring.» *Proceedings. SIGPLAN Symposium on Compiler Construction*. June 1982.
- CHEN94** Chen, P.; Lee, E.; Gibson, G.; Katz, R.; and Patterson, D. «RAID: High-Performance, Reliable Secondary Storage.» *ACM Computing Surveys*. June 1994.
- CHOW86** Chow, F.; Himmelstein, M.; Killian, E.; and Weber, L. «Engineering a RISC Compiler System.» *Proceedings. COMPON Spring '86*. March 1986.
- CHOW87** Chow, F.; Corteil, S.; Himmelstein, M.; Killian, E.; and Weber, L. «How Many Addressing Modes Are Enough?» *Proceedings. Second International Conference on Architectural Support for Programming Languages and Operating Systems*. October 1987.
- CHOW90** Chow, F., and Hennessy, J. «The Priority-Based Coloring Approach to Register Allocation.» *ACM Transactions on Programming Languages*. October 1990.

- CLAR85 Clark, D., and Emer, J. «Performance of the VAX-11/780 Translation Buffer: Simulation and Measurement.» *ACM Transactions on Computer Systems*, February 1985.
- COHE81 Cohen, D. «On Holy Wars and a Plea for Peace.» *Computer*, October 1981.
- COLW85a Colwell, R.; Hitchcock, C.; Jensen, E.; Brinkley-Sprunt, H.; and Kollar, C. «Computers, Complexity, and Controversy.» *Computer*, September 1985.
- COLW85b Colwell, R.; Hitchcock, C.; Jensen, E.; and Sprunt, H. «More Controversy About Computers, Complexity, and Controversy.» *Computer*, December 1985.
- COME95 Comerford, R. «An Overview of High Performance.» *IEEE Spectrum*, April 1995.
- COOK82 Cook, R., and Dande, N. «An Experiment to Improve Operand Addressing.» *Proceedings. Symposium on Architecture Support for Programming Languages and Operating Systems*, March 1982.
- COON81 Coonen, J. «Underflow and Denormalized Numbers.» *IEEE Computer*, March 1981.
- COUT86 Coutant, D.; Hammond, C.; and Kelley, J. «Compilers for the New Generation of Hewlett-Packard Computers.» *Proceedings. COMPCON Spring '86*, March 1986.
- CRAG79 Cragon, H. «An Evaluation of Code Space Requirements and Performance of Various Architectures.» *Computer Architecture News*, February 1979.
- CRAW90 Crawford, J. «The i486 CPU: Executing Instructions in One Clock Cycle.» *IEEE Micro*, February 1990.
- CRAG92 Cragon, H. *Branch Strategy Taxonomy and Performance Models*. Los Alamitos, CA: IEEE Computer Society Press, 1992.
- CRIS97 Crisp, R. «Direct RAMBUS Technology: The New Main Memory Standard.» *IEEE Micro*, November/December 1997.
- DATT93 Dattatreya, G. «A Systematic Approach to Teaching Binary Arithmetic in a First Course.» *IEEE Transactions on Education*, February 1993.
- DAVI87 Davidson, J., and Vaughan, R. «The Effect of Instruction Set Complexity on Program Size and Memory Performance.» *Proceedings. Second International Conference on Architectural Support for Programming Languages and Operating Systems*, October 1987.
- DENN68 Denning, P. «The Working Set Model for Program Behavior.» *Communications of the ACM*, May 1968.
- DEWA90 Dewar, R., and Smosna, M. *Microprocessors: A Programmer's View*. New York: McGraw-Hill, 1990.
- DIEF94 Dieffendorff, K.; Oehler, R.; and Hochsprung, R. «Evolution of the PowerPC Architecture.» *IEEE Micro*, April 1994.
- DIJK63 Dijkstra, E. «Making an ALGOL Translator for the XI.» in *Annual Review of Automatic Programming*, Vol. 4, Pergamon, 1963.
- DOET97 Doetting, G., et al. «S/390 Parallel Enterprise Server Generation 3: A Balanced System and Cache Structure.» *IBM Journal of Research and Development*, July/September 1997.
- DUBE91 Dubey, P., and Flynn, M. «Branch Strategies: Modeling and Optimization.» *IEEE Transactions on Computers*, October 1991.
- DULO98 Dulong, C. «The IA-64 Architecture at Work.» *Computer*, July 1998.
- ECKE90 Erkert, R. «Communication Between Computers and Peripheral Devices—An Analogy.» *ACM SIGSCE Bulletin*, September 1990.
- ELAY85 El-Ayat, K., and Agarwal, R. «The Intel 80386—Architecture and Implementation.» *IEEE Micro*, December 1985.
- EVER98 Evers, M., et al. «An Analysis of Correlation and Predictability: What Makes Two-Level Branch Predictors Work.» *Proceedings. 25th Annual International Symposium on Microarchitecture*, July 1998.
- FITZ81 Fitzpatrick, D., et al. «A RISCy Approach to VLSI.» *VLSI Design*, 4th quarter, 1981. Reprinted in *Computer Architecture News*, March 1982.

## Bibliografia

- FLYN71 Flynn, M., and Rosin, R. «Microprogramming: An Introduction and a Viewpoint.» *IEEE Transactions on Computers*, July 1971.
- FLYN72 Flynn, M. «Some Computer Organizations and Their Effectiveness.» *IEEE Transactions on Computers*, September 1972.
- FLYN85 Flynn, M.; Johnson, J.; and Wakefield, S. «On Instruction Sets and Their Formats.» *IEEE Transactions on Computers*, March 1985.
- FLYN87 Flynn, M.; Mitchell, C.; and Mulder, J. «And Now a Case for More Complex Instruction Sets.» *Computer*, September 1987.
- FRAI83 Frailey, D. «Word Length of a Computer Architecture: Definitions and Applications.» *Computer Architecture News*, June 1983.
- FRIE96 Friedman, M. «RAID Keeps Going and Going and ...» *IEEE Spectrum*, April 1996.
- FURH87 Furht, B., and Milutinovic, V. «A Survey of Microprocessor Architectures for Memory Management.» *Computer*, March 1987.
- GARR94 Garrett, B. «RDRAMs: A New Speed Paradigm.» *Proceedings. COMPCON '94*, March 1994.
- GEHR88 Gehringer, E.; Abullarade, J.; and Gulyn, M. «A Survey of Commercial Parallel Processors.» *Computer Architecture News*, September 1988.
- GIFF87 Gifford, D., and Spector, A. «Case Study: IBM's System/360-370 Architecture.» *Communications of the ACM*, April 1987.
- GILL97 Gillingham, P., and Vogley, B. «SLDRAM: High-Performance Open-Standard Memory.» *IEEE Micro*, November/December 1997.
- GJES92 Gjessing, S., et al. «A RAM Link for High Speed.» *IEEE Spectrum*, October 1992.
- GOLD91 Goldberg, D. «What Every Computer Scientist Should Know About Floating-Point Arithmetic.» *ACM Computing Surveys*, March 1991. Available at <http://www.validgh.com/>
- GOOR89 Goor, A. *Computer Architecture and Design*. Reading, MA: Addison-Wesley, 1989.
- HALF97 Halfhill, T. «Beyond Pentium II... Byte», December 1997.
- HAND98 Handy, J. *The Cache Memory Book*. San Diego: Academic Press, 1993.
- HAYE88 Hayes, J. *Computer Architecture and Organization*. New York: McGraw-Hill, 1988.
- HEAT84 Heath, J. «Re-evaluation of RISC I.» *Computer Architecture News*, March 1984.
- HENN82 Hennessy, J., et al. «Hardware/Software Tradeoffs for Increased Performance.» *Proceedings. Symposium on Architectural Support for Programming Languages and Operating Systems*, March 1982.
- HENN84 Hennessy, J. «VLSI Processor Architecture.» *IEEE Transactions on Computers*, December 1984.
- HENN91 Hennessy, J., and Jouppi, N. «Computer Technology and Architecture: An Evolving Interaction.» *Computer*, September 1991.
- HENN96 Hennessy, J., and Patterson, D. *Computer Architecture: A Quantitative Approach*. San Mateo, CA: Morgan Kaufmann, 1996.
- HIDA90 Hidaka, H.; Matsuda, Y.; Asakura, M.; and Kazuyasu, F. «The Cache DRAM Architecture: A DRAM with an On-Chip Cache Memory.» *IEEE Micro*, April 1990.
- HIGB90 Higbie, L. «Quick and Easy Cache Performance Analysis.» *Computer Architecture News*, June 1990.
- HILL64 Hill, R. «Stored Logic Programming and Applications.» *Datamation*, February 1964.
- HILL89 HILL, M. «EVALUATING ASSOCIATIVITY IN CPU CACHES.» *IEEE TRANSACTIONS ON COMPUTERS*, DECEMBER 1989.
- HP96 Hewlett Packard. *White Paper on Clustering*. Available at [http://www.hp.com/netserver/partners/papers/wp\\_clust\\_696.html](http://www.hp.com/netserver/partners/papers/wp_clust_696.html). June 1996.

- HUCK83 Huck, T. *Comparative Analysis of Computer Architectures*. Stanford University Technical Report No. 83-243, May 1983.
- HUGU91 Huguet, M., and Lang, T. «Architectural Support for Reduced Register Saving/Restoring in Single-Window Register Files.» *ACM Transactions on Computer Systems*, February 1991.
- HUTC96 Hutcheson, G., and Hutcheson, J. «Technology and Economics in the Semiconductor Industry.» *Scientific American*, January 1996.
- HWAN93 Hwang, K. *Advanced Computer Architecture*. New York: McGraw-Hill, 1993.
- HWU98 Hwu, W. «Introduction to Predicated Execution.» *Computer*, January 1998.
- IBM94 International Business Machines, Inc. *The PowerPC Architecture: A Specification for a New Family of RISC Processors*. San Francisco, CA: Morgan Kaufmann, 1994.
- IEEE85 Institute of Electrical and Electronics Engineers. *IEEE Standard for Binary Floating-Point Arithmetic*. ANSI/IEEE Std 754-1985, 1985.
- INTE98a Intel Corp. *Pentium Processors and Related Products*. Aurora, CO, 1998.
- INTE98b Intel Corp. *Pentium Pro and Pentium II Processors and Related Products*. Aurora, CO, 1998.
- JAME90 James, D. «Multiplexed Buses: The Endian Wars Continue.» *IEEE Micro*, September 1990.
- JOHN91 Johnson, M. *Superscalar Microprocessor Design*. Englewood Cliffs, NJ: Prentice-Hall, 1991.
- JOUP88 Jouppi, N. «Superscalar versus Superpipelined Machines.» *Computer Architecture News*, June 1988.
- JOUP89a Jouppi, N., and Wall, D. «Available Instruction-Level Parallelism for Superscalar and Superpipelined Machines.» *Proceedings, Third International Conference on Architectural Support for Programming Languages and Operating Systems*, April 1989.
- JOUP89b Jouppi, N. «The Nonuniform Distribution of Instruction-Level and Machine Parallelism and Its Effect on Performance.» *IEEE Transactions on Computers*, December 1989.
- KAEL91 Kaeli, D., and Emma, P. «Branch History Table Prediction of Moving Target Branches Due to Subroutine Returns.» *Proceedings, 18th Annual International Symposium on Computer Architecture*, May 1991.
- KANE92 Kane, G., and Heinrich, J. *MIPS RISC Architecture*. Englewood Cliffs, NJ: Prentice Hall, 1992.
- KATE83 Katevenis, M. *Reduced Instruction Set Computer Architectures for VLSI*. Ph.D. dissertation, Computer Science Department, University of California at Berkeley, October 1983. Reprinted by MIT Press, Cambridge, MA, 1985.
- KATZ89 Katz, R.; Gibson, G.; and Patterson, D. «Disk System Architecture for High Performance Computing.» *Proceedings of the IEEE*, December 1989.
- KNUT71 Knuth, D. «An Empirical Study of FORTRAN Programs.» *Software Practice and Experience*, vol. 1, 1971.
- KNUT98 Knuth, D. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Reading, MA: Addison-Wesley, 1998.
- KORE93 Koren, I. *Computer Arithmetic Algorithms*. Englewood Cliffs, NJ: Prentice Hall, 1993.
- KUCK72 Kuck, D.; Muraoka, Y.; and Chen, S. «On the Number of Operations Simultaneously Executable in Fortran-like Programs and Their Resulting Speedup.» *IEEE Transactions on Computers*, December 1972.
- KUGA91 Kuga, M.; Murakami, K.; and Tomita, S. «DSNS (Dynamically-hazard resolved, Statically-code-scheduled, Nonuniform Superscalar): Yet Another Superscalar Processor Architecture.» *Computer Architecture News*, June 1991.

## Bibliografia

- LÉE91 Lee, R.; Kwok, A.; and Briggs, F. «The Floating Point Performance of a Superscalar SPARC Processor.» *Proceedings. Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, April 1991.
- LILJ88 Lilja, D. «Reducing the Branch Penalty in Pipelined Processors.» *Computer*, July 1988.
- LILJ93 Lilja, D. «Cache Coherence in Large-Scale Shared-Memory Multiprocessors: Issues and Comparisons.» *ACM Computing Surveys*, September 1993.
- LOVE96 Lovett, T., and Clapp, R. «Implementation and Performance of a CC-NUMA System.» *Proceedings. 23rd Annual International Symposium on Computer Architecture*, May 1996.
- LUND77 Lunde, A. «Empirical Evaluation of Some Features of Instruction Set Processor Architectures.» *Communications of the ACM*, March 1977.
- LYNC93 Lynch, M. *Microprogrammed State Machine Design*. Boca Raton, FL: CRC Press, 1993.
- MACG84 MacGregor, D.; Mothersole, D.; and Moyer, B. «The Motorola MC68020.» *IEEE Micro*, August 1984.
- MAHL94 Mahlke, S., et al. «Characterizing the Impact of Predicated Execution on Branch Prediction.» *Proceedings. 27th International Symposium on Microarchitecture*, December 1994.
- MAHL95 Mahlke, S., et al. «A Comparison of Full and Partial Predicated Execution Support for ILP Processors.» *Proceedings. 22nd International Symposium on Computer Architecture*, June 1995.
- MAK97 Mak, P., et al. «Shared-Cache Clusters in a System with a Fully Shared Memory.» *IBM Journal of Research and Development*, July/September 1997.
- MALL79 Mallach, E. «The Evolution of an Architecture.» *Datamation*, April 1979.
- MALL75 Mallach, E. «Emulation Architecture.» *Computer*, August 1975.
- MALL83 Mallach, E., and Sondak, N. *Advances in Microprogramming*. Dedham, MA: Artech House, 1983.
- MANO97 Mano, M., and Kime, C. *Logic and Computer Design Fundamentals*. Upper Saddle River, NJ: Prentice Hall, 1997.
- MANS97 Mansuripur, M., and Sincerbox, G. «Principles and Techniques of Optical Data Storage.» *Proceedings of the IEEE*, November 1997.
- MARC90 Marchant, A. *Optical Recording*. Reading, MA: Addison-Wesley, 1990.
- MASH95 Mashey, J. «CISC vs. RISC (or what is RISC really).» *USENET comp.arch newsgroup*, article 46782, February 1995.
- MASS97 Massiglia, P. *The RAID Book: A Storage System Technology Handbook*. St. Peter, MN: The Raid Advisory Board, 1997.
- MAYB84 Mayberry, W., and Eland, G. «Cache Boosts Multiprocessor Performance.» *Computer Design*, November 1984.
- MCEL85 McEliece, R. «The Reliability of Computer Memories.» *Scientific American*, January 1985.
- MEE96a Mee, C., and Daniel, E. eds. *Magnetic Recording Technology*. New York: McGraw-Hill, 1996.
- MEE96b Mee, C., and Daniel, E. eds. *Magnetic Storage Handbook*. New York: McGraw-Hill, 1996.
- MIRA92 Mirapuri, S.; Woodacre, M.; and Vasseghi, N. «The MIPS R4000 Processor.» *IEEE Micro*, April 1992.
- MOOR65 Moore, G. «Cramming More Components Onto Integrated Circuits.» *Electronics Magazine*, April 19, 1965.
- MORG92 Morgan, D. *Numerical Methods*. San Mateo, CA: M&T Books, 1992.
- MORS78 Morse, S.; Pohlman, W.; and Ravenel, B. «The Intel 8086 Microprocessor: A 16-bit Evolution of the 8080.» *Computer*, June 1978.

- MOTO97** Motorola, Inc. *PowerPC Microprocessor Family: The Programming Environments for 32-bit Microprocessors*. Denver, CO, 1997.
- MYER78** Myers, G. «The Evaluation of Expressions in a Storage-to-Storage Architecture.» *Computer Architecture News*, June 1978.
- NAYF96** Nayfeh, B.; Olukotun, K.; and Singh, J. «The Impact of Shared Cache Clustering in Small-Scale Shared-Memory Multiprocessors.» *Proceedings of the Second International Symposium on High Performance Computer Architecture*, 1996.
- NCR90** NCR Corp. *SCSI: Understanding the Small Computer System Interface*. Englewood Cliffs, NJ: Prentice Hall, 1990.
- NORM98** Normoyle, K., et al. «UltraSPARC-II: Expanding the Boundaries of a System on a Chip.» *IEEE Micro*, March/April 1998.
- NOVI93** Novitsky, J.; Azimi, M.; and Ghaznavi, R. «Optimizing Systems Performance Based on Pentium Processors.» *Proceedings COMPCON '92*. February 1993.
- OBER97a** Oberman, S., and Flynn, M. «Design Issues in Division and Other Floating-Point Operations.» *IEEE Transactions on Computers*, February 1997.
- OBER97b** Oberman, S., and Flynn, M. «Division Algorithms and Implementations.» *IEEE Transactions on Computers*, August 1997.
- OMON94** Omondi, A. *Computer Arithmetic Systems: Algorithms, Architecture and Implementations*. Englewood Cliffs, NJ: Prentice Hall, 1994.
- PADE81** Padegs, A. «System/360 and Beyond.» *IBM Journal of Research and Development*, September 1981.
- PADE88** Padegs, A.; Moore, B.; Smith, R.; and Buchholz, W. «The IBM System/370 Vector Architecture: Design Considerations.» *IEEE Transactions on Communications*, May 1988.
- PAPW96** Papworth, D. «Tuning the Pentium Pro Microarchitecture.» *IEEE Micro*, April 1996.
- PARK89** Parker, A., and Hamblen, J. *An Introduction to Microprogramming with Exercises Designed for the Texas Instruments SN74ACT8800 Software Development Board*. Dallas, TX: Texas Instruments, 1989.
- PATT82a** Patterson, D., and Sequin, C. «A VLSI RISC.» *Computer*, September 1982.
- PATT82b** Patterson, D., and Piepho, R. «Assessing RISCs in High-Level Language Support.» *IEEE Micro*, November 1982.
- PATT84** Patterson, D. «RISC Watch.» *Computer Architecture News*, March 1984.
- PATT85a** Patterson, D. «Reduced Instruction Set Computers.» *Communications of the ACM*, January 1985.
- PATT85b** Patterson, D., and Hennessy, J. «Response to 'Computers, Complexity, and Controversy.'» *Computer*, November 1985.
- PATT88** Patterson, D.; Gibson, G.; and Katz, R. «A Case for Redundant Arrays of Inexpensive Disks (RAID).» *Proceedings ACM SIGMOD Conference of Management of Data*, June 1988.
- PATT98** Patterson, D., and Hennessy, J. *Computer Organization and Design: The Hardware/Software Interface*. San Mateo, CA: Morgan Kaufmann, 1998.
- PELE97** Peleg, A.; Wilkie, S.; and Weiser, U. «Intel MMX for Multimedia PCs.» *Communications of the ACM*, January 1997.
- PFIS98** Pfister, G. *In Search of Clusters*. Upper Saddle River, NJ: Prentice Hall, 1998.
- POPE91** Popescu, V., et al. «The Metallow Architecture.» *IEEE Micro*, June 1991.
- POTT94** Potter, T., et al. «Resolution of Data and Control-Flow Dependencies in the PowerPC 601.» *IEEE Micro*, October 1994.
- PRIN91** Prince, B. *Semiconductor Memories*. New York: Wiley, 1991.
- PRIN96** Prince, B. *High Performance Memories: New Architecture DRAMs and SRAMs. Evolution and Function*. New York: Wiley, 1996.

## Bibliografia

- PRZY88 Przybylski, S.; Horowitz, M.; and Hennessy, J. «Performance Trade-offs in Cache Design.» *Proceedings. Fifteenth Annual International Symposium on Computer Architecture*, June 1988.
- PRZY90 Przybylski, S. «The Performance Impact of Block Size and Fetch Strategies.» *Proceedings. 17th Annual International Symposium on Computer Architecture*, May 1990.
- PRZY94 Przybylski, S. *New DRAM Technologies*. Sebastopol, CA: MicroDesign Resources, 1994.
- RADI83 Radin, G. «The 801 Minicomputer.» *IBM Journal of Research and Development*, May 1983.
- RAGA83 RAGAN-KELLEY, R., AND CLARK, R. «APPLYING RISC THEORY TO A LARGE COMPUTER.» *COMPUTER DESIGN*, NOVEMBER 1983.
- RAUS80 Rauscher, T., and Adams, P. «Microprogramming: A Tutorial and Survey of Recent Developments.» *IEEE Transactions on Computers*, January 1980.
- RECH98 Reches, S., and Weiss, S. «Implementation and Analysis of Path History in Dynamic Branch Prediction Schemes.» *IEEE Transactions on Computers*, August 1998.
- ROSC97 Rosch, W. *Winn L. Rosch Hardware Bible*. Indianapolis, IN: Sams, 1997.
- SATY81 Satyanarayanan, M., and Bhandarkar, D. «Design Trade-Offs in VAX-11 Translation Buffer Organization.» *Computer*, December 1981.
- SCHA97 Schaller, R. «Moore's Law: Past, Present, and Future.» *IEEE Spectrum*, June 1997.
- SCHM97 Schmidt, F. *The SCSI Bus and IDE Interface*. Reading, MA: Addison-Wesley, 1997.
- SEBE76 Sebern, M. «A Minicomputer-compatible Microcomputer System: The DEC LSI-11.» *Proceedings of the IEEE*, June 1976.
- SEGE91 Segee, B., and Field, J. *Microprogramming and Computer Architecture*. New York: Wiley, 1991.
- SERL86 Serlin, O. «MIPS, Dhrystones, and Other Tales.» *Datumation*, June 1, 1986.
- SHAN38 Shannon, C. «Symbolic Analysis of Relay and Switching Circuits.» *AIEE Transactions*, vol. 57, 1938.
- SHAN95a Shanley, T. *PowerPC System Architecture*. Reading, MA: Addison-Wesley, 1995.
- SHAN95b Shanley, T., and Anderson, D. *PCI Systems Architecture*. Richardson, TX: Mindshare Press, 1995.
- SHAN98 Shanley, T. *Pentium Pro and Pentium II System Architecture*. Reading, MA: Addison-Wesley, 1998.
- SHAR97 Sharma, A. *Semiconductor Memories: Technology, Testing, and Reliability*. New York: IEEE Press, 1997.
- SHER84 Sherburne, R. *Processor Design Tradeoffs in VLSI*. Ph.D. thesis, Report No. UCB/CSD 84/173. University of California at Berkeley, April 1984.
- SIEW82 Siewiorek, D.; Bell, C.; and Newell, A. *Computer Structures: Principles and Examples*. New York: McGraw-Hill, 1982.
- SIMA97 Sima, D. «Superscalar Instruction Issue.» *IEEE Micro*, September/October 1997.
- SIMO69 Simon, H. *The Sciences of the Artificial*. Cambridge, MA: MIT Press, 1969.
- SMIT82 Smith, A. «Cache Memories.» *ACM Computing Surveys*, September 1982.
- SMIT87 Smith, A. «Line (Block) Size Choice for CPU Cache Memories.» *IEEE Transactions on Communications*, September 1987.
- SMIT89 Smith, M.; Johnson, M.; and Horowitz, M. «Limits on Multiple Instruction Issue.» *Proceedings. Third International Conference on Architectural Support for Programming Languages and Operating Systems*, April 1989.
- SMIT95 Smith, J., and Sohi, G. «The Microarchitecture of Superscalar Processors.» *Proceedings of the IEEE*, December 1995.

- SODE96 Soderquist, P., and Leeser, M. «Area and Performance Tradeoffs in Floating-Point Divide and Square-Root Implementations.» *ACM Computing Surveys*, September 1996.
- SOHI90 Sohi, G. «Instruction Issue Logic for High-Performance Interruptable, Multiple Functional Unit Pipelined Computers.» *IEEE Transactions on Computers*, March 1990.
- SOLA94 Solari, E., and Willse, G. *PCI Hardware and Software: Architecture and Design*. San Diego, CA: Annabooks, 1994.
- STAL97 Stallings, W. *Data and Computer Communications*, 5th ed. Upper Saddle River, NJ: Prentice Hall, 1997.
- STAL98 Stallings, W. *Operating Systems. Internals and Design Principles*, 3rd ed. Upper Saddle River, NJ: Prentice Hall, 1998.
- STEN90 Stenstrom, P. «A Survey of Cache Coherence Schemes of Multiprocessors.» *Computer*, June 1990.
- STEV64 Stevens, W. «The Structure of System/360, Part II: System Implementation.» *IBM Systems Journal*, Vol. 3, No. 2, 1964. Reprinted in [SIEW82].
- STON93 Stone, H. *High-Performance Computer Architecture*. Reading, MA: Addison-Wesley, 1993.
- STRE78 Strecker, W. «VAX-11/780: A Virtual Address Extension to the DEC PDP-11 Family.» *Proceedings. National Computer Conference*, 1978.
- STRE83 Strecker, W. «Transient Behavior of Cache Memories.» *ACM Transactions on Computer Systems*, November 1983.
- STRIT79 Stritter, E., and Gunter, T. «A Microprocessor Architecture for a Changing World: The Motorola 68000.» *Computer*, February 1979.
- SWAR90 Swartzlander, E., editor. *Computer Arithmetic. Volumes I and II*. Los Alamitos, CA: IEEE Computer Society Press, 1990.
- TABA91 Tabak, D. *Advanced Microprocessors*. New York: McGraw-Hill, 1991.
- TAMI83 Tamir, Y., and Sequin, C. «Strategies for Managing the Register File in RISC.» *IEEE Transactions on Computers*, November 1983.
- TANE78 Tanenbaum, A. «Implications of Structured Programming for Machine Architecture.» *Communications of the ACM*, March 1978.
- TANE90 Tanenbaum, A. *Structured Computer Organization*. Englewood Cliffs, NJ: Prentice Hall, 1990.
- THOM94 Thompson, T., and Ryan, B. «PowerPC 620 Soars.» *Byte*, November 1994.
- TI90 Texas Instruments Inc. *SN74ACT880 Family Data Manual*. SCSS006C, 1990.
- TJAD70 Tjaden, G., and Flynn, M. «Detection and Parallel Execution of Independent Instructions.» *IEEE Transactions on Computers*, October 1970.
- TOMA93 Tomasevic, M., and Milutinovic, V. *The Cache Coherence Problem in Shared-Memory Multiprocessors: Hardware Solutions*. Los Alamitos, CA: IEEE Computer Society Press, 1993.
- TOON81 Toong, H., and Gupta, A. «An Architectural Comparison of Contemporary 16-Bit Microprocessors.» *IEEE Micro*, May 1981.
- TREM96 Tremblay, M., et al. «VIS Speeds New Media Processing.» *IEEE Micro*, August 1996.
- TUCK67 Tucker, S. «Microprogram Control for System/360.» *IBM Systems Journal*, No. 4, 1967.
- TUCK87 Tucker, S. «The IBM 3090 System Design with Emphasis on the Vector Facility.» *Proceedings, COMPCON Spring '87*, February 1987.
- VOEL88 Voelker, J. «The PDP-8.» *IEEE Spectrum*, November 1988.
- VOGL94 Vogley, B. «800 Megabyte Per Second Systems Via Use of Synchronous DRAM.» *Proceedings, COMPCON '94*, March 1994.

## Bibliografia

- VONN45 Von Neumann, J. *First Draft of a Report on the EDVAC*. Moore School, University of Pennsylvania, 1945.
- VRAN80 Vranesic, Z., and Thurber, K. «Teaching Computer Structures.» *Computer*, June 1980.
- WALL85 Wallach, P. «Toward Simpler, Faster Computers.» *IEEE Spectrum*, August 1985.
- WALL90 Wallis, P. *Improving Floating-Point Programming*. New York: Wiley, 1992.
- WALL91 Wall, D. «Limits of Instruction-Level Parallelism.» *Proceedings, Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, April 1991.
- WARD90 Ward, S., and Halstead, R. *Computation Structures*. Cambridge, MA: MIT Press, 1990.
- WEIN75 Weinberg, G. *An Introduction to General Systems Thinking*. New York: Wiley, 1975.
- WEIS84 Weiss, S., and Smith, J. «Instruction Issue Logic in Pipelined Supercomputers.» *IEEE Transactions on Computers*, November 1984.
- WEIS94 Weiss, S., and Smith, J. *POWER and PowerPC*. San Francisco: Morgan Kaufmann, 1994.
- WHIT97 Whitney, S., et al. «The SGI Origin Software Environment and Application Performance.» *Proceedings, COMPCON Spring '97*, February 1997.
- WILK51 Wilkes, M. «The Best Way to Design an Automatic Calculating Machine.» *Proceedings, Manchester University Computer Inaugural Conference*, July 1951.
- WILK53 Wilkes, M., and Stringer, J. «Microprogramming and the Design of the Control Circuits in an Electronic Digital Computer.» *Proceedings of the Cambridge Philosophical Society*, April 1953. Reprinted in [SIEW82].
- WILL90 Williams, F., and Steven, G. «Address and Data Register Separation on the M68000 Family.» *Computer Architecture News*, June 1990.
- WILS84 Wilson, K. «Science, Industry, and the New Japanese Challenge.» *Proceedings of the IEEE*, January 1984.
- YEAG96 Yeager, K. «The MIPS R10000 Superscalar Microprocessor.» *IEEE Micro*, April 1996.
- YEH91 Yeh, T., and Patt, Y. «Two-Level Adaptive Training Branch Prediction.» *Proceedings, 24th Annual International Symposium on Microarchitecture*, 1991.

# Índice alfabético

1394 Trade Association, 214

## A

Acceso directo, 94

Acceso directo a memoria (DMA), 174, 183, 195-197  
funcionamiento del DMA, 195-197  
inconvenientes de E/S programadas y mediante interrupciones, 195  
robo de ciclo, 195

Acceso uniforme a memoria (UMA), 623

ACM, Grupo de Interés Especial en Arquitectura de Computadores del, 16

ACM, Grupo de Interés Especial en Sistemas Operativos del, 258

Acumulador (AC) y multiplicador cociente (MQ), 23

Aislada, E/S, 185

Aleatorio, acceso, 94

Aleatorio, memoria de acceso (RAM), 95, 99

Aleatorio, semiconductor de acceso, tipos de memoria, 99-101

Álgebra booleana, 646-647

Almacenamiento, ver también memorización

Almacenamiento de datos, 7-8, 30

características físicas del, 95  
instrucciones para, 316

ALU, ver también Unidad aritmético-lógica.

Ampliaciones de depuración (DE), 423

Análisis del flujo de datos, 40

Antidependencia, 490

Apilamiento, lista de, 371

Arbitraje de bus, 190-191

Arbitraje oculto, 84

Aritmética del computador, 265-310  
aritmética en coma flotante, 293-302  
aritmética entera o con enteros, 273-287.

representación de números enteros, 267-273

representación en coma flotante, 287-293

unidad aritmético-lógica (ALU), 267

Aritmética en coma flotante, 293-302

consideraciones sobre precisión, 297-300

bols de guarda, 297

redondeo, 298-300

División, 297

multiplicación, 297

normalización IEEE 754, 300-302

infinito, 300

números denormalizados, 301-302

señalizaciones NaN silenciosas, 301

resta, 294-297

Suma, 294-297

Aritméticas de desplazamiento, operaciones, 330

Aritméticas, operaciones, 327

Aritmético-lógica, unidad (ALU), 4, 10-11, 267, 396

Arquitectura de computadores

definición de, 5

ASCII (American Standard Code for Information Interchange), 177, 178, 179

caracteres de control, 179

Asignación asociativa, 113-119

Asignación de lecturas informes, 685

Asignación directa, 115-118

Asignación, función, 115-121

Atributos arquitecturales, 5

Autoindexación, 370-371

## B

Ballistics Research Laboratory (BRL), 19

Biestable, D, 673

Biestable S-R sincronizado, 672

Biestables, 671-674, 675

cerrojo S-R, 671-672

D, 673

J-K, 673-674

S-R con reloj, 672

Bifurcación condicional, 332, 412

Bifurcación, instrucción de, 316

Bits de guarda, 297

Bloques, 147

Boole, George, 646

Booleanas, ecuaciones, 650

Boot, algoritmo de, 281-284

Buffer de traducción anticipada (TLB), 347

Bus a 64 bits, terminales de extensión a, 79

Bus de datos, anchura del, 37

Bus de tiempo compartido, 605-607

Bus del sistema, 39

bus de datos, 68

componentes del computador, 51-53

definición de, 67-68

diseño de elementos para, 50

estructura del, 68-70

estructuras de interconexión, 66

funcionamiento del computador, 54-66

interconexión al, 76

líneas de control, 68

líneas de datos, 68

líneas de dirección, 68

PCI, 76-84

Bus, conseguir el, 69

Bus, interconexiones en, 76

estructura de, 68-70

jerarquías de buses múltiples, 70-72

Bus, petición de, 69

Buses, elementos de diseño de, 72-76

anchura de, 74

métodos de arbitraje en, 73

temporización en, 73-74

tipos de transferencias de datos en, 74-76

tipos de, 72-73

Búsqueda, tiempo de, 152, 153

## Índice alfabético

jeza, 147  
e cohorte NUM  
EC-NUMAL 624  
DRAM (CDRAM), 130  
inhabilitada (CD), 422  
memoria, 123-124  
coherencia de, 613-619  
rotóculos de sondeo, 615  
soluciones hardware, 614-615  
rotóculos de directorio, 614-615  
soluciones software, 613-614  
lo de operandos (CO), 407  
uso de modo, 365  
los de datos, 27  
idad de memoria, 93-94  
tación de dato, 395  
ción de instrucción (FI), 395, 406  
cación de operandos (FO), 407  
ción, solapamiento de la, 406  
co de registros amplio  
cache frente a, 447-448  
o de un, 444-448  
variables globales, 447  
ntanas de registros, 445-447  
de instrucciones, 464-467  
reptorio de instrucciones, 461-464  
roversia RISC/CISC, 473-474  
mentos clave de los, 439  
S 8400, 460-467  
presentación de cauce en, 457-460  
na instrucciones regulares, 457-459  
timización de la, 459-460  
ARC, 468-473  
njunto de registros, 468  
ormato de instrucción, 472-473  
ertorio de instrucciones, 468-472  
finición de, 164  
AM, 130  
MI, 163-166  
sidades en un, 165  
ato de bloque, 165  
mización de datos en un, 165-166  
idad angular constante (CAV), 164  
idad linear constante (CLV), 164  
tajas del, 165-166  
bus, 73  
de captación, 23, 54-58, 403, 534-  
e ejecución, 34, 54, 404, 537-538  
de reloj, 73  
astro, 213  
robo de, 195  
s combinacionales, 650-670  
lógico programable, 664-667  
dificadores, 662-664  
iones hookeanas, 650  
lementación de funciones booleanas,  
-652  
umentación NAND y NOR, 660  
s de Karnaugh, 653-657  
na sólo lectura (ROM), 667-668  
odo de Quine-Mccluskey, 657-660  
hexores, 661-662  
los gráficos, 650  
ficción algebraica, 653  
tores, 668-670

tabla verdad, 650  
CISC (Computador de conjunto complejo de instrucciones), 442  
características comparativas entre los RISC y los, 455-457  
objetivos de los, 451-452  
Clusters, 600, 619-623  
balanceado o equilibrado de la carga en, 622-633  
configuraciones de, 619-622  
cuestiones relacionadas con el diseño de sistemas operativos para, 622-623  
disco compartido en, 622  
elemento secundario activo, 621  
especificaciones y objetivos de diseño, 619  
espera pasiva, 620-621  
gestión de fallos en, 622  
métodos de organización de, 621  
nada compartido en, 622  
Codificación funcional, 576  
Código Hamming, 107  
Códigos de condición (indicadores), 398  
Códigos de corrección de errores, 107  
Códigos de detección de errores, 181  
Cola a corto plazo, 237, 238  
Compactación, 242  
Comparaciones temporales, 154  
Compartidas, coches L2, 611-612  
Compartir nada, sin, 622  
Componente discreto, 28-29  
Componentes del computador, 31-53  
Comprobación de alineamiento (AC),  
Registros EFLAGS, 420-421  
Computación vectorial, 627-639  
encadenamiento, 631  
enfoques de la, 627-633  
posibilidades del IBM 3090, 633-639  
procesamiento paralelo, 628  
procesamiento vectorial, 628  
segmentación de cauce de operaciones, 631  
segmentación de cauce dentro de una operación, 630  
Computador completo, 600  
Computadores comerciales, 24-26  
Computadores de repertorio reducido de instrucciones (RISC)  
Ver también Paralelismo a nivel de instrucción en RISC, 483  
Computadores  
breve historia de los, 19-38  
ENIAC (Electrónico Numerical Integrator And Computer), 19  
Máquina de von Neumann, 20-26  
primera generación de, 19-26  
computadores comerciales, 24-26  
segunda generación de, 26-28  
IBM 7094, 27-28  
tercera generación de, 28-34  
DEC PDP-8, 33-34  
IBM Sistema/360, 32-33  
Microelectrónica, 29-32  
últimas generaciones de, 34-39  
memoria de semiconductor, 36  
microprocesadores, 36-39  
Computadores, generaciones de, 26  
Comunicación con el dispositivo, 181  
Comunicación con el procesador, 180-181  
datos, 180  
decodificación de órdenes, 180  
informe de estado, 180  
reconocimiento de dirección, 181  
Comunicación de datos, 8  
Conexión en cadena, 190-191  
Conexión rápida, 209  
Contabilidad en el bloque de control del proceso, información de, 234  
Contador de programa en el bloque de control del proceso, 234  
Contadores, 576-680  
contador de onda, 677-678  
contadores síncronos, 678-680  
Contadores síncronos, 678-680  
Contexto de datos en el bloque de control del proceso, 234-235  
Control, 55  
tipo de instrucciones de, 316  
Controlador de bus, 73  
Controlador de disco, 178  
Coprocessador presente (MP), indicador de, 421  
Corrección de errores simples, código de (SEC), 111  
Correspondencia asociativa por conjuntos, 119-121  
Corto plazo, planificación a, 233-238  
estados de los procesos, 234-235  
técnicas de, 235-238  
CPU Info Center (página web), 16  
CPU, interconexiones en la, 11  
CPU, operando fuente y resultado y registro de la, 314  
CPU, ver también Unidad central de procesamiento  
Charles Babbage Institute, 46  
Chips, 31

D

Datos en buffer, 181  
De enteros, representación, 275-282  
conversión entre diferentes longitudes (número de bits), 271-273  
representación en coma fija, 273  
representación en complemento a dos, 269-271  
representación en signo y magnitud, 268  
DEC PDP-8, 33-34  
diagrama de bloques, 35  
Evolución, 34  
proyectos para enseñanza de, 683-685  
Densidad de datos, relación con la anchura de la capa de aire, 151  
Densidades en CD-ROM, 165  
Desbordamiento, 276  
Desbordamiento a cero del exponente, 294  
Desbordamiento de exponente, 305  
Desbordamiento hacia cero gradual, 302  
Desplazamiento para direccionamiento, 369-371  
direcciónamiento con registro-base, 370  
direcciónamiento relativo, 369  
Indexación, 370-371

- Desplazamiento, registros de, 675-676  
 Developer Home (página web), 46  
 Digital Equipment Corp. (DEC), 27  
 DEC PDP-8, 33-34  
 Dirección base, 242  
 Dirección efectiva (EA), 365  
 Direcciónamiento, 365-372  
 campo de modo de, 365  
 con desplazamiento, 369-371  
 dirección efectiva (EA), 365  
 directo, 367  
 indirecto, 367-368  
 inmediato, 367  
 modos de, 366  
 pila de, 371  
 registro de, 368-369  
 Direcciónamiento con registro-base, 370  
 Direcciónamiento de registros, 368-369  
 Direcciónamiento directo, 367  
 Disco de cabeza fija, 148  
 Disco óptico borrable, 164, 167  
 Disco Winchester, 151-152  
 formato de pistas en, 149  
 Discos de doble superficie, 150  
 Discos de superficie única (simple superficie), 150  
 Discos, parámetros de prestaciones, 152-154  
 comparación de tiempos, 154  
 retraso rotacional, 152, 153  
 tiempo de acceso, 152  
 tiempo de búsqueda, 152, 153  
 tiempo de transmisión de datos, 153  
 Dispositivos externos, 176-179  
 categorías de, 176  
 datos de los, 176-177  
 lógica de control de, 177  
 señales de control de, 177  
 señales de estado, 177  
 teclado/monitor, 177  
 transductor, 177  
 unidad de disco, 178  
 Diáquete, 151  
 División, 234-237  
 en aritmética de coma flotante, 297  
 en complemento a 2  
 ejemplos de, 287  
 DRAM mejorada, 129-130  
 DRAM síncrona (SDRAM), 130-131  
 DRAM, 41-42  
 cache DRAM (CDRAM), 130  
 DRAM mejorada, 129-130  
 RamLink, 132-134  
 Rambus DRAM (RDRAM), 131  
 síncrona (SDRAM), 130-131  
 DVD, 164, 167
- E
- E/S, 173-217  
 acceso directo a memoria (DMA), 174, 183, 195-197  
 canales de E/S y procesadores, 198-200  
 controladas por interrupciones, 174, 183, 186-194  
 dispositivos externos, 176-179  
 FireWire, 174  
 interfaz externa, 200-213
- módulos de, 175, 180-182  
 programadas, 174, 183-186  
 SCSI, 174  
 Ver también: Acceso directo a memoria (DMA);  
 Dispositivos externos  
 canales y procesadores  
 E/S controladas por interrupciones  
 E/S programada; SCSI  
 E/S (entrada/salida), 10  
 E/S desde o hacia memoria, 66  
 E/S, transferencia al procesador desde, 66  
 E/S, canales (procesadores) de, 182, 198-200  
 canal multiplexor, 199  
 canal selector, 199  
 características de los, 198-200  
 control, 199  
 evolución de la función de E/S, 198  
 multiplexor de bloques, 200  
 multiplexor de bytes, 199  
 E/S, cola de, 237  
 E/S, componentes de, 32  
 E/S, controlador/dispositivo de control de, 182  
 E/S, escritura en una, 69  
 E/S, funcionamiento, 65-66  
 evolución del, 198  
 E/S, indicadores del privilegio de (NOPL), registros EFLAGS, 421  
 E/S, información, en el bloque de control del proceso, del estado de, 234-235  
 E/S, lectura en una, 69  
 E/S, módulos de, 66, 175, 180-182  
 funcionamiento de, 180-181  
 almacenamiento temporal de datos, 181  
 comunicación con el dispositivo de E/S, 181  
 comunicación con el procesador, 180-181  
 control y temporización, 188  
 detección de errores, 181  
 estructura de los, 182  
 E/S, operando fuente y resultado de, 314  
 E/S, registro de dirección de (E/OAR), 53  
 Eckert, John Presper, 19, 24  
 EDRAM, 129-130  
 EDVAC (Electronic Discrete Variable Computer), 20  
 Ejecución, estado de proceso en, 234  
 Endian, 357-361  
 orden de los bits, 361  
 orden de los bytes, 357-361  
 ENIAC (Electronic Numerical Integrator And Computer), 19  
 Ensamblador, 347  
 Entera aritmética, 273-287  
 División, 234-237  
 Multiplicación, 277-284  
 Negación, 273-275  
 Resta, 273-276  
 Suma, 273-276  
 Enteros sin signo, 277-279  
 Entrada salida (E/S), 8  
 Entrada salida, instrucciones, 331  
 Error transitorio, 106
- Escribir operando (WO), 407  
 Escritura de datos, 395  
 Especulativa, ejecución, 40  
 Espera, estado de un proceso, 234  
 Estado, en el procesador parado (HALT), 234  
 Estado, en el bloque de control del proceso, 234-235  
 Estado, sechas de, 177  
 Estática, RAM, 99-100  
 Estructura de un computador, 6, 9-12  
 E/S (entradas/salidas), 10  
 interconexiones del sistema, 10  
 memoria principal, 10  
 Unidad central de procesamiento (CPU), 4, 9, 10-11  
 Estructura, 6, 9-11  
 evolucionada y prestaciones de los, 17-46  
 breve historia de los, 19-38  
 diseño buscando buenas prestaciones de los, 38-42  
 Pentium, 42-43  
 PowerPC, 42-43, 44-45  
 Extraible, disco, 148
- F
- Fallo permanente, 106  
 Fallo, recuperación después de un, 622  
 Fallo, transferencia por, 622  
 Familia, concepto de, 439  
 Fast Ethernet, 71  
 Fila, señales de dirección de selección de (RAS), 103  
 FireWire, 72, 174, 209-213  
 capa de enlace, 210, 212-213  
 capa de transacción, 210  
 capa física, 210-212  
 intervalos de equidad, 211  
 capas del protocolo, 210, 211  
 conexión rápida, 209-210  
 configuraciones, 209-210  
 intervalo de reconocimiento, 212  
 muestra de ciclo, 213  
 periodo de intervalo de subacción, 212  
 periodo de la secuencia de arbitraje, 212  
 periodo de reconocimiento, 212  
 periodo de transmisión de paquete, 212  
 subacción, 212  
 transmisión asíncrona, 212  
 transmisión isocrónica, 212  
 ventajas de, 209
- Física, ampliación de dirección (PAE), 423  
 Física, dedicación, 72-73  
 Física, dirección, 242  
 Flanco de subida de la señal, 83  
 Flujo de datos, 403-405  
 Formato de bloque en CD-ROM, 165-166  
 Función, 6, 7-9  
 control de, 8  
 de memorización de datos, 7-8  
 de procesamiento de datos, 7  
 de transferencia de datos, 7, 8  
 Funcionamiento de un computador, 6, 7-9, 34-66  
 ciclo de captación, 54-58

## Índice alfabético

do de ejecución, 54-58  
do de instrucción, 54  
ntró del, 8, 30  
iciones de E/S, 63-66  
errupciones, 58-63  
clases de, 58  
nhabilización de, 62-64  
múltiples, 62-65  
y el ciclo de instrucción, 60-62  
nsferencias de datos, 7-8, 30  
tonamiento de una unidad de control, 531-554  
ntrol del procesador, 539-550  
Intel 8085, 545-550  
organización interna del procesador, 544-545  
requisitos o funcionales, 539-540  
señales de control, 540-544  
plementación cableada, 550-553  
entradas de la unidad de control, 550-551  
lógica de la unidad de control, 551-553  
cro-operaciones, 533-539  
ciclo de captación, 534-536  
ciclo de ejecución, 537-538  
ciclo de instrucción, 538-539  
ciclo de interrupción, 536-537  
ndo indirecto, 536

or de interrupciones, 61  
computador SMP, 609-612  
he L3, 612  
hes L2 compartidas, 611-612  
reconexión conmutada, 610, 611  
escalas, integración a, 34

itación de test de máquina (MCE),  
ware, 51-52  
paginación, 245

Merced, 511-522  
ga especulativa, 519-522  
ceptos básicos subyacentes, 512  
ucción con predicados, 515-519  
naro de instrucción, 514-515  
ivación, 512-513  
nternazación, 513-514  
omputador, 20-24, 25  
o de captación, 23-24  
o de ejecución, 24  
o de instrucción, 23  
uctura del, 20-21  
rucción aritmética, 24, 25  
rucción de modificación de direcciones, 24, 25  
ndo del, 21-23  
torio de instrucciones, 24, 25  
transmisión) incondicional, 24, 25  
ferencia de datos, 24, 25  
ndo de control, 21

IBM 3090, ejecución de microinstrucciones en el, 581  
IBM 3090, posibilidades vectoriales, 633-639  
instrucciones compuestas, 637-638  
organización del, 633-635  
registros del, 635-637  
repertorio de instrucciones, 638-639  
IBM 7094, 27, 28  
IBM S/360 Modelo 85, 439  
IBM S/360, 32-33  
características clave de la familia, 32-33  
IBM S/370, arquitectura del, 5-6  
Identificador en el bloque de control del proceso, 234  
IEEE normalización, 754, 266, 291-293, 300-302, 303  
IEEE, Comité Técnico en Arquitectura de Computadores, 16  
If-then-else, instrucción, 516-519  
Incremento, operación de, 327  
Indexación, 370-371  
Indicador de dirección (ID) en registro EFLAGS, 421  
Indicador de enunciación (EM), 421  
Indicador de identificación (ID) en el registro EFLAGS, 421  
Indicador de trampa (TF) en registro EFLAGS, 420  
Indicadores, 397, 398  
Índices, registros, 370, 397  
Indirecto, ciclo, 402-403, 404, 536  
Indirecto, direccionamiento, 367-368  
Inhabilitación de marca de tiempo (TSD), 423  
Inicio (RESET), 69  
Inmediato, direccionamiento, 367  
Instrucción de modificación de dirección, 24-25  
Instrucción Decode (DI), 407  
Instrucción Execute (EJ), 407  
Instrucción, cálculo de dirección de, 57  
Instrucción, captación de (I), 537  
Instrucción, cause de, 418  
buffer de bucles, 413  
cause de la CPU de instrucciones en 6 etapas, 409  
en el Intel 80486, 417-418  
estrategia de segmentación, 405-410  
dujos múltiples, 412  
instrucciones de bifurcación condicional, 412  
precipitación del destino del salto, 412  
predicción de saltos, 411-417  
prestaciones de la segmentación, 410-411  
salto retardado, 417  
tratamiento de saltos (bifurcaciones), 425-432  
Instrucción, ciclo de código de (ICCL), 538  
Instrucción, ciclo de, 23, 52, 54-58, 401-405, 538-539  
ciclo indirecto, 402-403  
e interrupciones, 60-62, 63  
flujo de datos, 403-405  
Instrucción, formato de, 364  
Instrucción, longitud de, 377

Instrucción, operación de decodificación (iod), 57  
Instrucción, paralelismo a nivel de, 486-487  
Instrucción, registro de (IR), 23, 314, 399, 403  
Instrucción, registro temporal de (IBR), 23  
Instrucciones aritméticas, 24, 25, 316  
Instrucciones de bifurcación, ver también instrucciones de salto  
Instrucciones de control del sistema, 331  
Instrucciones de conversión, 330  
Instrucciones de longitud variable, 381-385  
Instrucciones de salto, ver también instrucciones de bifurcación  
Instrucciones de test, 316  
Instrucciones, precipitación de, 406  
Instrucciones, repertorio de, 24, 25  
Instrucciones, repertorios de 311-361  
características de las instrucciones máquinas, 313-319  
ódigo de operación (opcode), 312, 313  
Direccionamiento, 319, 365-371  
diseño de, 318-319  
formatos de instrucciones, 318, 364, 376-385  
instrucciones de longitud variable, 381-385  
lenguaje ensamblador, 346-347  
longitud de instrucciones, 377  
modos formatos de direccionamiento, 363-391  
número de direcciones, 316-318  
operaciones comunes, 324-325  
posición de bits, 378-381  
procesadores big-endian-little-endian/bi-endian, 312  
referencia a la siguiente instrucción, 312, 313  
referencia a operando, 364  
repertorio de operaciones, 318  
representación de instrucciones, 314-315  
tipos de datos, 319  
tipos de instrucciones, 315-316  
tipos de operaciones, 324-326  
aritméticas, 327  
control de flujo, 331-336  
control del sistema, 331  
conversión, 330  
entrada/salida, 331  
lógicas, 328-330  
transferencia de datos, 326-327  
tipos de operandos, 319-321  
Instrucciones SIMD, 688, 690  
Instrucciones, ventana de, 489  
Integración a muy gran escala (VLSI), 34  
Integrados, circuitos, 28-34  
DEC PDP-8, 33-34  
IBM S/360, 32-33  
memoria semiconductor, 36  
Microelectrónica, 29-32  
Microprocesadores, 36-38  
Intel 80486, segmentación del, 417-418  
Intel 8085, 545-550  
Intel 82C55A, programable, 200-203  
dispositivo periférico, 192-195  
Intel Technology Journal, 16

Intel, evolución de los microprocesadores de, 37-38

Intercambiabilidad, 238-240

Interconexión del sistema, 10

Interconexión, estructuras de, 66

memoria, 66

módulo de E/S, 66

procesador, 66

soporte de transferencias, 66

Interfaces, tipos de, 200

Interfaz externa, 200-213

configuraciones punto a punto y multi-punto, 201

FireWire, 209-213

SCSI (interfaz para computadores pequeños), 201-209

tipos de, 200

Interfaz, terminales de control de, 77, 78

Interior en el chip, cache, 124

Interior, memoria, 91-144

cache, 93, 111-124, 439

memoria principal semiconductor, 99-11

organización de DRAM, 129-134

organización de la cache del Pentium II, 125-128

organización de la cache del PowerPC, 128-129

Interior, organización del procesador, 544-545

Interior, bus de la CPU, 396

Interprete de instrucciones, 395

Interrupción ACK, 69

Interrupción vectorizada, 191

Interrupción, ciclo de, 402, 404, 536-537

Interrupción, petición de, 60, 69

Interrupciones inhabilitadas, 62

Interrupciones, 58-65

clases de, 58

inhabilitación de, 62

múltiples, 62-65

Interrupciones, E/S realizada por medio de, 174, 183, 186, 195

controlador de interrupciones Intel 82C55A, 192-195

controlador de interrupciones Intel 82C59A, 191, 192

cuestiones de diseño, 190-191

desventajas de las, 195

procesamiento de interrupciones, 187-190

Interrupciones, indicador de habilitación de (IFL, registro EFLAGS, 421

Interrupciones, procesamiento por el Pentium II de, 424-427

gestión de interrupciones, 425-427

interrupciones y excepciones, 424-425

tabla de vectores de interrupción, 425

Interrupciones, terminales de, 80

Investigación, proyectos de, 684

Isócrono, intervalo, 213

J

J-K, biestable, 673-674

K

Karnaugh, mapas de, 653-657

## L

L3, cache, 612

Largo plazo, cola a, 237-238

Largo plazo, planificación a, 233

Lazos, bufer de, 412-413

Lectura, memoria de solo (ROM), 95, 100-101, 667-668

Leer después de escribir, 75

Leer/Modificar/Escribir, operación de, 75

Lenguaje ensamblador, 346-347

Línea, tamaño de, 123

Líneas de control, 68-69

Líneas de datos, 67

Líneas de direccionamiento, 68, 103

Little-endian, procesadores, 312, 361  
Ver también endian

Local, redes de área (LAN), 70

Localidad de las referencias, 98

Lógica de control, 177

Lógica del chip, 102-104

Lógica digital, 645-682

álgebra booleana, 646-647

circuitos combinacionales, 650-670

circuitos secuenciales, 671-680

puertas, 648-650

Lógica, dirección, 242

Lógicas, instrucciones, 316

Lógicas, operaciones, 328-330

LSI-11, secuenciamiento de direcciones en el, 571

LSI-11, ejecución de microinstrucciones en el, 578-581

formato de microinstrucciones, 580-581

organización de la unidad de control, 578-580

## M

Magnética, cinta, 168-169

Magnética, memorias de superficie, 95

Magnético, disco, 147-154

Bloques, 147

cabeza, 147

características físicas, 148-152

densidad, 147

disco de cabezas móviles, 148

disco intercambiable, 148

disco no intercambiable, 148

discos de cabezas fijas, 148

discos de doble superficie, 150

discos de platos múltiples, 150, 151

discos de una superficie, 150

disquete, 151

organización de los datos, 148

organización/formato de los datos, 147-148

paquete de discos, 150

parámetro de prestaciones, 152-154

parámetros de prestaciones, 152-154

pistas, 147

sectores, 147

Magnético-optico (MO) discos, 164, 168

Mancha, desbordamiento a cero de la, 294

Manita, desbordamiento de la, 294

Máquina, características de las instrucciones, 313-319

Máquina, ciclo, 453

Máquina, paralelismo, 486-487, 491-492

Marcos, 242

Máscara de alineamiento, 422

Mauchly, John, 19, 24

Medio plazo, planificación a, 233

Memoria asociativa, 94-95

Memoria cache, 95, 111-124, 439

algoritmos de reemplazo o de sustitución, 121-122

diseño de los elementos de la, 113-124

función de correspondencia, 115-121

correspondencia asocialiva, 118-119

correspondencia directa, 115-118

interna en el chip del procesador, 124

número de cachés, 123-124

organización en el Pentium II de la, 125-128

organización en el PowerPC de la, 128-139

política de escritura en la, 123-123

principios de la, 111-113

tamaño de la, 114

tamaño de línea, 123

unificada, 124

Memoria de control, 559

Memoria en un computador, sistemas de, 93-99

características clave, 93-95

jerarquía de los, 95-99

Memoria externa, 145-171

cinta magnética, 168-169

disco magnético, 147-154

óptica, 163-168

RAID, 155-163

Ver también Disco magnético; Memoria óptica; RAID

Memoria flash, 101

Memoria no segmentada ni paginada, 250

Memoria paginada no segmentada, 250

Memoria sólo lectura borrable y programable (EPROM), 101

Memoria sólo lectura programable y borrable eléctricamente (EEPROM), 101

Memoria virtual, 243-246

demandas de página, 243-245

estructura de la tabla de páginas, 245-246

y operando fuente y resultado, 313

Memoria, 53, 66

dispositivos físicos de, 95

memoria de acceso aleatorio (RAM), 95

memoria de solo lectura (ROM), 95

métodos de acceso a, 94-95

prestaciones de la, 95

unidad de transferencia, 94

Ver también tipos de memoria de computador, capacidad, 93-94

Memoria, E/S asignadas en, 185

Memoria, escritura en, 69

Memoria, gestión de, 231, 238-249, 253-254

buffer de traducción rápida (TLB), 247

Definición, 238

intercambio con disco, 238-240

memoria virtual, 243-246

Paginación, 242-243

Particiones, 240-242

Segmentación, 247-249

## Índice alfabético

iria, instrucciones de. 316  
iria, jerarquía. 95-99  
iria, lectura de. 69  
iria, módulo de. 53  
iria, punteros en el bloque de control del proceso. 234  
iria, registro de dirección de (MAR). 53, 399, 579  
iria, registro temporal de (MBR). 13, 399  
iria, señales de. 58  
iria, tiempo de ciclo de. 27, 95  
iria, transferencia de procesador a. 68  
rización, se *también* almacenamiento de transferencia sincrona de datos. 206  
n de comprobar si la unidad está lata. 208  
orización. 203-206  
iones. 201  
protocolo. 616-619  
to de escritura. 618  
to de lectura. 617  
istancia entre cachés L1-L2. 618-9  
lo compartido. 616  
lo exclusivo. 616  
lo modificado. 616  
lo no válido. 616  
de escritura. 617-618  
de lectura. 616-617  
computadores. 6  
electrónica. 29-32  
instrucción horizontal. 558, 575  
instrucción, bus (MIB). 578-579  
instrucción, ejecución de. 566, 571-  
secuencia de microinstrucción. 575-8  
3033, 581, 582  
11, 578-581  
mato de microinstrucción. 580, 581  
paritación de la unidad de control. 578-580  
ramificación de microinstrucciones. 572-3  
instrucción, generación de dirección. 59-570  
instrucciones verticales. 558, 561.  
instrucciones. 557-559  
iria de control. 559  
instrucción horizontal. 558  
instrucción vertical. 558  
ras de control. 557-558  
instrucciones, espacio de. 574  
instrucciones, secuenciamiento. 566-  
jerarquías de diseño. 566  
acción de direcciones. 569-570  
I, 571  
as de secuenciamiento. 566-568  
secciones. 533-539  
procesador, velocidad de. 39-40

Microprocesadores. 36-38  
evolución de los de INTEL. 37-38  
Microprograma, definición de. 556, 557  
Microprogramación, lenguaje para. 557  
Microprogramada, implementación. 11  
Microprogramada, unidad de control. 439, 556, 559-562  
decodificador. 561  
lectura de una microinstrucción en la memoria de control. 560  
microarquitectura. 560  
microinstrucciones verticales. 561  
organización de la. 572  
registro de dirección de control. 559  
registro intermedio de control. 560  
unidad de control cableada. 565  
Microprogramado, control. 555-596  
aplicaciones de la microprogramación. 594-595  
control de Wilkes. 362-365  
ejecución de microinstrucción. 566  
microinstrucciones. 571-582  
secuenciamiento de microinstrucciones. 566-571  
T18800. 583-593  
unidad de control microprogramada. 556, 559-562  
ventajas/desventajas de. 565-566  
Microprogramación, aplicaciones de la. 594-595  
MIPS R10000. 507-509  
preddecodificación. 507-508  
MIPS R4000. 460-467  
cauce de instrucción. 464-467  
repertorio de instrucciones. 461-464  
MMX, tecnología. 341-343  
repertorio de instrucciones. 342  
Modularidad. 333  
Monitor. 177  
Moore, Gordon. 31-32  
Moore, ley de. 31-32  
Motorola MC68000, 400  
Móviles, disco de cabezas. 148  
Múltiples instrucciones, dato único (MUSD), secuencia de. 602  
Múltiples instrucciones, secuencia de múltiples datos (MIMD), 602  
en sistemas de procesadores paralelos. 601-603  
Múltiples, discos de platos. 150  
Múltiples, flujos. 412  
Múltiples, interrupciones. 62-65  
Múltiples, jerarquía de buses. 70-72  
Múltiples, líneas de interrupción. 190  
Múltiples, organizaciones de procesadores. 601-603  
Múltiples, procesadores. 439  
Multiplexación en el tiempo. 72  
Multiplexor, canal. 199  
Multiplexores. 27, 661-662  
Multiplicación. 277-284  
acceso no uniforme a memoria (NUMA). 623-627  
motivación. 624  
organización. 624-626  
pros/contras. 626-627  
terminología. 623-624  
bus de tiempo compartido. 605-607  
clusters. 619-623  
configuraciones. 619-622  
cuestiones relacionadas con el diseño de sistemas operativos. 622-623  
enfoque de disco compartido. 622  
equilibrado de carga. 622-623  
especificaciones de objetivos/diseño. 619  
espera pasiva. 620  
gestión de fallos. 622  
métodos usados en cluster. 621  
secundario activo. 621  
sin compartir nada. 622  
SMP en comparación con. 623  
computación vectorial. 627-639  
encadenamiento. 631  
enfoque de la. 627-633  
operaciones a través del cauce segmentado. 631  
posibilidades vectoriales del IBM 3090. 633-639  
procesamiento paralelo. 628-629  
procesamiento vectorial. 627-633  
segmentación de cauce dentro de una operación. 630  
crecimiento incremental. 604  
de enteros sin signos. 277-279  
en aritmética en coma flotante. 297  
en complemento a dos. 279-284  
escalado. 604  
gran computador SMP. 609-612  
cache L3. 612  
caches L2 compartidas. 611-612  
comunicados de interconexión. 611  
memoria multipuerto. 607-608  
organización. 605-608  
clasificación de. 605  
prestaciones. 604  
protocolo MESI. 616-619  
acuerdo de escritura. 618  
acuerdo de lectura. 617  
consistencia entre cachés L1-L2. 618-619  
estado compartido. 616  
estado exclusivo. 616  
estado modificado. 616  
estado no válido. 616  
fallo de escritura. 617-618  
fallo de lectura. 616-617  
sistema operativo multiprocesador. 609  
unidad central de control. 608  
Multiplicación en complemento a dos. 279-284  
Multiprocesadores simétricos (SMP). 600, 601, 604-612  
coherencia de cache. 613-619  
protocolos de directorio. 614-615  
protocolos de sondeo. 615  
soluciones hardware. 614-615  
soluciones software. 613-614  
disponibilidad. 604  
ventajas sobre arquitecturas monoprocesador. 604  
Multiprogramación. 228, 229  
Multitarea. 229  
MYSN, señal de sincronización de maestro. 73

- N  
 Negación, inversión, 273-275  
 Negate, operación de, 327  
 Negativo, desbordamiento, 289  
 Negativos, desbordamiento a cero, 289  
 Nemotómicos, 315  
 No escritura inmediata (en Pentium) (NWI), 422  
 No extraible, disco, 148  
 No uniforme, acceso a memoria (NUMA), 600, 601, 602, 623-627  
 definición, 623-624  
 motivación, 624  
 multiprocesadores simétricos (SMP), 623-627  
 organización, 624-626  
 pros/contras del sistema CC-NUMA, 626-627  
 términos, 623-624  
 No utilizable como cache, memoria, 123  
 Núcleo magnético, 36  
 Nuevo, estado de proceso, 234  
 NUMA, ver también Acceso a memoria no uniforme  
 Numérico, error, 422  
 Números denormalizados en IEEE 754, 301-302  
 Números en coma flotante, 266
- O  
 Oblea de silicio, 31  
 Onda, contador de, 677-678  
 Opcodes (códigos de operación), 312  
 Open Group Research Institute Operating System Program, 258  
 Operación absolute, 327  
 Operación cíclico de desplazamiento rotación, 329, 330  
 Operación de decrementar, 327  
 Operaciones con datos, 58  
 Operaciones de transferencia de control bifurcación, 331-336  
 instrucciones de llamada a procedimientos, 333-336  
 instrucciones de ramificación, 332-333  
 instrucciones de salto implícito, 333  
 Operando, cálculo de direcciones de (loc), 57  
 Operando, captación de (lof), 58  
 Operando memorización de (ost), 58  
 Operando, referencias a, 364  
 Operandos  
 tipos de, 319-321  
 caracteres, 320-321  
 datos lógicos, 321  
 números, 319-320  
 Operandos, referencias a, 312  
 Óptica, memoria, 163-168  
 CD-ROM, 163-166  
 disco óptico borrable, 164, 167  
 discos magnéticos-opticos, 164, 168  
 video disco digital (DVD), 167  
 WORM, 164, 166  
 Optimización de registros por el compilador, 448-450  
 Orden de solicitud de situación, 208  
 fase de reselección, 202  
 fase selección, 202  
 fase estado, 202  
 líneas de control SCSI-1, 203  
 orden de enviar diagnóstico, 208  
 señales y fases, 201-203  
 Organización comparada con arquitectura, 5-6  
 Organización de datos en un CD-ROM, 165  
 Organización de un computador comparada con su arquitectura, 5-6  
 proyectos de enseñanza de, 683-685
- P  
 Página global, habilitación de (PGE), 423  
 Página, ampliación del tamaño de (PSE), 423  
 Página, estructura de la tabla de, 245-246  
 Página, fallo de, 244  
 Página, tabla de, 243  
 Paginación, 242-243  
 Paginación (PGI), indicador de, 422  
 Paginación por demanda, 243-245  
 Páginas, 242  
 Palabra, 94  
 síndrome, 108  
 Palabras de control, 557-558  
 Paquete de disco, 150  
 Paralelo, procesamiento, 599-643  
 multiprocesadores simétricos (SMP), 600, 601, 604-612  
 organizaciones de procesadores múltiples, 601-603  
 organizaciones paralelas, 603  
 tipos de sistemas de procesadores paralelos, 601-603  
 Paralelos, procesadores, 632  
 Paralelos, registros, 675  
 Paridad, bit de, 107  
 Particionado, 240-242  
 Particiones de tamaño fijo, 240  
 Particiones de tamaño variable, 241-242  
 Patillas (terminales) de dirección, 77, 78  
 Patillas (terminales) del sistema, 76, 78  
 PCL, 76-84  
 arbitraje, 83-84  
 arbitrio del bus, 83  
 ciclo de direccionamiento dual, 80, 81  
 definición de, 76  
 estructura del bus, 76-80  
 líneas de señal obligatorias, 78  
 orden de escritura de E/S, 30, 80-81  
 orden de lectura de E/S, 30, 80-81  
 orden escritura de memoria, 30, 81  
 orden lectura de memoria, 30, 81  
 órdenes, 80-81  
 órdenes de ciclo especial, 80  
 órdenes de configuración, 80, 81  
 patillas (terminales) de dirección y de datos, 77, 78  
 terminales de ampliación a bus de 64 bits, 79, 80  
 terminales de arbitraje, 77, 78  
 terminales de control de interfaz, 77, 78  
 terminales de interrupción, 79, 80  
 terminales de soporte de cache, 79  
 terminales de test bus PCI (JTAG/boundary scan), 79, 80  
 terminales del sistema, 76, 78  
 terminales para señales de error, 77, 78  
 transferencias de datos, 81-83  
 PDP-10, 380-381  
 PDP-11, 382, 578-579  
 VAX, 382-385  
 PDP-8, 3891, 439  
 Pentium, 43  
 Pentium II, 43, 494-500  
 buffer de reordenación (ROB), 497  
 consistencia de la cache de datos, 127  
 control de cache, 123  
 formatos de instrucción, 385-387  
 bytes prefijados, 386-387  
 campos de instrucciones, 387  
 gestión de memoria, 250-254  
 espacios de direcciones, 250  
 formatos, 252  
 paginación, 251-254  
 parámetros, 253  
 segmentación, 250-251  
 modos de direccionamiento, 371-374  
 direccionamiento relativo, 374  
 modo base con desplazamiento, 374  
 modo base con índice escalado y desplazamiento, 374  
 modo base, 374  
 modo de desplazamiento con índice escalado, 374  
 modo de operando en registro, 373  
 modo desplazamiento, 373  
 modo desplazamiento con índice escalado, 374  
 modo inmediato, 373  
 organización de cache, 125-128  
 depósito de instrucciones, 126  
 unidad de captación/decodificación, 126  
 unidad de envío/ejecución, 126  
 unidad de retirada, 126  
 organización de registros, 419-424  
 procesamiento de interrupciones, 424-427  
 puntero de instrucción, 419  
 registro de control, 420  
 registro de estado, 420  
 registro EFLAGS, 420-421  
 registros de control, 421-423  
 registros de segmento, 419  
 registros de uso general, 419  
 registros indicadores, 419  
 registros MMX, 423-424  
 registros numéricos, 420  
 predicción de saltos, 499-500  
 procesador, 419-427  
 tipos de datos, 321-322  
 tipos de operaciones, 336-343  
 códigos de condición, 339-341  
 gestión de memoria, 339  
 instrucciones de llamada/retorno, 336-339  
 instrucciones MMX, 341-343  
 unidad de captación y de codificación de instrucciones, 493-497

## Índice alfabético

- id de envío/ejecución, 497-499  
id de retiro (RU), 499  
n.º III, 43, 688-700  
en Pro, 43  
a escala, integración a (SSI), 31  
en, componente de interconexión  
área principal semiconductora, 99-  
tas de memoria de computadores, 99  
características clave, 93-95  
memoria, 95-99  
núcleo PCI  
os, 8  
os, dispositivos, ver también Dis-  
tos externos  
l, 317, 353-357  
le, 355  
ción de, 355  
ición de expresiones con, 355-357  
entación de, 353-355  
de la, 355  
o de, 355  
acionamiento de, 371  
rco de, 335  
tura de, 397  
47  
ción, 232-238  
ación a corto plazo, 233-238  
ación a largo plazo, 233  
ación a medio plazo, 233  
númer Corto plazo, planificación a  
desbordamiento a cero, 289  
desbordamiento, 289  
nación, 371  
1, 42-43, 44-45, 500-507  
le instrucciones, 502-504  
o de instrucciones, 388-389  
de memoria, 255-258  
cionamiento bloqueado, 255  
atos, 255  
mismo de traducción de direc-  
ones, 258  
neiros, 257  
de páginas, 254  
de direccionamiento, 374-376  
monitamiento cargar/memorización,  
5-376  
ionamiento de bifurcaciones, 376  
iciones aritméticas, 376  
ación de la cache, 128-129  
ación de registros, 427-430  
ro condición, 428  
ro de cuenta, 429  
ro de enlace, 429  
ro de estudio y control de coma  
ante (FPSCR), 428  
ro de excepción (XER), 428  
os de uso general, 428  
Web, 46  
lor, 427-433  
nimiento de interrupciones, 430-  
1 de interrupciones, 432-433  
o de estado de la máquina, 432
- tipos de interrupciones, 430  
procesamiento de ramificaciones (sal-  
tos), 504-505  
tipos de datos, 322-324  
tipos de operaciones, 343-346  
instrucciones carga/almacenamiento  
(load/store), 344-346  
instrucciones dedicadas a bifurca-  
ciones, 343  
unidad de coma flotante, 502  
unidad de enteros, 502  
unidad de envío, 502  
unidad de procesamiento de saltos, 502  
Precapacitación del destino del salto, 412  
Predescodificación, 507  
Pre-indexación, 371  
Preparado, estado de proceso, 234  
Prestaciones, 95  
balance de prestaciones, 40-42  
diseño buscando buenas prestaciones,  
38-42  
velocidad del microprocesador, 39-40  
Prestaciones, habilitación del contador de  
(PCE), 423  
Principal, memoria, 10, 53, 98, 243  
operando fuente y resultado, 313, 314  
Prioridad en el bloque de control del pro-  
ceso, 234  
Privilegio requerido, nivel de (RPL), 251  
Procedimiento, 333  
Procedimiento, instrucciones de llamada  
a, 333-336  
Procesador, 9, 66  
Procesador de E/S, 54  
Procesador G4, 45  
Procesador-memoria, 54  
Procesador para transferencias a memo-
- Programable, array lógico, 664-667  
Programable, ROM (PROM), 101  
Programadas, E/S, 182, 183-186  
inconvenientes de las, 195  
instrucciones de E/S, 185-186  
órdenes de E/S, 183-185  
visión general de las, 183  
Protección de escritura (WP), 422  
Protección, indicador de habilitación de  
(PE), 421  
Protegido, interrupciones virtuales en mo-  
do (PV), 423  
Protocolos de directorio, 614-615  
Protocolos de sondeo, 615  
Pseudoinstrucción, 346  
Puertas, 648-650  
conjunto funcionalmente completo de,  
649
- Q
- Quine-McCluskey, método de, 657-660
- R
- RAID, 155-163  
definición de, 155  
nivel 0, 156-160  
para alta capacidad de transferencia  
de datos, 158-159  
para altas frecuencias de petición de  
E/S, 159-160  
nivel 1, 160  
nivel 2, 160-161  
nivel 3, 161-162  
prestaciones, 162  
redundancia, 161-162  
nivel 4, 162  
nivel 5, 163  
nivel 6, 163  
RAM dinámica, 99-100  
Rambus DRAM (RDRAM), 131  
Ramificación incondicional, 24, 25  
RDRAM, 131  
Ren, memoria, 345  
Reanudación, indicador de (RF, registro  
EFLAGS), 421  
Recursos, codificación por, 376  
Red, conexiones, 70  
Redondeo, 298-300  
a más y menos infinito, 298, 300  
al más próximo, 298, 299  
hacia cero, 298, 300  
Reentrantes, procedimientos, 335  
Referencia a operandos fuente, 313  
Registro de dirección de control (CAR),  
567  
Registros, 11, 21, 396, 674-676  
Registros de control y de estado, 398-400  
Registros de datos, 397  
Registros de dirección, 397  
Registros de uso general, 397  
Registros visibles para el usuario, 397-398  
Registros, organización de, 396-401  
códigos (indicadores) de condición, 398  
ejemplo de organizaciones de registros  
en microprocesadores, 400-401  
registros de control y estado, 398-400

- registros de datos, 397  
 registros de dirección, 397  
 registros de uso general, 397  
 registros visibles para el usuario, 397-398  
**Registros, renombramiento de**, 490-491  
**Relativo, direccionamiento**, 369  
**Reloj**, 69  
**Repetitorio de instrucciones**  
 3D NOW!, 690  
 MAX2, 690  
 MDMX, 690  
 MVI, 690  
 SEE, 688, 690-700  
 SIMD, 688-690  
 VIS, 690  
**Representación en coma fija**, 272  
**Representación en coma flotante**, 287-293  
 normalización IEEE 754, 291-293  
 principios de la, 287-291  
**Representación en complemento a dos**, 269-271  
**Residente, monitor**, 225  
**Resta**, 276  
 en aritmética de coma flotante, 294-297  
**Resultado, referencia al operando de**, 313  
**Retardo de puerta**, 648  
**RISC (computador de repertorio reducido de instrucciones)**, 319  
**RISC, sistemas**, 437-478  
 Arquitectura, 439  
 características de la ejecución de instrucciones, 439-444  
 consecuencias, 444  
 llamadas a procedimientos, 443  
 operaciones, 441-442  
 operandos, 442-443  
 características de las arquitecturas, 452-455  
 formatos de instrucción sencillos, 453-454  
 modos de direccionamiento sencillos, 453  
 operaciones registro a registro, 452, 453  
 una instrucción por ciclo, 452, 453  
 características/motivación de los, 450-457  
**CISC (computador de repertorio complejo de instrucciones)**, 442  
 características de los CISC frente a los RISC, 455-457  
 objetivos de los, 451-452  
 optimización de registros por el compilador, 448-450  
**ROM (memoria de solo lectura)**, 95, 100-101, 667-668  
**Rotación, operaciones de**, 329, 330  
**Rotacional, detección de posición RPS**, 152  
**Rotacional, retardo**, 152, 153
- S**
- S/390, instrucción de desplazamiento de caracteres (MVC)**, 454  
**Salto condicional**, 24, 25  
**Salto hacia adelante**, 332  
**Salto hacia atrás**, 322  
**Salto implícito, instrucciones de**, 333  
 Instrucciones de bifurcación, 332  
**Salto retardado**, 417  
**Salto (bifurcaciones), tratamiento de**, 411-417  
**Salto, predicción de**, 413-417, 492-493  
**SCSI (interfaz para pequeños computadores)**, 70, 174, 201-209  
 bloque descriptor de orden (CDB), 207-208  
 código de operación, 207  
 control, 208  
 dirección de bloque lógico, 207  
 longitud de la asignación, 208  
 longitud de la lista de parámetros, 208  
 longitud de la transferencia, 208  
 número de la unidad lógica, 207  
 fase de arbitraje, 202  
 fase de bus libre, 202  
 fase de datos, 202  
 fase de orden, 202  
 fase mensaje, 202  
 mensaje de abortar, 206  
 mensaje de desconexión, 206  
 mensaje de error detectado en el iniciador, 206  
 mensaje de orden completa, 206  
 mensajes, 206  
 orden pregunta, 208  
 órdenes, 206-209  
 señales del bus, 204  
**SCSI Trade Association**, 214  
**SDRAM**, 130-131  
**Sectores**, 147  
**Secuencia de instrucciones única y múltiples secuencias de datos (SIMD)**, 601-602  
**Secuencia de instrucciones única y secuencias de datos únicas (SISD)**, 601  
**Secuencial, acceso**, 94  
**Secuenciales, circuitos**, 671-680  
 Biestables, 671-674  
 Contadores, 676-680  
 Registros, 674-676  
**Secundario activo**, 621  
**Segmentación**, 247-249, 456-457, 476-480  
 estrategia de, 405-410  
**Segmentación no paginada, memoria con**, 250  
**Segmentación paginada, memoria con**, 250  
**Segmentación, prestaciones**, 410-411  
**Segmento, número de**, 251  
**Segmento, punteros de**, 397  
**Selección de columna de dirección (CAS)**  
 señales de, 103  
**Selector, canal**, 199  
**Semántico, salto**, 440  
**Semiconductor, memoria principal**, 99-114  
 corrección de errores, 106-111  
 encapsulado del chip, 104-105  
 lógica del chip, 102-104  
 memoria de acceso aleatorio  
 organización en módulos, 105-106  
 organización, 101-102  
 tipos de, 99-101  
**Semiconductor, memoria**, 36, 95  
**Señales de control**, 68, 177, 540-544  
**Señales de flancos de bajada**, 88  
**Señales de temporización**, 68  
**Signo y magnitud, representación en**, 268  
**Siguiente instrucción, referencia a**, 312, 313  
**Simbólica de instrucciones máquina, representación**, 315  
**Simbólico, programa**, 346  
**SimpleScalar, simulador**, 684-685  
**Simplificación algebraica**, 653  
**Simulación de campos continuos**, 627  
**Simulación, proyectos de**, 684-685  
**Síndrome, palabra de**, 108  
**Sistema jerárquico, definición**, 6  
**Sistema operativo**, 219-261  
 acceso a dispositivos de E/S, 222  
 acceso controlado a ficheros, 222  
 como gestor de recursos, 223-224  
 como interfaz usuario computador, 221-223  
 contabilidad, 223  
 conveniencia del, 221  
 creación de programas, 222  
 de tiempo compuesto, 222  
 definición del, 220  
 detección/respuesta a errores, 222  
 eficiencia del, 221  
 ejecución de programas, 222  
 gestión de la memoria, 231, 238-249  
 gestión de memoria en el Pentium II/  
 PowerPC, 250-258  
 monoprogramación, 224, 231  
 multiprogramación, 224, 231  
 núcleo, 223  
 objetivosfunciones de, 221-224  
 planificación en el, 233-238  
 primeros, 224-225  
 sistema de acceso al, 222  
 sistemas por lotes (cola), 225-232  
 cola única, 225-227  
 instrucciones privilegiadas, 227  
 interrupciones, 227  
 lenguaje de control de trabajos (JCL), 226-227  
 monitor residente, 225  
 multiprogramación, 227-232  
 protección de memoria, 227  
 reloj, 227  
 tipos de, 224-232  
 ver también Gestión de memoria; Planificación.  
 visión general de un, 221-232  
**SPM, ver también Multiprocesamiento simétrico**  
**Software**, 52  
**Software del sistema**, 27  
**Software, conjunto**, 190  
**SPARC**, 468-473  
 conjunto de registros, 468  
 formato de instrucción, 472-473  
 repertorio de instrucciones, 468-472  
**SSYN (sincronización del maestro), señal**, 73  
**Suma**, 275-276  
 en aritmética de coma flotante, 294-297  
**Sumadores**, 668-670  
**Supercomputadores**, 601, 627  
**Superscalar, organización**, 493  
**Superscalares, procesadores**, 479-537  
 carga especulativa, 519-522

- con finalización desordenada, 489  
 con finalización en orden, 487-488  
 conceptos básicos subyacentes, 511-512  
 conflictos en los recursos, 486  
 cuestiones sobre diseño de, 486-494  
 definición de, 480, 481  
 dependencia de datos verdadera, 484  
 dependencias de procedimientos, 485-486  
 ejecución con predicciones, 515-519  
 emisión desordenada con finalización desordenada, 489-490  
 emisión en orden  
 formato de instrucción, 514-515  
 IA-64 MERCEDE, 511-522  
 limitaciones, 483-486  
 MIPS R10000, 507-509  
 motivación, 512-513  
 organización, 513-514  
 paralelismo a nivel de instrucción, 483, 486-487  
 paralelismo de la máquina, 456-457, 491-492  
 políticas de emisión de instrucciones, 487-490  
 Predicción, 507  
 predicción de saltos, 492-493
- T**
- TIO (página web), 214  
 Tabla verdad, 630  
 Tabla, indicador de (TI), 251  
 Tarea comunitada (TS), 421  
 Tarjeta de consulta RAID, 169  
 Tasa de transferencia, 95  
 Teclado, 177-178  
 Teclado-monitor, 177-178  
 regla de DeMorgan, 649  
 terminales de arbitraje, 77-78  
 terminales de datos, 77, 78  
 Terminales de soporte de cache, 83  
 Terminales de test bus PCI/JTAG/boundary scan, 79, 80  
 Terminales para señales de error, 77, 78
- TF 8800, 583-593  
 ALU con registros, 589-593  
 formato de microinstrucción, 584  
 microsecuenciador, 584-589  
 control del microsecuenciador, 588  
 pila, 586-587  
 registros/contadores, 586
- Tiempo de acceso 95, 152  
 Tiempo de transferencia, 153  
 Tipo de coprocesador (ETL), 421  
 del Pentium II y PowerPC.  
 formatos de instrucciones, 385-389  
 modos de direccionamiento, 371-376  
 tipos de datos, 321-324  
 tipos de operaciones, 336-346  
 pila, 312, 317  
 referencia al operando fuente, 313  
 referencia al operando resultado, 313  
 registros, 319
- Trabajo, 225  
 Trabajos, lenguaje de control de (JCL), 226-227
- Transductor, 177
- Transferencia ACK, 69  
 Transferencia de datos, 24, 25, 324, 326-327
- Transferencias de datos, 8, 30  
 instrucciones para, 316
- Transistores, 26-27
- Translación doble de dirección, mecanismo de, 254
- Transparencia del hardware, 123
- U**
- Ubicación de bits, 373-379  
 PDP-8, 379-380  
 PDP-10, 380-381
- Último en entrar-primer en salir, cola, 371
- UltraSPARC-II, 509-511  
 cauce segmentado, 510-511  
 organización interna, 509
- Un bit por chip, organización, 102
- Unidad de control (CU), 10, 12, 21, 396  
 Unidad de control cableada, 565  
 Unidad de procesamiento central (CPU), 4, 9, 10-11, 27  
 bifurcación de la instrucción, 418  
 ciclo de instrucción de la, 401-405  
 del Pentium II, 419-427  
 del PowerPC, 427-433  
 estructura interna de la, 396  
 estructura y función de la, 393-436  
 organización de la, 393-396  
 organización de los registros de la, 396-401  
 ver también Ciclo instrucción; Computación vectorial (encadenamiento), 631-632
- Unidad de transferencia, 94
- Unidades de dirección, 94
- UNIVAC I y III, 24-26
- USENET, grupo de noticias, 16
- V**
- Velocidad angular constante (CAV), 164  
 Velocidad lineal constante (CLV), 164  
 Videodiscos digitales o Disco video digital (DVD), 164, 167
- Vigilancia del bus con escritura inmediata, 123
- Virtual 3086 (VME), ampliación del modo, 423
- voltajes decimal/ASCII, 359  
 ver también Endian
- Von Neumann, arquitectura, 51  
 Von Neumann, John, 30, 51  
 Von Neumann, máquina, 20-24  
 ver también Computador IAS
- W**
- Web, sitios, 15-16  
 Wilkes, control de, 562-565  
 WORM, 164, 166
- WWW, página web principal de arquitectura de computadores, 16

# ACRÓNIMOS

| Acrónimo | Término en inglés                                   | Término en castellano                                     |
|----------|-----------------------------------------------------|-----------------------------------------------------------|
| ALU      | Arithmetic and Logic Unit                           | Unidad aritmético-lógica                                  |
| ANSII    | American National Standards Institute               | Instituto Nacional Americano para Normalizaciones         |
| ASCII    | American Standards Code for Information Interchange | Código Estándar Americano para Intercambio de Información |
| BCD      | Binary Coded Decimal                                | Cifras decimales codificadas en binario                   |
| CD       | Compact Disk                                        | Disco compacto                                            |
| CD-ROM   | Compact Disk-Read Only Memory                       | Disco compacto de sólo lectura                            |
| CISC     | Complex Instruction Set Computer                    | Computador con repertorio complejo de instrucciones       |
| CPU      | Central Processing Unit                             | Unidad central de procesamiento                           |
| DMA      | Direct Memory Access                                | Acceso directo a memoria                                  |
| DRAM     | Dynamic Random-Access Memory                        | Memoria dinámica de acceso aleatorio                      |
| EPIC     | Explicitly Parallel Instruction Computing           | Computación con paralelismo de instrucciones explícito    |
| EPROM    | Erasable Programmable Read-Only Memory              | Memoria de sólo lectura borrable eléctricamente           |
| I/O      | Input/Output                                        | Entrada/salida                                            |
| IAR      | Instruction Address Register                        | Registro de dirección de instrucción                      |
| IC       | Integrated Circuit                                  | Circuito integrado, chip                                  |
| IEEE     | Institute of Electrical and Electronics Engineers   |                                                           |
| ILP      | Instruction-Level Parallelism                       | Paralelismo a nivel de instrucción                        |
| IR       | Instruction Register                                | Registro instrucción                                      |
| LAN      | High-Level Language                                 | Lenguaje de alto nivel                                    |
| LRU      | Least Recently Used                                 | Menos usada recientemente                                 |
| LSI      | Large-Scale Integration                             | Integración en gran escala                                |
| MAR      | Memory Address Register                             | Registro de dirección de memoria                          |
| MBR      | Memory Buffer Register                              | Registro intermedio de memoria                            |
| MESI     | Modify-Exclusive-Shared-Invalid                     | Modificado-excluido-compartido-no válido                  |
| MMU      | Memory Management Unit                              | Unidad de gestión de memoria                              |
| MSI      | Medium-Scale Integration                            | Integración de media escala                               |
| NUMA     | Nonuniform Memory Access                            | Acceso a memoria no uniforme                              |
| PC       | Program Counter                                     | Contador de programa                                      |
| PCB      | Process Control Block                               | Bloque de control del proceso                             |
| PCI      | Peripheral Component Interconnect                   | Interconexión para componentes periféricos                |
| PROM     | Programmable Read-Only Memory                       | Memoria de sólo lectura programable                       |
| PSW      | Processor Status Word                               | Palabra de estado del procesador                          |
| RAID     | Redundant Array of Independent Disks                | Arraíz redundante de discos independientes                |
| RALU     | Register/Arithmetic Logic Unit                      | Unidad aritmético-lógica con registros                    |
| RAM      | Random-Access Memory                                | Memoria de acceso aleatorio                               |
| RISC     | Reduced Instruction Set Computer                    | Computador con repertorio reducido de instrucciones       |
| ROM      | Read-Only Memory                                    | Memoria de sólo lectura                                   |
| SCSI     | Small Computer System Interface                     | Interfaz para computadores pequeños                       |
| SMP      | Symmetric Multiprocessors                           | Multiprocesadores simétricos                              |
| SRAM     | Static Random-Access Memory                         | Memoria de acceso aleatorio estática                      |
| SSI      | Small-Scale Integration                             | Integración en pequeña escala                             |
| VLSI     | Very Large-Scale Integration                        | Integración en muy gran escala                            |
| VLWI     | Very Long Instruction Word                          | Palabra de instrucción muy larga                          |
| WORM     | Write-Only Read-Many                                | Memoria de una sola escritura y múltiples lecturas        |