

ARQUITECTURA DE COMPUTADORAS

Patricia Quiroga



Alfaomega

ARQUITECTURA DE COMPUTADORAS

ARQUITECTURA de COMPUTADORAS

Patricia Quiroga



Buenos Aires • Bogotá • México DF • Santiago de Chile

Quiroga, Irma Patricia.
Arquitectura de computadoras. - 1a ed. - Buenos Aires : Alfaomega
Grupo Editor Argentino, 2010.
372 pp. ; 24 x 21 cm.

ISBN 978-987-1609-06-2

1. Informática. 2. Arquitectura de Computadoras. I. Título
CDD 621.39

Queda prohibida la reproducción total o parcial de esta obra, su tratamiento informático y/o la transmisión por cualquier otra forma o medio sin autorización escrita de Alfaomega Grupo Editor Argentino S.A.

Edición: Damián Fernández

Corrección: Paula Smulevich y Silvia Mellino

Diseño de interiores: Juan Sosa

Diagramación de interiores: Diego Linares

Corrección de armado: Silvia Mellino

Revisión técnica: José Luis Hamkalo

Diseño de tapa: Diego Linares

Dibujos: Tomas L'Estrange

Internet: <http://www.alfaomega.com.mx>

Todos los derechos reservados © 2010, por Alfaomega Grupo Editor Argentino S.A.
Paraguay 1307, PB, oficina 11

ISBN 978-987-1609-06-2

Queda hecho el depósito que prevé la ley 11.723

NOTA IMPORTANTE: La información contenida en esta obra tiene un fin exclusivamente didáctico y, por lo tanto, no está previsto su aprovechamiento a nivel profesional o industrial. Las indicaciones técnicas y programas incluidos han sido elaborados con gran cuidado por el autor y reproducidos bajo estrictas normas de control. Alfaomega Grupo Editor Argentino S.A. no será jurídicamente responsable por errores u omisiones, daños y perjuicios que se pudieran atribuir al uso de la información comprendida en este libro, ni por la utilización indebida que pudiera dársele.

Los nombres comerciales que aparecen en este libro son marcas registradas de sus propietarios y se mencionan únicamente con fines didácticos, por lo que Alfaomega Grupo Editor Argentino S.A. no asume ninguna responsabilidad por el uso que se dé a esta información, ya que no infringe ningún derecho de registro de marca. Los datos de los ejemplos y pantallas son ficticios, a no ser que se especifique lo contrario.

Los hipervínculos a los que se hace referencia no necesariamente son administrados por la editorial, por lo que no somos responsables de sus contenidos o de su disponibilidad en línea.

Empresas del grupo:

Argentina: Alfaomega Grupo Editor Argentino, S.A.
Paraguay 1307 P.B. "11", Buenos Aires, Argentina, C.P. 1057
Tel.: (54-11) 4811-7183 / 8352
E-mail: ventas@alfaomegagroupeditor.com.ar

México: Alfaomega Grupo Editor, S.A. de C.V.
Pitágoras 1139, Col. Del Valle, México, D.F., México, C.P. 03100
Tel.: (52-55) 5089-7740 – Fax: (52-55) 5575-2420 / 2490. Sin costo: 01-800-020-4396
E-mail: atencionalcliente@alfaomega.com.mx

Colombia: Alfaomega Colombiana S.A.
Carrera 15 No. 64 A 29, Bogotá, Colombia
PBX (57-1) 2100122 - Fax: (57-1) 6068648
E-mail: scliente@alfaomega.com.co

Chile: Alfaomega Grupo Editor, S.A.
General del Canto 370-Providencia, Santiago, Chile
Tel.: (56-2) 235-4248 – Fax: (56-2) 235-5786
E-mail: agechile@alfaomega.cl

A mi hijo, Pablo J. Sabbatiello y a mi hija, Bárbara A. Sabbatiello, que me acompañan en todo momento.

A mis padres, María y Argentino Quiroga, a los que les debo el apoyo incondicional
a lo largo de mi vida personal y académica.

Patricia Quiroga

Agradecimientos

Al grupo de docentes que constituyeron la Cátedra: Elvira Quiroga, Roberto Tenuta, Edgar Leal, José Bellesi, Miguel Ángel Di Paolo, Mario Albarracín, Antonio Shütz, Rubén López, Eva Bernardez García, Silvana Panizzo, Hugo Borchert y muy especialmente a María Paz Colla, que colaboró en la creación de los PowerPoint y autoevaluaciones.

A la Dra. Alicia Ortiz, a la Lic. Patricia Machado, al Lic. Daniel Jaloff y al Lic. Julio César Liporace, que me han apoyado en la concreción de esta obra desde sus respectivas áreas de conocimiento.

Un profundo reconocimiento a Tanya Itzel Arteaga Ricci y Eduardo Espinosa Avila, profesores de la Universidad Nacional Autónoma de México (Facultad de Ingeniería), quienes realizaron varias observaciones sobre el manuscrito de esta obra.

Mensaje del Editor

Los conocimientos son esenciales en el desempeño profesional. Sin ellos es imposible lograr las habilidades para competir laboralmente. La Universidad o las instituciones de formación para el trabajo ofrecen la oportunidad de adquirir conocimientos que serán aprovechados más adelante en beneficio propio y de la sociedad. El avance de la ciencia y de la técnica hace necesario actualizar continuamente esos conocimientos. Cuando se toma la decisión de embarcarse en una vida profesional, se adquiere un compromiso de por vida: mantenerse al día en los conocimientos del área u oficio que se ha decidido desempeñar.

Alfaomega tiene por misión ofrecer conocimientos actualizados a estudiantes y profesionales, dentro de lineamientos pedagógicos que faciliten su utilización y permitan desarrollar las competencias requeridas por una profesión determinada. Alfaomega espera ser su compañera profesional en este viaje de por vida por el mundo del conocimiento.

Alfaomega hace uso de los medios impresos tradicionales en combinación con las tecnologías de la información y las comunicaciones (IT) para facilitar el aprendizaje. Libros como éste tienen su complemento en una página Web, en donde el alumno y su profesor encontrarán materiales adicionales, información actualizada, pruebas (test) de autoevaluación, diapositivas y vínculos con otros sitios Web relacionados.

Esta obra contiene numerosos gráficos, cuadros y otros recursos para despertar el interés del estudiante y facilitarle la comprensión y apropiación del conocimiento.

Cada capítulo se desarrolla con argumentos presentados en forma sencilla y estructurada claramente hacia los objetivos y metas propuestas. Cada uno de ellos concluye con diversas actividades pedagógicas para asegurar la asimilación del conocimiento y su extensión y actualización futuras.

Los libros de Alfaomega están diseñados para ser utilizados dentro de los procesos de enseñanza-aprendizaje y pueden ser usados como textos guía en diversos cursos o como apoyo para reforzar el desarrollo profesional.

Alfaomega espera contribuir así a la formación y el desarrollo de profesionales exitosos para beneficio de la sociedad.

Sobre la autora**Ing. Patricia Quiroga**

Analista Universitario de Sistemas, egresada de la Universidad Tecnológica Nacional (Facultad Regional Buenos Aires).

Ingeniera en Informática, egresada de la Universidad Católica de Salta.

Especialista en Criptografía y Seguridad Teleinformática, egresada del Instituto de Enseñanza Superior del Ejército.

Secretaria Académica de las Carreras Ingeniería en Informática, Ingeniería en Telecomunicaciones y Licenciatura en Informática en la Universidad Católica de Salta (Subsede Buenos Aires)

Titular de la Cátedra de Arquitectura de los Computadores de la Universidad Tecnológica Nacional (Facultad Regional Buenos Aires).

En el ámbito privado, se desempeña como Perito Informático.

Revisor técnico: Dr. José Luis Hamkalo

Profesor de la Facultad de Ingeniería de la Universidad de Buenos Aires, Argentina. Sus áreas de interés son Arquitecturas de Computadoras y Jerarquías de Memoria. Es miembro del IEEE y de la ACM. Recibió los títulos de Ingeniero Electrónico y Doctor de la Universidad de Buenos Aires. Área Ingeniería.

Contenido		
Capítulo 1		
Evolución del procesamiento de datos	1	28
1.1 Organización y arquitectura de una computadora	2	29
1.2 Estratificación del software	3	30
1.3 Evolución del procesamiento de datos.....	4	30
1.3.1 Los comienzos de la computación	4	30
1.3.2 La primera máquina y su evolución.....	5	31
1.3.3 La máquina de tarjetas perforadas.....	5	31
1.3.4 La calculadora secuencial automática (IBM).....	5	32
1.3.5 El programa almacenado.....	6	35
1.4 Clasificación de las computadoras.....	7	37
1.4.1 Analógicas	7	37
1.4.2 Digitales.....	8	37
1.4.3 Híbridas.....	8	39
1.5 Generaciones de computadoras digitales	9	40
1.5.1 Computadoras de 1 ^a generación	9	40
1.5.2 Computadoras de 2 ^a generación	9	40
1.5.3 Computadoras de 3 ^a generación	10	40
1.5.4 Computadoras de 4 ^a generación	11	40
1.5.5 Computadoras de 5 ^a generación	12	40
1.6 Procesamiento de datos y sistemas de información.....	13	42
1.7 Sistemas sincrónicos de propósito general.....	15	43
1.8 Arquitectura de computadoras: Los primeros conceptos.....	15	46
1.9 Arquitectura de una unidad central de proceso (CPU).....	17	47
1.10 Lógica digital y componentes electrónicos.....	18	47
1.11 El Sistema Operativo. La Dinámica del Sistema	23	49
1.12 Resumen	23	50
1.13 Contenido de la página Web de apoyo	24	50
Capítulo 2		
Sistemas numéricos.....	25	53
2.1 Introducción	26	53
2.2 Sistemas de notación posicional.....	26	54
2.2.1 Expresión generalizada de un número en potencias de su base	26	54
2.2.2 Sistema decimal.....	26	55
2.2.3 Sistema binario	27	55
2.2.4 Sistema octal.....	27	56
2.3 Métodos de conversión de números enteros y fraccionarios	28	56
2.3.1 Método de conversión de números de otras bases a decimal	28	56
2.3.2 Método de divisiones sucesivas (para convertir un número entero decimal a otras bases).....	29	57
2.3.3 Método de multiplicaciones (para convertir un número fraccionario decimal a otras bases).....	29	58
2.3.4 Pasaje directo entre las bases 2 a 8 y 2 a 16	30	58
2.4 Operaciones fundamentales en binario.....	31	59
2.4.1 Suma.....	31	59
2.4.2 Resta o sustracción.....	31	59
2.5 Operaciones fundamentales en octal y hexadecimal.....	32	60
2.5.1 Suma octal	32	60
2.5.2 Técnica para sumar números grandes en cualquier base	32	60
2.5.3 Suma hexadecimal	33	61
2.6 Complemento de un número.....	33	62
2.6.1 Complemento a la base, a la raíz o auténtico	33	62
2.6.2 Su utilización para la representación de binarios negativos complementados a "2"	34	63
2.6.3 Complemento a la base -1 o restringido	34	63
2.6.4. Su utilización para la representación de binarios negativos complementados a "1"	35	64
2.7 Resumen	35	64
2.8 Ejercicios propuestos	36	66
2.9 Contenido de la página Web de apoyo.....	37	67
Capítulo 3		
Representación de datos en la computadora.....	39	69
3.1 Introducción	40	69
3.2 Flujo de datos dentro de una computadora.....	40	70
3.3 Códigos de representación de caracteres alfanuméricos..	41	73
3.3.1 Código ASCII	41	73
3.3.2 Código ASCII ampliado	42	74
3.3.3 Delimitación de strings	42	75
3.4 Códigos de representación decimal (BCD).....	42	76
3.4.1 BCD puro o natural	42	76
3.4.2 BCD exceso tres	43	78
3.4.3 BCD AIKEN o 2421	43	78
3.5 Códigos de representación numérica no decimal	43	79
3.5.1 Coma o punto fijo sin signo (enteros positivos)	43	79
3.5.2 Coma o punto fijo con signo (enteros)	44	80

3.5.3 Coma o punto fijo con signo con negativos complementados a "2" (enteros)	60	5.5.3 "Compuerta OR EXCLUSIVE" o "compuerta exclusiva"	96		
3.5.4 Coma o punto fijo con signo con negativos complementados a "1" (enteros)	61	5.5.4 "Compuerta NOT" o "inversión"	96		
3.5.5 Reales en coma o punto flotante (números muy grandes y números reales)	62	5.5.5 "Compuertas con funciones negadas"	97		
3.6 Representaciones redundantes.....	68	5.6 Circuito lógico	99		
3.6.1 Códigos de detección y/o corrección de errores.		5.6.1 Transistor	100		
Introducción	68	5.6.2 Compuerta triestado	101		
3.6.2 Paridad vertical simple o a nivel carácter	68	5.7 Circuito sumador-binario en paralelo	102		
3.6.3 Paridad horizontal a nivel de bloque	68	5.7.1 Circuito semisumador (SS) o <i>Half Adder</i> (HA)	103		
3.6.4 Paridad entrelazada	68	5.7.2 Circuito sumador completo (SC) o <i>Full Adder</i> (FA) ..	103		
3.6.5 Código de Hamming	69	5.8 Formas normales o canónicas de una función.....	104		
3.7 Resumen	71	5.8.1 Forma normal disyuntiva	105		
3.8 Ejercicios propuestos	71	5.8.2 Forma normal conjuntiva	105		
3.9 Contenido de la página Web de apoyo.....	72	5.9 Circuitos equivalentes.....	107		
Capítulo 4					
Aritmética de la computadora	73	5.10 Minimización de circuitos.....	108		
4.1 Introducción	74	5.10.1 Ejemplos de minimización a partir de distintos mapas de Karnaugh de 2, 3 y 4 variables	109		
4.2 Aritmética binaria.....	74	5.11 Resumen.....	110		
4.2.1 Representación de datos en punto fijo (binarios enteros)	74	5.12 Ejercicios propuestos	111		
4.2.2 Operaciones aritméticas con enteros signados	75	5.13 Contenido de la página Web de apoyo.....	112		
4.2.3 Operaciones aritméticas en punto flotante	78	Capítulo 6			
4.3 Aritmética decimal	79	Lógica digital	113		
4.3.1 Operaciones con operandos BCD.....	80	6.1 Introducción	114		
4.4 Resumen.....	84	6.2 Circuitos lógicos de sistemas digitales	114		
4.5 Ejercicios propuestos	85	6.3 Circuitos combinacionales	114		
4.6 Contenido de la página Web de apoyo.....	86	6.3.1 Circuito generador de paridad	115		
Capítulo 5					
Álgebra de Boole.....	87	6.3.2 Circuito comparador de magnitud	116		
5.1 Introducción	88	6.3.3 Circuitos convertidores de código	117		
5.2 Álgebra de los circuitos digitales	88	6.3.4 Circuitos codificadores	118		
5.2.1 Elementos de Álgebra proposicional.....	88	6.3.5 Circuito decodificador de código	120		
5.3 Álgebra de Boole	89	6.3.6 Circuito decodificador $n \cdot 2n$	121		
5.3.1 Operadores	89	6.3.7 Circuitos multiplexores y demultiplexores	124		
5.3.2 Tablas de verdad	90	6.3.8 Circuitos "programables" para múltiples funciones	127		
5.3.3 Propiedades del Álgebra de Boole	90	6.4 Circuitos secuenciales	131		
5.3.4 Teoremas del Álgebra de Boole	91	6.4.1 Biestables o <i>flip-flops</i>	131		
5.4 Función booleana.....	92	6.4.2 Registros	136		
5.5 Compuertas lógicas o gates	93	6.5 Resumen.....	141		
5.5.1 "Compuerta AND", "compuerta y" o "compuerta producto lógico"	95	6.6 Ejercicios propuestos	142		
5.5.2 "Compuerta OR", "compuerta +" o "compuerta suma lógica"	95	6.7 Contenido de la página Web de apoyo.....	143		

Contenido	XIII
7.3.1 Instrucciones	148
7.4 Presentación del modelo de estudio	150
7.4.1 Fase fetch: búsqueda de una instrucción en memoria	152
7.4.2 Fase execute	157
7.4.3 Flujo de datos entre los registros de la computadora básica	161
7.4.4 Juego completo de instrucciones de "X"	163
7.4.5 Unidad de control y sincronización del tiempo	165
7.4.6 El módulo de cálculo: unidad aritmético-lógica	170
7.5 Resumen	176
7.6 Ejercicios propuestos	177
7.7 Contenido de la página Web de apoyo	177
Capítulo 8	
Microprocesadores	179
8.1 Introducción	180
8.2 Microprocesadores y microcontroladores	180
8.2.1 Chips y microprocesadores	181
8.3 Longitud de palabra	182
8.4 Capacidad de direccionamiento	182
8.5 Número de instrucciones	183
8.6 Número de registros internos	184
8.6.1 Registros de uso general IA-16 e IA-32	185
8.7 Velocidad del microprocesador	188
8.8 Ciclo de instrucciones	188
8.8.1 Secuencia de llenado de la cola	190
8.8.2 Etapas de ejecución de la rutina ejemplo	190
8.9 Capacidad de interrupción	192
8.9.1 Concepto de pila	195
8.10 Alimentación	199
8.11 Tecnología	200
8.11.1 CISC	200
8.11.2 RISC	200
8.11.3 EPIC	201
8.12 Resumen	202
8.13 Ejercicios propuestos	203
8.14 Contenido de la página Web de apoyo	205
Capítulo 9	
Memorias	207
9.1 Introducción	208
9.2 Clasificación de memorias	209
9.2.1 Clasificación según el modo de acceso a la unidad de información	209
9.2.2 Clasificación según las operaciones que aceptan por cada acceso	210
9.2.3 Clasificación según la duración de la información	210
9.3 Dimensión de la memoria	210
9.4 Memorias RAM estáticas y dinámicas	211
9.4.1 Memorias SRAM (<i>Static Random Access Memory</i>) ..	211
9.4.2 Memorias DRAM (<i>Dynamic Random Access Memory</i>) ..	211
9.4.3 RAM con acceso directo	211
9.4.4 RAM con acceso asociativo	215
9.5 Jerarquía de memorias	215
9.6 Memorias caché	217
9.6.1 Principios de funcionamiento	220
9.6.2 <i>Caching</i>	221
9.6.3 Actualización de caché	225
9.6.4 Actualización de la memoria principal	226
9.6.5 Niveles de caché	227
9.7 Memoria principal	227
9.7.1 Memoria a nivel lógica digital	228
9.7.2 Memorias RAM dinámicas	229
9.8 La memoria como en un espacio lógico	233
9.8.1 Almacenamiento de bytes en memoria. <i>Big-Endian</i> y <i>Little-Endian</i>	234
9.8.2 Gestión de memoria y modos de operación de los procesadores	234
9.8.3 Modelo de memoria segmentada pura	236
9.8.4 Modelo de memoria virtual	236
9.8.5 Modelo de memoria virtual paginada o paginación por demanda	238
9.8.6 Memoria virtual segmentada o segmentación por demanda	239
9.9 Administración de memorias externas	243
9.9.1 Archivos	244
9.9.2 Sistema de archivos en discos de tecnología magnética	246
9.9.3 Disco magnético desde el punto de vista lógico	246
9.9.4 <i>Buffers</i> y cachés de disco	253
9.9.5 Discos virtuales	254
9.9.6 Sistema de archivos en discos de tecnología óptica	254
9.10 Resumen	254
9.11 Ejercicios propuestos	255
9.12 Contenido de la página Web de apoyo	256
Capítulo 10	
Instrucciones	257
10.1 Introducción	258
10.2 Formato de instrucción	258
10.2.1 Instrucciones sin dirección	258
10.2.2 Instrucciones de una sola dirección	259

10.2.3 Instrucciones de dos direcciones	260	12.3.5 Tarjetas SRAM	297
10.2.4 Instrucciones de tres direcciones	261	12.3.6 Tarjetas Flash.....	297
10.2.5 Instrucciones de cuatro direcciones.....	261	12.4 Resumen	297
10.3 Modos de direccionamiento.....	262	12.5 Ejercicios propuestos	298
10.3.1 Direccionamiento directo de memoria.....	263	12.6 Contenido de la página Web de apoyo	298
10.3.2 Direccionamiento implícito.....	263		
10.3.3 Direccionamiento inmediato	264		
10.3.4 Direccionamiento indirecto	264		
10.3.5 Direccionamiento de la CPU asociado a registros	265		
10.3.6 Direccionamiento directo por registro.....	266		
10.3.7 Direccionamiento indexado.....	266		
10.3.8 Direccionamiento relativo a la base	267		
10.3.9 Direccionamiento a una pila (stack)	268		
10.4 Tipos válidos de instrucción	270		
10.5 Resumen	271		
10.6 Ejercicios propuestos	271		
10.7 Contenido de la página Web de apoyo	275		
Capítulo 11			
Software del sistema.....	277		
11.1 Introducción.....	278		
11.2 Clasificación del software de sistema	278		
11.3 Sistema operativo.....	278		
11.3.1 Niveles de administración del sistema operativo ...	279	13.4.1 Transferencia controlada por programa	312
11.3.2 Tipos de sistemas operativos	281	13.4.2 Transferencia iniciada por interrupción	313
11.4 Traductores de lenguaje	282	13.4.3 Transferencia con acceso directo a memoria	313
11.4.1 Ensambladores	282	13.5 Resumen	315
11.4.2 Intérpretes	284	13.6 Ejercicios propuestos	315
11.4.3 Compiladores	284	13.7 Contenido de la página Web de apoyo	316
11.5 Resumen	285		
11.6 Ejercicios propuestos	286		
11.7 Contenido de la página Web de apoyo	288		
Capítulo 12			
Dispositivos de entrada/salida.....	289		
12.1 Introducción.....	290		
12.2 Discos rígidos	290		
12.2.1 Controladora de disco	291	14.3 Paralelismo a nivel arquitectura	322
12.2.2 Especificaciones técnicas de un disco	291	14.3.1 Taxonomía de Flynn. Una clasificación de	
12.2.3 Tiempos de acceso a disco.....	293	arquitecturas paralelas	323
12.2.4 Tiempo de acceso a los datos.....	293	14.4 Descripción de microprocesadores avanzados.....	326
12.3 Dispositivos de almacenamiento removibles.....	294	14.4.1 Descripción de la arquitectura Itanium.....	326
12.3.1 Discos ópticos.....	294	14.4.2 Descripción de la arquitectura AMD64	334
12.3.2 Discos magneto-ópticos (MO).....	295	14.5 Resumen	339
12.3.3 Tarjetas de memoria	296	14.6 Contenido de la página Web de apoyo	340
12.3.4 Tarjetas ROM y OTP	296	Bibliografía	341
		Índice analítico.....	345



Información del contenido de la página Web

El material marcado con asterisco (*) sólo está disponible para docentes.

Capítulo 1. Evolución del procesamiento de datos

- Resumen gráfico del capítulo
- Autoevaluación
- Lecturas adicionales:
 - *Las comunicaciones. Conceptos básicos* de Antonio Castro Lechalter y Rubén Fusario, es parte del libro "Telecomunicaciones para Ingenieros de Sistemas" (de próxima aparición) de Alfaomega Grupo Editor (64 páginas). Agradecemos a sus autores por permitir que su escrito sea parte de las lecturas complementarias de esta obra.
- Presentaciones*

Capítulo 2. Sistemas numéricos

- Resumen gráfico del capítulo
- Simulación
 - Herramienta interactiva que permite realizar conversiones y operaciones entre sistemas numéricos.
- Autoevaluación
- Evaluaciones Propuestas*
- Presentaciones*

Capítulo 3. Representación de datos en una computadora

- Resumen gráfico del capítulo
- Simulación
 - Permite ingresar un texto y lo codifica en ASCII.
- Autoevaluación
- Video explicativo (02:44 minutos aprox.)
- Audio explicativo (02:44 minutos aprox.)
- Evaluaciones Propuestas*
- Presentaciones*

Capítulo 4. Aritmética de la computadora

- Resumen gráfico del capítulo
- Simulación
 - Resuelve el algoritmo de Booth paso a paso.
- Autoevaluación
- Video explicativo (02:13 minutos aprox.)
- Audio explicativo (02:13 minutos aprox.)

- Evaluaciones Propuestas*
- Presentaciones*

Capítulo 5. Álgebra de Boole

- Resumen gráfico del capítulo
- Simulación
 - Herramienta interactiva que permite crear el diagrama lógico de una expresión booleana.
- Animación
 - Cómo trabajan los interruptores no mecánicos.
- Autoevaluación
- Lecturas adicionales:
 - *Álgebra booleana* de José A. Jiménez Murillo, es parte del libro "Matemáticas para la Computación" de Alfaomega Grupo Editor (42 páginas). Agradecemos a su autor por permitir que su escrito sea parte de las lecturas complementarias de esta obra
- Video explicativo (01:44 minutos aprox.)
- Audio explicativo (01:44 minutos aprox.)
- Evaluaciones Propuestas*
- Presentaciones*

Capítulo 6. Lógica digital

- Resumen gráfico del capítulo
- Simulación
 - Decodificador de dos entradas.
 - Display BCD siete segmentos.
- Autoevaluación
- Video explicativo (01:34 minutos aprox.)
- Audio explicativo (01:34 minutos aprox.)
- Evaluaciones Propuestas*
- Presentaciones*

Capítulo 7. Diseño de una computadora digital

- Resumen gráfico del capítulo
- Animación
 - Demostración de las distintas fases de la CPU
- Autoevaluación
- Video explicativo (01:53 minutos aprox.)
- Audio explicativo (01:53 minutos aprox.)
- Evaluaciones Propuestas*
- Presentaciones*

Capítulo 8. Microprocesadores

- Resumen gráfico del capítulo
- Autoevaluación
- Video explicativo (02:04 minutos aprox.)
- Audio explicativo (02:04 minutos aprox.)
- Evaluaciones Propuestas*
- Presentaciones*

Capítulo 9. Memorias

- Resumen gráfico del capítulo
- Simulación
 - Ejercicios con memorias.
- Animación
 - Conceptos generales sobre memorias.
- Autoevaluación
- Lecturas adicionales
 - *Memoria* de Martín Silva, es parte del libro "Sistemas Operativos" de Alfaomega Grupor Editor (48 páginas). Agradecemos a su autor por permitir que su escrito sea parte de las lecturas complementarias de esta obra.
- Video explicativo (01:57 minutos aprox.)
- Audio explicativo (01:57 minutos aprox.)
- Evaluaciones Propuestas*
- Presentaciones*

Capítulo 10. Instrucciones

- Resumen gráfico del capítulo
- Autoevaluación
- Video explicativo (02:57 minutos aprox.). Capítulos 10 y 11
- Audio explicativo (02:57 minutos aprox.). Capítulos 10 y 11
- Evaluaciones Propuestas*
- Presentaciones*

Capítulo 11. Software del sistema

- Resumen gráfico del capítulo
- Autoevaluación
- Lecturas adicionales:
 - *El proceso de compilación* de Gustavo López, Ismael Jeder y Augusto Vega, es parte del libro "Análisis y Diseño de Algoritmos" de Alfaomega Grupo Editor (16 páginas). Agradecemos a sus autores por permitir que su escrito sea parte de las lecturas complementarias de esta obra.

- Evaluaciones Propuestas*
- Presentaciones*

Capítulo 12. Dispositivos de entrada / salida

- Resumen gráfico del capítulo
- Autoevaluación
- Video explicativo (01:39 minutos aprox.). Capítulos 12 y 13
- Audio explicativo (01:39 minutos aprox.). Capítulos 12 y 13
- Evaluaciones Propuestas*
- Presentaciones*

Capítulo 13. Transferencia de información

- Resumen gráfico del capítulo
- Autoevaluación
- Evaluaciones Propuestas*
- Presentaciones*

Capítulo 14. Procesadores avanzados

- Resumen gráfico del capítulo
- Animación
 - Demostración de las ventajas del *Pipelining*.
- Autoevaluación
- Video explicativo (01:12 minutos aprox.)
- Audio explicativo (01:12 minutos aprox.)
- Evaluaciones Propuestas*
- Presentaciones*

Vínculos a páginas especialmente seleccionadas sobre Arquitectura de Computadoras.

Glosario

Registro en la Web de apoyo

Para tener acceso al material de la página Web de apoyo del libro:

1. Ir a la página <http://virtual.alfaomega.com.mx>
2. Registrarse como usuario del sitio y propietario del libro.
3. Ingresar al apartado de inscripción de libros y registrar la siguiente clave de acceso

4. Para navegar en la plataforma virtual de recursos del libro, usar los nombres de Usuario y Contraseña definidos en el punto número dos. El acceso a estos recursos es limitado. Si quiere un número extra de accesos, escriba a webmaster@alfaomega.com.mx

Estimado profesor: Si desea acceder a los contenidos exclusivos para docentes, por favor contacte al representante de la editorial que lo suele visitar o escribanos a:

webmaster@alfaomega.com.mx

Convenciones utilizadas en el texto

	Conceptos para recordar: bajo este ícono se encuentran definiciones importantes que refuerzan lo explicado en la página.
	Comentarios o información extra: este ícono ayuda a comprender mejor o ampliar el texto principal.
	Contenidos interactivos: indica la presencia de contenidos extra en la Web.

Prefacio

A lo largo de más de 20 años impartiendo la materia "Arquitectura de Computadoras" redacté una gran cantidad de escritos que resultan útiles para el dictado de mis clases. En conversaciones con colegas llegué a la conclusión de que esos escritos podían ser interesantes para otros docentes: es por eso que escribo este libro, para hacer extensivo mi trabajo a los profesores de otras universidades.

Algunos aportes surgen de la necesidad de explicar de forma más amena conceptos que no siempre son complejos, pero que, al ser producto de malas traducciones o de excesivo tecnicismo, quedan fuera del alcance de los alumnos; otros son aportes los necesarios para poder comprender la materia. En conjunto cubren los contenidos de la asignatura.

Introducción

El propósito de este libro es servir de guía en la enseñanza y el aprendizaje de la materia Arquitectura de Computadoras; para tal fin se organiza en catorce capítulos, orientados al conocimiento gradual de la asignatura.

En el capítulo 1 se introducen los conceptos básicos de la Ciencia de la Computación. Podemos clasificar la temática del capítulo en: conceptos de la Ciencia Informática, recursos de la Informática, elementos fundamentales del hardware, elementos fundamentales del software y estratificación del software. También incluye la clasificación en generaciones de computadoras.

El capítulo 2 tiene por fin la enseñanza de los distintos sistemas numéricos relacionados con el uso de la computadora y la técnica empleada para lograr encontrar las equivalencias entre ellos. Los sistemas de numeración son utilizados luego, en los dos capítulos siguientes, tanto para su operación aritmética (binario), como para la representación sintética de información binaria (octal y hexadecimal).

En el capítulo 3 se desarrollan los conceptos de los sistemas de numeración desde el punto de vista de su relación con la computadora, ya sea para entidades numéricas o alfanuméricas, se introduce el concepto de código y se explica la relación entre el convenio de representación y la declaración de variables en el lenguaje de programación. Se explica la determinación de los rangos de representación en formato de n bits y ciertas consideraciones relacionadas con la precisión de los números; se justifica el uso de variables binarias en la representación, tratamiento y transferencia de información.

El capítulo 4 detalla distintos métodos de operaciones aritméticas, que se desprenden de cómo operan las unidades de cálculo (unidad de enteros y unidad de coma flotante) para binarios de coma fija (enteros) y para binarios de coma flotante. Se introducen las condiciones de *underflow* y *overflow* en resultados y los códigos más sencillos que se utilizan para la detección y/o corrección de errores que puedan ocurrir durante una transmisión.

En el capítulo 5 se le brinda al alumno algunos conceptos básicos para comprender la teoría matemática, que sustenta el diseño de los circuitos que forman parte de una computadora.

En el capítulo 6 se explican métodos de análisis/diseño de los circuitos combinacionales básicos, las operaciones básicas sobre registros y las aplicaciones de estos dispositivos en relación con la ejecución de instrucciones. Se presenta una celda de memoria estática y se explica la representación y uso de arreglos programables para implementar funciones lógicas.

En el capítulo 7 se presenta una computadora digital con sus componentes genéricos: CPU, memoria interna, buses y reloj; se explica de manera detallada su funcionamiento a nivel lógico-digital como no sería posible en una computadora real, dada la extrema complejidad de cada uno de estos componentes. De ésta manera, surgen los conceptos fundamentales para considerar en el funcionamiento de cualquier computadora, como por

ejemplo los ciclos de captación y ejecución de instrucciones, incluyendo en el ciclo de máquina el concepto de interrupción. Todo esto permite, además, que el alumno comprenda la relación entre lo físico y lo lógico, la relación entre lenguajes de distinto nivel y adquiera el concepto de generación de señales de control y de sincronización. La computadora de estudio es una máquina Von Neumann y su inclusión como máquina elemental permite demostrar la importancia del estudio del Álgebra de Boole y de la Teoría de la Lógica Digital, presentadas en los capítulos anteriores.

En el capítulo 8 se pretende lograr el aprendizaje de las funciones internas y de entorno de los microprocesadores, relacionadas con la ejecución de programas. Se explica qué es un chip y cómo se relaciona la tecnología de semiconductores con los microprocesadores, con las memorias y con los microcontroladores. Se presentan las características para evaluar en microcomputadoras, como por ejemplo el número de registros internos, la capacidad de interrupción y la tecnología que sustenta la ejecución de su set de instrucciones. Se explican las posibles políticas para la implementación de ejecución en paralelo a nivel instrucción y demás soluciones tecnológicas que consiguen mejorar el rendimiento.

En el capítulo 9 se explica, desde el diseño más simple, un módulo de memoria semiconductor con componentes RAM hasta el estudio de los modelos de memoria, que incluyen a las memorias de almacenamiento masivo. El propósito es que el alumno interprete las ventajas de una organización de memoria jerarquizada, como estrategia para el desarrollo de máquinas más eficientes. Respecto a la gestión de memoria, se presentan los modelos de paginación, segmentación y segmentos paginados y se presentan en detalle las técnicas de memoria virtual y traducción de direcciones.

En el capítulo 10 se presenta el estudio de los distintos formatos de instrucción y de las diversas modalidades de direccionamiento de operandos, ya que, tanto uno como el otro, forman parte de la descripción de la arquitectura de una computadora. Se muestra la relación que existe en el análisis de un código de instrucción, por ejemplo, cómo cada uno de sus bits permiten establecer la operación que se va a ejecutar, los registros de CPU que están involucrados, la relación con las direcciones de memoria y los distintos métodos que permiten la obtención de los datos.

En el capítulo 11 se desarrolla una introducción, sin intención de profundizar, de los conceptos básicos sobre sistemas operativos y sus principales funciones en la administración de los recursos de una computadora. Se presentan, además, los restantes componentes del software de sistema.

En el capítulo 12 se presentan las funciones y principios de operación de los dispositivos que sustentan los soportes de almacenamiento, evaluándolos como dispositivos complementarios de la arquitectura. Se describen sus componentes físicos y especificaciones técnicas. También se presentan distintas tecnologías de memoria, que se utilizan para brindar una mayor portabilidad e incluso como extensión de la memoria principal.

En el capítulo 13 se presentan las nuevas tecnologías en materia de conectividad, el estudio de las funciones y de los principios de operación de los dispositivos de adaptación y de conexión, estableciéndose una jerarquía que va desde la comunicación más sencilla entre dos buses hasta la descripción de las características de los buses más complejos. También se detallan las diversas modalidades de transferencia de una operación de entrada/salida.

En el capítulo 14 se profundiza el detalle de las técnicas que sustentan el paralelismo a nivel instrucción, luego se presenta la clásica taxonomía de Flynn para desarrollar los distintos modelos de paralelismo a nivel arquitectura. Por último, se describen dos microprocesadores Itanium de Intel y AMD64, con la finalidad de ofrecer al alumno una descripción más amigable que la de un manual.

Para el profesor

El libro está orientado a alumnos que cursan estudios universitarios en carreras asociadas a la Ciencia de la Computación y a la Ingeniería de Sistemas. En las Licenciaturas y en la mayoría de las carreras de Ingeniería (sean éstas en Informática, en Sistemas o en Telecomunicaciones) se cursa en los primeros años de la carrera. Es para estos casos que he desarrollado los capítulos del 2 al 6, con la intención de brindar al alumno los conocimientos básicos necesarios para abordar la asignatura.

En algunas universidades la materia se encuentra en los años de especialización, es decir que el alumno cursa primero materias comunes a varias carreras de Ingeniería y luego las particulares de su carrera. En estos casos, ya habrán tenido asignaturas como Matemáticas Discretas y Técnicas Digitales. Si ésta es su situación, los capítulos 2 a 6 pueden ser considerados como un repaso, o bien puede abordar el capítulo 1 y continuar por el 7.

Se ha intentado secuenciar cada uno de los capítulos con el mismo orden en el que se imparten las clases en la mayoría de las planificaciones consultadas de distintas Universidades de América Latina (como la Universidad Nacional Autónoma de México –Facultad de Contaduría y Administración– y la Universidad Tecnológica Nacional de Argentina).

Para facilitar el dictado de la materia, cada capítulo cuenta con sus correspondientes diapositivas en PowerPoint, de acceso exclusivo para docentes.

También puede descargar exámenes sobre la base de lo explicado en el libro.

Para el estudiante

Es un libro que responde a mi experiencia a través de todos estos años, sobre la base de las dificultades que surgen en cada tema.

El enfoque del libro es claramente didáctico, por lo que no encontrarán referencias a cuestiones como el funcionamiento de un dispositivo externo. Si estas cuestiones son de su interés, lo invito a visitar la Web del libro, que cuenta con numerosos vínculos a las páginas Web de los fabricantes de hardware.

Desde el punto de vista del nivel de profundidad y complejidad, la obra avanza en la medida que avanzan los capítulos; de esta forma acompaña el aprendizaje.

Los numerosos simuladores con que cuenta la obra le permitirán fijar aún mejor los conceptos aprendidos. También cuenta con exámenes *online* para autoevaluarse.

Para el profesional

Encontrará en este libro de texto un refresco de los conceptos clásicos que adquirió en sus años de estudiante y una actualización de los temas de mayor importancia que se conocen en el área.

Objetivos

- Que el docente cuente con herramientas didácticas para la enseñanza de la materia.
- Que el estudiante conozca o repase los temas necesarios para abordar la asignatura.
- Que el estudiante conceptualice los conocimientos necesarios para comprender la Arquitectura de Computadoras.

1

Evolución del procesamiento de datos

Contenido

1.1 Organización y arquitectura de una computadora.....	2
1.2 Estratificación del software.....	3
1.3 Evolución del procesamiento de datos.....	4
1.4 Clasificación de las computadoras.....	8
1.5 Generaciones de computadoras digitales.....	9
1.6 Procesamiento de datos y sistemas de información.....	13
1.7 Sistemas sincrónicos de propósito general.....	15
1.8 Arquitectura de computadoras: Los primeros conceptos.....	15
1.9 Arquitectura de una unidad central de proceso (CPU)	17
1.10 Lógica digital y componentes electrónicos	18
1.11 El Sistema Operativo. La Dinámica del Sistema.....	23
1.12 Resumen	23
1.13 Contenido de la página Web de apoyo.....	24

Objetivos

- Realizar una introducción a la arquitectura de computadoras.
- Diferenciar las distintas generaciones de computadoras digitales.
- Incorporar terminología que apunte al entendimiento del lenguaje técnico, propio del área de competencia.

1.1 Organización y arquitectura de una computadora

La primera pregunta que surge a un lector que recién comienza sus estudios en la ciencia informática es: ¿Qué es una computadora? Como primera respuesta, diremos que en este capítulo la mejor definición será aquella que reúna los aspectos comunes a todas las computadoras, desde una computadora personal hasta una supercomputadora, con prestaciones de baja, mediana o alta complejidad.

Una computadora es un **dispositivo electrónico**, diseñado para aceptar **datos de entrada** y realizar **operaciones** sobre ellos (organizadas en una secuencia lógica y predeterminada por un **algoritmo**), para elaborar **resultados** que se puedan obtener como **salidas**. Un algoritmo computacional se determina por una secuencia de operaciones finita que permite resolver un problema computacional. Se representa con instrucciones que la computadora puede interpretar y ejecutar. Al conjunto de instrucciones que representa un algoritmo se lo denomina programa; expresado de otra manera, un **programa** es la representación de un algoritmo en un **lenguaje** de programación.

Los componentes de una computadora son los dispositivos físicos que le permiten llevar a cabo su función, y que representaremos en el esquema de la figura 1.1.

Computadora: dispositivo electrónico, diseñado para aceptar datos de entrada y realizar operaciones sobre ellos (organizadas en una secuencia lógica y predeterminada por un algoritmo), para elaborar resultados que se puedan obtener como salidas.

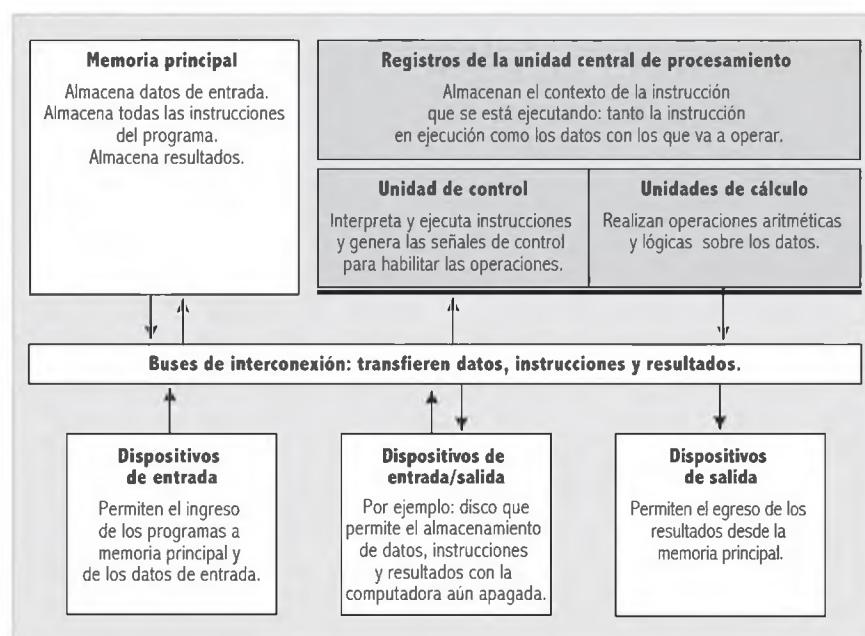


Fig. 1.1. Componentes de una computadora.

La "relación" entre los distintos componentes y su diseño y tecnología, sea en un nivel de detalle como el presentado o en uno menos abstracto, se define como organización de una computadora.

El esquema anterior muestra tres cuadros en gris que constituyen la unidad central de proceso (CPU o *Central Processing Unit*). La "relación" entre los distintos componentes y su diseño y tecnología, sea en un nivel de detalle como el presentado o en uno menos abstracto, se define como **organización de una computadora**. El **set de instrucciones** de una computadora permite representar los algoritmos que solucionan los problemas. Así que para definir la arquitectura de una computadora, a la descripción de los componentes le agrega-

mos la descripción de la manera en que nos comunicamos con ella. Debemos explicar qué puede hacer, es decir que es necesario conocer las **instrucciones** definidas para su CPU, los **tipos de datos** con los que puede operar, las modalidades de acceso a ellos y la forma en que se atienden eventos externos. Cuando nos referimos a la "arquitectura" podemos indicar que una unidad de cálculo permite "determinada" operación con enteros, haciendo abstracción de cómo está implementada en hardware, razón por la cual el manual de un procesador del mercado actual, como el Itanium®, nos indica que la multiplicación de enteros se lleva a cabo en la unidad de cálculo de coma flotante, pero no especifica cómo lo hace. El texto tomado del manual dice así:

"La multiplicación de enteros se ejecuta en la unidad de coma flotante utilizando instrucciones de tipo XMA (instrucciones de tres operandos). Los operandos y el resultado de estas instrucciones se almacenan en registros de coma flotante..."

Cuando un profesional del área de sistemas piensa en términos de arquitectura, tiene en mente las demandas de procesamiento de información que requiere su área de trabajo. Es una mirada desde la funcionalidad de un sistema: se pregunta si necesitaría una o varias computadoras personales, un servidor, una supercomputadora, qué tipo de sistema operativo, etcétera. Cuando un fabricante piensa en términos de arquitectura, tiene en mente las necesidades de procesamiento de un mercado determinado; no es lo mismo fabricar computadoras para el hogar que servidores de red. Los desafíos que se han de resolver en cuanto al diseño de una computadora tienen que ver con la funcionalidad, el alto rendimiento, el bajo costo y la inserción en el mercado.

En el concepto de arquitectura de computadoras se considera la descripción de las características visibles relativas a las prestaciones que una determinada configuración interna de computadoras puede brindar. Como ya indicamos, este concepto incluye los aspectos relacionados con el formato del conjunto de instrucciones que el procesador pueda ejecutar, la representación interna de los datos y el estudio de los módulos de hardware que sostienen la dinámica del conjunto, desde la perspectiva del sistema informático.

La organización de una computadora permite la identificación de los componentes desde el punto de vista de su estructura, la manera en que se relacionan entre sí y las cuestiones de índole tecnológico.

En este libro se tratan ambos conceptos en los distintos niveles desde los que se puede enfocar el estudio de una computadora como herramienta automática en el procesamiento de datos.

Interrupciones: son eventos externos producidos por dispositivos entrada/salida (E/S).

1.2 Estratificación del software

También se pueden establecer niveles funcionales respecto del software. Por un lado, la jerarquía más alta corresponde a los programas de uso particular de los usuarios, denominados **aplicaciones** (que se programan en lenguajes de alto nivel); en el extremo opuesto están las señales que genera la unidad de control para el gobierno de los distintos dispositivos físicos, por ejemplo, una orden de lectura a memoria. Podemos ver la relación entre las distintas jerarquías de software y el hardware en el esquema siguiente:

El concepto de arquitectura de computadoras incluye los aspectos relacionados con el formato del conjunto de instrucciones que el procesador pueda ejecutar, la representación interna de los datos y el estudio de los módulos de hardware que sostienen la dinámica del conjunto, desde la perspectiva del sistema informático.

Aplicaciones: reproductor de video, navegador de Internet, procesador de texto.
Software para producir aplicaciones: editores, compiladores.
Software de gestión de recursos: sistema operativo.
Arquitectura del set de instrucciones.
Lenguaje de señales que permiten la ejecución de las instrucciones .
Hardware



Código de máquina: es el lenguaje que interpreta la CPU y pertenece al nivel de arquitectura del set de instrucciones.

Un usuario que sólo utiliza un software para enviar correo electrónico o *e-mails* se comunica con la computadora con la interfaz gráfica del sistema operativo y no requiere muchos conocimientos en ciencias de la computación. Un programador que desarrolla software de aplicación requiere conocimientos formales en arquitectura de computadoras, en sistemas operativos y, por supuesto, en diseño de algoritmos, lenguajes de programación y estructuras de datos. El programador desarrolla software que, por ejemplo, le sirva a una empresa para administrar su *stock*, su facturación, etcétera.

Un programador que desarrolla software de sistema debe tener conocimientos profundos en arquitectura de computadoras, en lenguajes de programación que le permitan comandar el hardware y en sistemas operativos que le sirvan, por ejemplo, para programar un software de "supervisión" para un dispositivo físico.



La arquitectura del set de instrucciones determina el formato de las instrucciones, los tipos de datos que puede operar, las distintas formas de obtener datos de memoria, que se denominan "modo de direccionamiento", y la forma en que se atienden eventos externos.

Todos los programas se compilan o reciben algún proceso de traducción a **código de máquina**, que es el lenguaje que interpreta la CPU y pertenece al nivel de arquitectura del set de instrucciones. Por efecto de esta "interpretación", la CPU genera señales sincronizadas en el tiempo que controlan el hardware implicado en la operación, por ejemplo, "orden de suma a una unidad de cálculo". Por último, el que realiza la operación es el hardware.

La **arquitectura del set de instrucciones** (ISA o *Instruction Set Architecture*) determina el formato de las instrucciones, los tipos de datos que puede operar, las distintas formas de obtener datos de memoria, que se denominan "modo de direccionamiento", y la forma en que se atienden eventos externos. En los capítulos de este libro se desarrolla cada uno de estos módulos de aprendizaje.

Cada instrucción implica "algo que hacer", un "verbo", que en lenguaje técnico se denomina **código de operación** (grupo de bits que interpreta un diseño específico de CPU). La forma en que se implementan los códigos de operación se denomina nivel de microarquitectura. La **microarquitectura** determina la forma en que se ejecuta la instrucción. Dos CPU pueden compartir el mismo set de instrucciones pero estar diseñadas con distintas microarquitecturas, como es el caso de las CPU AMD, que ejecutan software de la industria *80X86* de Intel, lo que les permite "ejecutar las mismas instrucciones" y mantener así la compatibilidad del software.



Es probable que la primera máquina típicamente digital que el hombre utilizó para resolver problemas aritméticos haya sido el ábaco.

1.3 Evolución del procesamiento de datos

1.3.1 Los comienzos de la computación

Desde épocas remotas (alrededor de 3.000 años a.C.) el hombre trató de liberarse de hacer cálculos en forma manual. Es probable que la primera máquina típicamente digital que utilizó

para resolver problemas aritméticos haya sido el **ábaco**. Ya en la Era Grecorromana se usaron varias versiones de este dispositivo, que también se utilizó en Egipto y en China. No obstante, según las teorías de físicos como Galileo, Descartes y Newton, estos instrumentos de cálculo no se desarrollaron en la Europa Occidental hasta el siglo XVII.

En la primera mitad del siglo XVII **John Napier** introdujo el concepto de logaritmo, con el que la tarea de multiplicar se simplificó. A partir de ese momento, se comenzaron a construir las máquinas de cálculo llamadas analógicas o máquinas de medida. Es factible que Napier sólo haya descubierto un dispositivo físico para hacer más rápida la multiplicación. Estas máquinas fueron de uso habitual en el siglo XVII y todavía se las puede ver en varios museos.

1.3.2 La primera máquina y su evolución

En 1642 **Blaise Pascal** construye, en Francia, una máquina para su padre –empleado contable– con la que tuvo gran éxito; esta máquina fue considerada la primera calculadora digital, llamada así porque acumulaba las operaciones aritméticas –suma y sustracción– en un acumulador o contador de enteros. Su mecanismo se basaba en ruedas dentadas que tenían 10 posiciones (de 0 a 9); cada vez que una rueda pasaba de 9 a 0, la rueda inmediatamente a la izquierda avanzaba una posición. En el presente las máquinas de oficina y las computadoras utilizan el mismo principio, sustituyendo el mecanismo de ruedas dentadas por un circuito electrónico.

En 1671 **Gottfried Wilhelm Von Leibniz** inventó una máquina que permite automatizar la multiplicación por sumas sucesivas. El mecanismo que utilizaba era una combinación de engranajes que permitía la multiplicación y la división de números en el sistema binario.

Charles Babbage empezó a construir la primera computadora digital en 1823 con la ayuda del gobierno británico. Incorporó una rutina de operaciones en tarjetas perforadas –en términos modernos, un programa perforado– que representó un gran paso para su próxima máquina.

En 1833 concibió la idea de una calculadora digital universal, a la que llamó máquina analítica. Esta máquina no se pudo construir, por falta de tecnología apropiada, hasta un siglo después.

En el siglo XIX se hicieron grandes avances en física matemática y se lograron mejorar los instrumentos de cálculo.

Para evitar los problemas de Babbage, se desarrolló una máquina nueva, que no era digital sino analógica, y que se llamó **máquina de medidas**, porque los resultados se obtenían midiendo la salida de los dispositivos. Ésta también se denominó máquina continua, porque la información que se obtenía a la salida era una representación en una magnitud continua, análoga a la real. Estos dispositivos podían ser rápidos aunque no muy precisos, debido a que dependían de analogías y medidas físicas.

En el siglo XIX **George Boole** desarrolló un Álgebra que no utiliza números, sino que establece la relación entre conceptos lógicos. Se hizo un paralelismo entre las leyes del pensamiento y las operaciones algebraicas. Esto permitió la representación de conceptos lógicos en términos algebraicos y se denominó **Lógica booleana**. El **Álgebra de Boole** es un ente matemático que fundamenta los principios de la teoría de circuitos.

1.3.3 La máquina de tarjetas perforadas

Mientras los trabajos sobre máquinas analógicas seguían desarrollándose, hubo una revolución en el campo digital cuando **Herman Hollerith** (de la oficina de censos de los EE.UU.)



Ábaco: objeto que sirve para facilitar cálculos sencillos (sumas, restas, multiplicaciones) y operaciones aritméticas. Se trata de cierto número de cuentas engarzadas con varillas, cada una de las cuales indica una cifra del número que representa.



Napier (1550-1617). Matemático escocés, reconocido por haber descubierto los logaritmos o "números artificiales".



Pascal (1623-1662). Matemático, filósofo y teólogo francés, considerado el "padre de las computadoras" junto con Babbage.



Von Leibniz (1646-1716). Filósofo, matemático, jurista y político alemán. Descubrió el cálculo infinitesimal, independientemente de Newton, e inventó el sistema de numeración binario en que se basan casi todas las arquitecturas de computación actuales.



Un programa es la representación de un algoritmo en un lenguaje de programación.



Babbage (1791-1871). Matemático británico y científico de la computación, considerado "el padre de las computadoras" junto con Pascal.



Hollerith (1860-1929). Estadístico estadounidense que inventó la máquina tabuladora. Es considerado el primero en lograr un tratamiento automático de la información.

inventó la técnica para procesar gran cantidad de datos por medio de tarjetas perforadas, para luego clasificar y analizar los datos perforados en ellas.

Esta técnica se aplicó en los censos de 1890 en los EE.UU. y de 1911 en Gran Bretaña.

Las ideas de Hollerith fueron tomadas y perfeccionadas por la empresa IBM (International Business Machines). IBM desarrolló un dispositivo básico conocido como tarjeta perforada de 80 columnas. Cada tarjeta era leída por una lectora que permitía detectar las perforaciones en el soporte mediante conmutadores eléctricos. Cuando estos contactos atravesaban las celdillas perforadas, la unidad interpretaba el dato según fuera la combinación de perforaciones en cada columna.

Desde alrededor de 1930 hasta la década de 1970 la tarjeta perforada desempeñó un papel importante en el procesamiento de datos y reemplazó en gran medida al procesamiento manual. El medio básico para el procesamiento era la tarjeta perforada que contenía 80 columnas por 12 filas.

La combinación de zona y dígito permitía obtener una configuración distinta para cada letra, número o carácter especial. En 1969 IBM empezó a utilizar una tarjeta de 96 columnas para el Sistema/3 (8,25 x 6,68 cm) que se perforaba con agujeros circulares.

Los periféricos que permitieron su utilización fueron:

- La lectora de tarjetas.
- La perforadora de tarjetas.
- La lectoperforadora de tarjetas.

En el ejemplo siguiente se puede ver la utilidad de la tarjeta perforada como soporte para instrucciones y datos. En la figura 1.2 se representa el bloque procesador y la memoria que reciben las instrucciones del programa y los datos perforados en lotes de tarjetas. O sea que el sistema operativo organizaba el procesamiento comenzando con una orden de lectura a la lectora de tarjetas. En una tarea se separaba el lote de instrucciones del lote de datos con tarjetas de control que marcaban el comienzo y el fin de las tarjetas de instrucciones, así como el comienzo y el fin de las tarjetas de datos; cuando se finalizaba la lectura, las instrucciones y los datos quedaban almacenados en la memoria y recién entonces el sistema operativo ordenaba la ejecución. Por último, los resultados obtenidos se imprimían en papel.



Las instrucciones de un programa se ejecutan unas tras otras, siguen una lógica secuencial, salvo que haya una instrucción de salto.

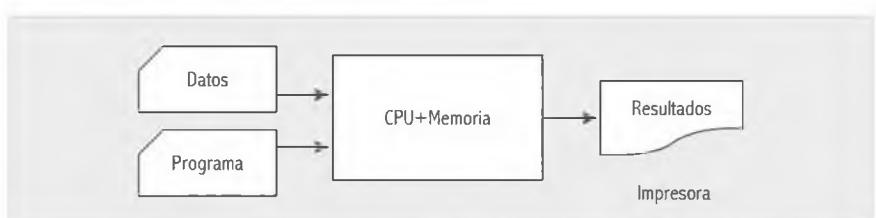


Fig. 1.2. Lectora de tarjetas. Cada instrucción como mínimo en una tarjeta.

La cinta de papel, también un medio primitivo de almacenamiento (al igual que la tarjeta perforada), quedó en desuso. Los datos se perforaban en ella en la forma de agujeros circulares pequeños. Las cintas se utilizaron en mayor medida en máquinas de sumar, máquinas de contabilidad y cajas registradoras.

1.3.4 La calculadora secuencial automática (IBM)

En 1939 comenzaron los trabajos en gran escala para lograr automatizar y poner en funcionamiento la máquina diferencial de Babbage, con el propósito de tabular polinomios.

La computadora secuencial automática de IBM fue puesta en operación en 1944 en la **Universidad de Harvard**, a cargo del físico **Howard Aiken** (cuyo trabajo fue subvencionado por IBM y la Universidad). Esta máquina constaba de partes electromecánicas provistas por IBM y estaba controlada por una cinta de papel perforada (similar a la tarjeta perforada).

Después de ésta, se diseñaron otras máquinas electromecánicas. Una fue la que Aiken llamó MARK II, utilizada por la Marina de los EE.UU.

Durante la Segunda Guerra Mundial se desarrolló la computadora ENIAC o *Electronic Numerical Integrator And Calculator*, en la cual el cambio de sus programas se hacia mediante el recableado de unas borneras, operadas por técnicas.

1.3.5 El programa almacenado

La ejecución de una instrucción de ruptura de secuencia permite que, en determinado lugar del programa, se salte a una instrucción que no es la siguiente. En su diseño original, ENIAC era capaz de almacenar distintos programas. Para pasar de uno a otro, los ingenieros tenían que modificar parte de los circuitos de la máquina, con el fin de que ésta efectuara las operaciones requeridas para la solución de cada problema específico.

En 1945 John von Neumann logró una **máquina de programa almacenado** a la que se denominó **computadora**. Esta máquina no fue diseñada para una aplicación concreta, sino que se trató de una **máquina de propósito general**, capaz de almacenar instrucciones y datos en una memoria. Esto permite sustituir el conexionado fijo entre los componentes de la máquina por un programa de instrucciones intercambiable.

La máquina de von Neumann se fundamenta en tres principios que en el presente todavía se aplican:

1. Máquina electrónica digital, que trabaja con información codificada en binario (digital binario = 0,1 = dos estados).
2. Programa almacenado en memoria.
3. Posibilidad de provocar una ruptura de secuencia de instrucciones en un programa.

La figura 1.3 responde al bosquejo de una computadora von Neumann, que, como se puede apreciar, no se diferencia en nada respecto del presentado en la figura 1.1. Está formada por los módulos CPU, memoria y unidades de E/S. La CPU cuenta con una unidad que procesa datos denominada unidad aritmético-lógica (ALU o *Arithmetic Logic Unit*) y otra llamada unidad de control y secuenciamento (CU o *Control Unit*). Esta unidad emite órdenes para llevar a cabo en forma secuencial y sincronizada las operaciones elementales que permiten la ejecución de instrucciones. Como vemos en la figura 1.3, las órdenes a los módulos son generadas desde la CU. Podemos afirmar que una posible orden a la memoria es una orden de lectura y una orden a la ALU puede ser una orden de suma. La CU es también un canalizador de datos, ya que su función principal es gestionar la transferencia de información almacenada en módulos diferentes. Por esta razón, los módulos están relacionados entre sí por colectores de datos e instrucciones denominados también **buses**. Sobre estos colectores se puede crear gran cantidad de rutas imaginarias. Si se supone que una ruta es un camino posible entre un origen y un destino, cuando sea necesaria una transferencia, la CU la habilitará desde un sólo origen y hacia un sólo destino.



Aiken (1900-1973). Ingeniero estadounidense, pionero en computación al ser el ingeniero principal tras la creación de un dispositivo electromecánico de computación.



La ejecución de una instrucción de ruptura de secuencia permite que en determinado lugar del programa se salte a una instrucción que no es la siguiente.

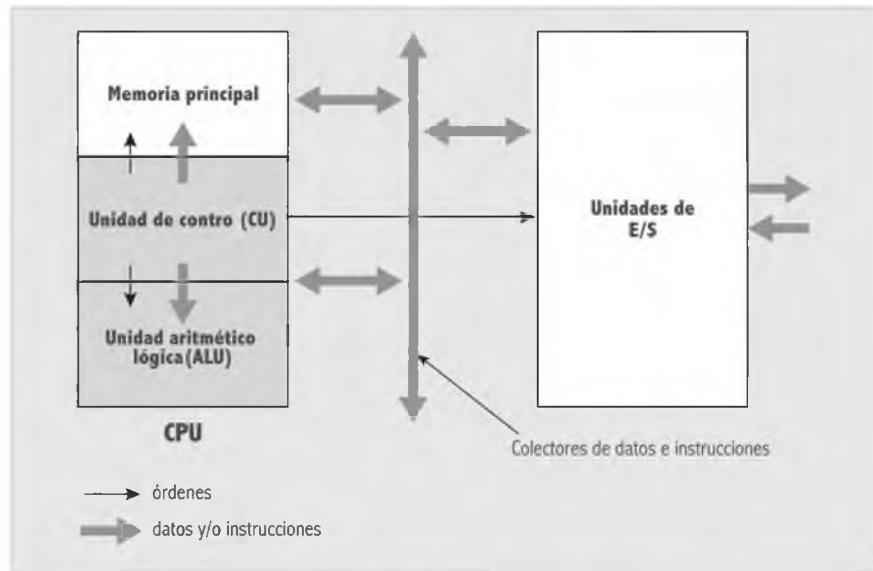


Fig. 1.3. Esquema básico de una computadora Von Neumann.

1.4 Clasificación de las computadoras

1.4.1 Analógicas

Las primeras computadoras analógicas se implementaron para estudiar un modelo semejante (análogo) a una ecuación; el resultado se elaboró tras la medición del valor que asumían las salidas del modelo. Sin embargo, el sistema adolecía de dos desventajas importantes: por un lado, la falta de exactitud en la salida, a causa del carácter continuo de esta magnitud, y por el otro, el modelo construido, que representaba a una única aplicación y no servía para otra.

En este momento, por ejemplo, se utilizan sistemas analógicos modernos para procesos que involucran la toma de medidas en industrias (refinerías de petróleo), simuladores de vuelo, simuladores de redes eléctricas y otras aplicaciones en las que sea importante representar la variación en el tiempo de magnitudes continuas.

La ventaja más destacada que ofrece esta computadora es su capacidad de procesar datos no discretos (temperaturas, presión, altura), además de permitir la simulación de modelos en la etapa de desarrollo de un proyecto; así, en un modelo más pequeño que el real las variables se representan con magnitudes proporcionales, lo que genera la consecuente disminución de los costos de desarrollo.

1.4.2 Digitales

Estas computadoras se denominan digitales porque procesan "dígitos" binarios, "ceros" y "unos" que representan datos. Un dato es un binario que corresponde a un código preestablecido. Entre las ventajas que presentan podemos destacar que efectúan cálculos precisos y que el modelo se arma sobre un programa intercambiable, lo que posibilita no tener que cambiar partes físicas para modificarlo, sino sólo ingresar un programa nuevo.

Para procesar variables continuas y estudiar el modelo a fondo en este tipo de computadoras, es necesario ingresar datos con diferencias infinitesimales, motivo por el cual su utilización en estos casos resulta inadecuada.

1.4.3 Híbridas

Una variable analógica puede asumir infinitos valores dentro de un rango y se utiliza para representar cantidades "naturales", como la temperatura, la presión o la distancia. Sin embargo, a efectos de una medición que pueda ser interpretada por los seres humanos, se las convierte a valores discretos, por ejemplo, se mide la temperatura en grados o la distancia en metros o pulgadas. Esto elimina el carácter "infinito" de los posibles valores que la variable pueda asumir.

Los sistemas híbridos son una combinación de analógicos y digitales. Mientras que la porción analógica se encarga de tomar los datos continuos (temperatura, presión, etc.), la parte digital efectúa los cálculos. Estas computadoras se construyen para propósitos especiales; un ejemplo actual es el GPS de nuestro auto.

1.5 Generaciones de computadoras digitales

Según la tecnología con la que operan, las técnicas de organización y su explotación se establece la siguiente clasificación de las computadoras digitales:

1.5.1 Computadoras de 1^a generación

Estas computadoras estaban constituidas por válvulas de vacío, que disipaban gran cantidad de calor y ocupaban una superficie muy amplia. Las tareas se ejecutaban en forma secuencial, lo que implicaba que:

1. El programa, almacenado en tarjetas o cintas perforadas, era cargado en memoria principal por un programa llamado cargador, perteneciente al sistema operativo.
2. Se ejecutaba el programa instrucción por instrucción.
3. Se imprimían los resultados.

Las operaciones de entrada, procesamiento y salida de los datos se encontraban encadenadas en el tiempo, por lo que la duración del proceso era igual a la suma de todas las operaciones.

Las computadoras de 1^a generación se utilizaron durante el período comprendido entre 1954 y 1959, mientras que las fabricadas antes de 1954 se tomaron como máquinas experimentales y, por ello, no se incluyen dentro de esta generación.

1.5.2 Computadoras de 2^a generación

Las computadoras de 2^a generación estaban constituidas por transistores y utilizaron circuitos impresos, lo que permitió reducir el tamaño con respecto a las anteriores. Posibilitaron la simultaneidad entre un cálculo y una operación de E/S. Este concepto en la práctica dio pocos resultados, debido, en gran medida, a la desproporción entre la velocidad de cálculo interno y las velocidades de E/S, que hacían que la CPU no se utilizara más que en un pequeño porcentaje de tiempo. El paliativo para este problema fue que las operaciones de E/S se realizaran utilizando como soporte de almacenamiento unidades de cinta magnética, mucho más rápidas que las lectoras de tarjetas y las impresoras. Para lograrlo, se copiaba la información contenida en el soporte tarjeta a soporte cinta magnética y de ésta a impresora con un proce-

sador auxiliar. Este procedimiento de explotación, cuyo esquema se observa en la figura 1.4, se denominó procesamiento por lotes. En esta modalidad de procesamiento, cada lote estaba compuesto por varios trabajos y era preciso esperar que el lote cargado en la cinta magnética se procese por completo para obtener los resultados de cada trabajo.

El período de explotación de estas computadoras fue el comprendido entre 1959 y 1964.

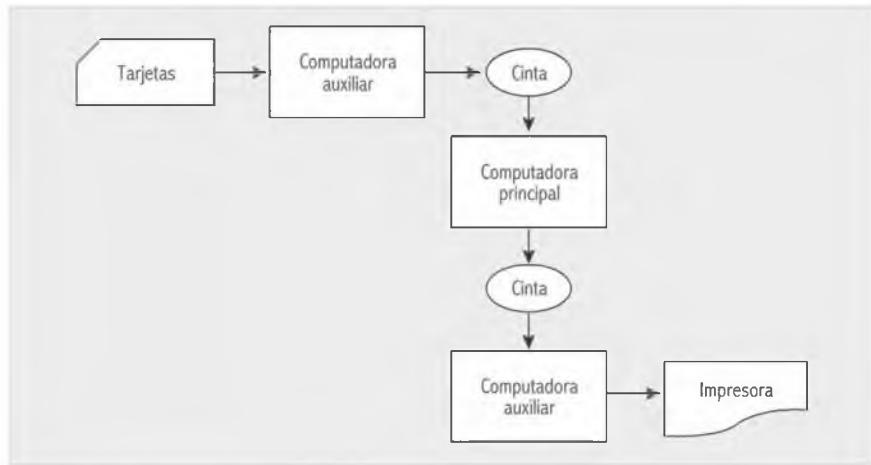


Fig. 1.4. Procesamiento por lotes.

1.5.3 Computadoras de 3^a generación

A partir de 1964 comenzó la 3^a generación de computadoras con tecnología de circuitos integrados (varios componentes electrónicos incluidos en un solo bloque de silicio). Estos circuitos eran del tipo SSI o *Small Scale Integration* (pequeña escala de integración) y MSI o *Medium Scale Integration* (mediana escala de integración) y permitieron el incremento de la velocidad interna de la computadora y la reducción de la energía que utilizaban.

En esta etapa se pudo explotar la **multiprogramación**, método que consiste en que varios programas residen en forma simultánea en la memoria en estado de ejecución. En un instante dado sólo uno de ellos utiliza la CPU, pero los otros pueden efectuar operaciones de entrada/salida en forma simultánea. Cuando el programa que ejecuta la CPU se detiene en espera de una operación de entrada/salida, otro programa toma su lugar, deja al primero suspendido y evita, así, que se produzcan tiempos inactivos en la CPU.

Las computadoras de 3^a generación dividen su memoria "lógicamente" en dos zonas: una reservada a los "trabajos del usuario" y la otra a la "conversión de soportes y carga". A simple vista podría afirmarse que las particiones corresponden a la computadora principal y a la auxiliar de la 2^a generación, respectivamente. Sin embargo, hay una diferencia importante: la "carga por lotes" se sustituyó por la "carga continua". Los trabajos se ponen en **cola de espera** en un disco magnético y el sistema operativo es el que se encarga de ejecutarlos según un nivel de prioridad. Los resultados, que son transferidos al disco, luego son extraídos por la impresora. En la figura 1.5 se representa el esquema.

Multiprogramación: método que consiste en que varios programas residen en forma simultánea en la memoria en estado de ejecución.

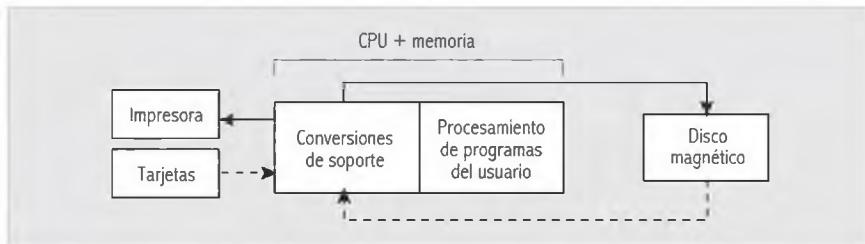


Fig. 1.5. Procesamiento por lotes en la 3^a generación.

Se hace necesario establecer la diferencia entre la multiprogramación y el multiprocесamiento. El término **multiprocесamiento** se utiliza cuando se procesan varios programas, en forma simultánea, en CPU diferentes que se relacionan entre sí. A partir de la 3^a generación, es posible la gestión denominada **teleproceso** o procesamiento a distancia. El teleproceso es un sistema de carga directa, con la ventaja de que los datos pueden ingresar y egresar por terminales remotas según una prioridad dada. Durante este periodo se desarrollan los primeros sistemas **interactivos**, que permiten que el usuario intervenga en la actividad del procesamiento. El diálogo se gestiona utilizando un terminal con menús o cuestionarios que el sistema formula y el usuario responde.

A fin de atender gran número de procesos, una computadora puede asignar en forma alternada una parte del tiempo de CPU a cada uno, esto produce un efecto de aparente simultaneidad de ejecución. Esta forma de organizar el procesamiento se denomina **tiempo compartido** o **time sharing**.

Cuando un sistema admite la atención de múltiples usuarios se denomina **multiusuario**. Todo sistema multiusuario es **multitarea** y, además, se ocupa de la administración de los recursos asignados a los usuarios.

Por último, en esta etapa se desarrolla el concepto de **máquina virtual**, que simplifica la labor del programador, ya que éste no conoce más que una máquina ficticia creada y controlada por el sistema operativo, que no presenta limitaciones de configuración (sobre todo en capacidad de memoria central), provocada en gran medida por la actividad compartida de varios usuarios. Esta generación se extendió hasta 1971.

1.5.4 Computadoras de 4^a generación

Se considera que el período de la 4^a generación de computadoras está comprendido entre 1971 y 1987. La tecnología aplicada en la fabricación de circuitos pertenece a la clasificación LSI o *Large Scale Integration* –escala de integración grande–, que permitió incluir una CPU completa en una sola pastilla, que se denomina **microporcesador**. En esta etapa el procesamiento se realiza en mayor medida en **tiempo real**. Considerando los sistemas interactivos, se hace posible la consulta y la actualización de datos, así como el acceso a **grandes bancos de datos** que utilizan, incluso, unidades inteligentes distribuidas en redes, como los sistemas de cajeros automáticos bancarios. Se desarrollan nuevas utilidades a partir de la sencilla comunicación usuario-máquina: enseñanza asistida por computadora, consultas telefónicas que entregan una respuesta oral al usuario, regulación automática de semáforos, control automático de procesos relativos a una línea de producción manufacturera, diseño de proyecto asistido por computadora, hojas de cálculo y poderosos procesadores de texto que automatizan prácticamente el total de las tareas de oficina.



Cuando un sistema admite la atención de múltiples usuarios se denomina **multiusuario**.



Una CPU completa incluida en un solo chip se denomina **microporcesador**.

Hasta esta etapa, los avances tecnológicos estuvieron concentrados en lograr mejorar el hardware de la computadora y así obtener equipos más pequeños, menos costosos y más rápidos. A partir de entonces, también se puso atención en la necesidad de mejorar el software para que permitiera una mayor velocidad de procesamiento, ya que los avances en el hardware parecían agotados.

1.5.5 Computadoras de 5^a generación

En la década de 1980 se llevó a cabo una revolución en la concepción de diseño de una computadora (arquitecturas RISC, pipelines, superescalaridad, niveles de caché, etc.) y se desarrollaron los primeros avances considerados como el ingreso en una nueva etapa, la 5^a generación de computadoras. En ella, las supercomputadoras desarrollaron funciones inteligentes, basadas en experiencias sobre inteligencia artificial. Las cuatro generaciones anteriores se definieron con precisión, pero para la mayoría de los autores la 5^a todavía está en proceso, debido al actual desarrollo de programas de investigación, cuyo paradigma propone que es necesario diseñar funciones inteligentes capaces de adaptarse a aplicaciones que así lo requieran. En la mayoría de las arquitecturas propuestas hasta el presente se hace claramente visible la distancia entre el hardware y el software. Un objetivo de las nuevas tendencias es que éstos tiendan a unirse.

En los procesadores actuales se ejecutan varias etapas de las instrucciones en paralelo y éste es un ejemplo de avance respecto de las máquinas anteriores, que procesaban cada etapa de una instrucción en un orden secuencial estricto. Sin embargo, las expectativas de esta nueva generación apuntan a que los cambios crean una revolución profunda y no una mejoría gradual de las computadoras anteriores, como la exemplificada. Los objetivos son por demás ambiciosos. Los precursores de esta idea indicaron que uno de los parámetros que permite dar un gran salto evolutivo es la comprensión del lenguaje hablado, el reconocimiento de las formas, la lectura de información escrita y, en definitiva, el reconocimiento de toda forma de comunicación humana. Se resalta la necesidad de una interfaz hombre/máquina de elevada inteligencia, que demuestre cualidades de juicio similares a las del hombre.

El Gobierno japonés es uno de los más destacados propulsores de este proyecto; no obstante, más allá de los orígenes del programa de 5^a generación, su paulatina concresción influye gradualmente en los problemas sociales y económicos de todos los países del mundo, y es por esto una iniciativa internacional. Ya en octubre de 1981 se celebró en Tokio una conferencia sobre el tema, cuya finalidad era buscar la colaboración occidental de manera de aunar esfuerzos. Las características de las máquinas de 5^a generación dependen de proyectos ya en marcha, en los que se diferencian tres áreas fundamentales de desarrollo: integración de microcircuitos en muy alta escala, nuevos métodos de procesamientos de datos, nuevas arquitecturas y avances en el diseño de memorias con mayor capacidad y velocidad.

Respecto de la arquitectura de computadoras, una postura es la que divide las máquinas de quinta generación en niveles. Los superiores se encargan de la aplicación y la interfaz hombre/máquina en lenguaje natural: un procesador de conocimientos implementado sobre una máquina virtual con una base de datos y una red de comunicación, una configuración secuencial de procesadores de propósito específico, un procesador vectorial, otro de comunicaciones, uno de base de datos y otro de sistema operativo.

El procesador de conocimiento es el corazón de esta máquina, el lenguaje de este núcleo es de inferencia lógica, de manera de dotarlo de capacidad de resolución de problemas. El procesador de base de datos representaría el centro neurológico del procesador de conocimiento. La complejidad de los proyectos en investigación supone la concreción de muchos proyectos individuales que no terminarán su evolución al mismo tiempo. En algunos de ellos la propuesta plantea el alejamiento de la tendencia actual hacia las máquinas de propósito general y se incrementa el número de procesadores de propósito dedicado.

Se sustituyen lenguajes de alto nivel por otros de mayor capacidad para la manipulación de símbolos y recursos lógicos, semejantes a los precursores utilizados en los comienzos de la inteligencia artificial, Prolog y Lisp.

Es evidente que las computadoras van a participar de manera muy activa en su propia evolución; los diseños de alta prestación requieren cada vez más de la inteligencia de las computadoras, independientemente del tipo de material eléctrico que las sustente.

El objetivo del programa de 5^a generación supone que las computadoras tengan capacidades cognitivas superiores a la humana. Por lo tanto, los distintos soportes de información deberán contener los datos de la información cultural que posee el hombre, además de los conocimientos específicos o expertos de una materia en sí. Éste es el campo que se conoce como Ingeniería del conocimiento; su objetivo es verter en la memoria de la computadora un vasto cúmulo de conocimientos útiles y que ésta, a su vez, pueda relacionarlos. Esto implica una mayor densidad de memoria y un ajuste en su organización, que depende del desarrollo de sistemas más eficientes, ya sea en la tecnología de estado sólido como en las memorias de masa. El desarrollo de tecnologías para almacenamiento de información es esencial para realizar tareas inteligentes, puesto que la información debe almacenarse a corto o a largo plazo, procesarse y recuperarse cuando sea necesario.

1.6 Procesamiento de datos y sistemas de información

Para interpretar qué se entiende por **procesamiento de datos y sistemas de información**, primero debemos definir algunos conceptos. Los **datos** son conjuntos de símbolos que representan un objeto concreto o abstracto, mientras que la **información** es la consecuencia de procesar los datos para que tengan un significado. Así, la información permite disminuir la incertidumbre y facilita la toma de decisiones más acertadas. Procesar datos implica que se relacionen entre sí o que formen parte de un contexto, o ambas situaciones; por ejemplo, "38 °C" es un dato aislado que si se contempla en el contexto del pronóstico del clima nos permite prepararnos para una jornada de calor, pero si se trata de la temperatura corporal nos indica que es preciso tomar una aspirina y llamar al médico.

Los datos que procesa una computadora no tienen significado para ella, sino que lo tienen para el usuario que los recibe y puede interpretarlos en su propio contexto. De lo expresado antes se desprende que no todos los datos representan información, pero que la información implicó la elaboración de un conjunto de datos.

Los términos **sistema de información** o **sistema de informática** se utilizan para referirse al conjunto coordinado de elementos, datos y procesos, cuya interacción permite la obtención de la información; en el caso de sistemas automatizados se utiliza como "unidad de procesamiento" la computadora (hardware + software), o bien un conjunto de computadoras relacionadas entre sí.

En un sistema de información deben coordinarse los procesos y los datos del sistema que administra la computadora y los procesos y los datos del usuario. Siempre todos cooperan entre sí para la elaboración de la información (**procesamiento de datos** o **cómputo de datos**).

Un proceso especifica la actividad interna del sistema y estará representado por una secuencia de instrucciones pertenecientes a uno o varios programas relacionados, de manera que verifique la serie finita de operaciones definida en un **algoritmo**.

Procesar datos significa realizar alguna operación sobre ellos (véase la figura 1.6). Por lo tanto, los "datos de entrada al proceso" (**input**) se transformarán en "resultados" (**output**), en general, luego de que alguna de estas operaciones los afecte:



Datos: conjuntos de símbolos que representan un objeto concreto o abstracto.



Información: datos procesados con significado en un contexto real que permita al usuario la toma de decisiones.



Sistema de información o sistema de informática: se refiere al conjunto coordinado de elementos, datos y procesos, cuya interacción permite la obtención de la información.

- Cálculo.
- Comparación.
- Clasificación (ordenamientos, intercalaciones).
- Transformación (por ejemplo, de datos numéricos a gráficos).
- Transferencia.
- Almacenamiento (para usarlos con ulterioridad).
- Consulta.



Fig. 1.6. Esquema de proceso de datos.



El conjunto de los resultados constituye **información**, ya que los datos elaborados o procesados están incluidos en un contexto válido para quien los interpreta y no son elementos aislados. Esto implica que el objetivo del **procesamiento de datos** es obtener **información**.

Los datos de entrada son, internamente, una estructura binaria organizada que respeta un convenio dispuesto de antemano, denominado **código**. Éste permite establecer una relación entre dígitos binarios (símbolos de la computadora) y los símbolos que el usuario utiliza para construir datos: números, letras, caracteres especiales, auditivos, gráficos o visuales. Toda operación que se realice sobre los datos debe practicarse en cierto orden; así, los resultados de una operación pueden ser usados por otras, o sea, en orden secuencial. La secuencia lógica de pasos se describe con la confección de un programa. Las salidas también son información codificada en binario (lenguaje poco amistoso para el usuario) y, para que se represente en símbolos fáciles de comprender, se decodifican. La **decodificación** convierte el código binario a código de usuario (letras, números, puntos, comas, etc.).

Un dato puede ser una cadena de caracteres o *string*, que ingresa por teclado y se ve en el monitor. Sin embargo, la mayoría de los procesos en un sistema informático toma y vuelve datos organizados en estructuras mayores denominadas **archivos** (*files*), según se puede observar en la figura 1.7.

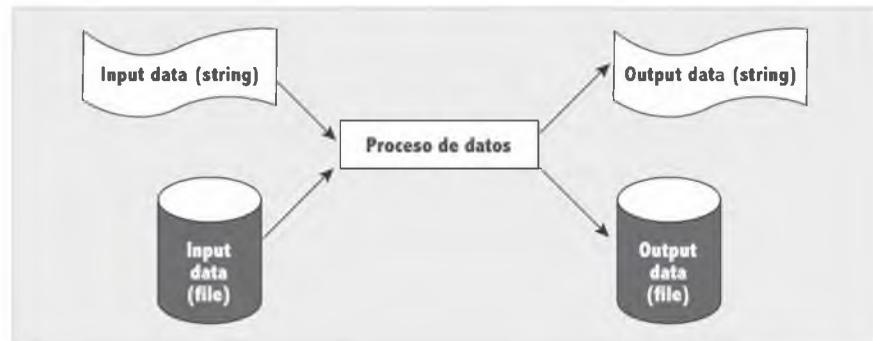


Fig. 1.7. Entrada/salida en archivos.

1.7 Sistemas sincrónicos de propósito general

Hay sistemas orientados a tareas que no cambian. En estos sistemas, llamados de **propósito específico**, la tarea que debe realizar la computadora es fija y está definida por los circuitos que constituyen una unidad de control; por lo tanto, la memoria sólo soporta datos de entrada y resultados obtenidos como salida. Un cambio en la especificación de la tarea implica una alteración del hardware del procesador. Este tipo de computadoras se emplea en aplicaciones que no requieren cambios continuos.

En los sistemas llamados de **propósito general** o **computadoras** la tarea asignada a la computadora se puede cambiar, está definida en software y se denomina **aplicación**. La CPU recibe las instrucciones desde la memoria principal, donde fueron almacenadas previamente junto con los datos de entrada.

Estos sistemas requieren que primero se desarrolle la tarea de programación, que consiste en la elaboración de un algoritmo y su representación en un lenguaje. El resultado de esta actividad se denomina **programa**. La tarea de la computadora cambia, en tanto cambie el programa almacenado en memoria. Cada instrucción del programa es transferida desde la memoria a la CPU, y es el módulo **unidad de control** (CU) el que se encarga de organizar en el tiempo el conjunto de operaciones elementales que permitan su ejecución. A cada una de estas operaciones elementales se las denomina **microoperaciones**.



En sistemas de propósito específico, la tarea que debe realizar la computadora es fija.

1.8 Arquitectura de computadoras: Los primeros conceptos

La tarea fundamental de una computadora digital es realizar cálculos o comparaciones entre datos u operandos. La palabra digital implica que el valor de los datos es discreto y limitado y se representa con combinaciones de **0** y **1**, o sea, son máquinas binarias. Los datos son agrupaciones de bits que, como ya señalamos, al aplicar distintas técnicas de codificación se convierten en números y letras. Los comandos e instrucciones que controlan la operación de la computadora también se codifican en binario, de modo tal que los bits constituyen el "alfabeto" de la computadora.

Una computadora es un sistema que incluye módulos de hardware y de software. El **hardware** es el conjunto de dispositivos electrónicos y electromecánicos que constituyen la estructura física de la computadora. Es la porción "dura", lo tangible (*hard* = duro; *hardware* = ferretería). En tanto que la parte lógica o "lógica", se denomina **software**, que es el nombre que identifica al conjunto de programas para procesar datos en un equipo de computación, esto es, su parte lógica (*soft* = blando). Sin el software, el hardware no podría procesar dato alguno o quedaría limitado a una tarea fija, como decíamos en procesadores de propósito específico. Un término más, que involucra ambos conceptos, es **firmware** y se utiliza para identificar los dispositivos físicos programados, como puede ser la programación de los múltiples usos que presta un electrodoméstico, o sea, que se fusionan los conceptos de hardware y software.

Por ejemplo, cuando se enciende la computadora se pone en funcionamiento el primer elemento software, un programa que lo hace apto para que el usuario lo pueda utilizar. En una PC (o *Personal Computer*), parte de este programa de arranque se define a nivel físico (*firmware*) junto con otros servicios dedicados a la atención rápida de entrada o salida de información, por ejemplo, servicio de atención del teclado.

Si se tiene claro que el esfuerzo empleado en el desarrollo de computadoras tiene como objetivo el tratamiento automático de datos, es imprescindible, entonces, conocer los fundamentos de los datos y sus formatos.



Bit es el acrónimo de *binary digit* (dígito binario) y es la unidad mínima de información.

Cómo decíamos, en las computadoras **digitales** las entidades (datos e instrucciones) están constituidas por agrupaciones de bits. **Bit** es el acrónimo de *binary digit* (dígito binario) y es la unidad mínima de información. Por ejemplo, si en un formulario se pide "marque con una cruz" la opción verdadera y "deje en blanco" las que considera falsas, las opciones verdaderas serán identificadas con un "**1**" en el campo de "opción marcada" y las falsas con un "**0**".

Hay dos valores de bits diferentes, "**0**" y "**1**", que tienen el significado numérico obvio, *0* y *1*. También se puede pensar en significados alternativos: *off* y *on*; falso y verdadero; no y sí; cerrado y abierto. Sin embargo, puede resultar difícil imaginárselos. No se pueden ver ni tocar; además, si usted compra una computadora nadie le vende los bits para trabajar con ella. Estos razonamientos llevan a la conclusión de que los bits "viven" dentro de su computadora, y esa es la realidad. Internamente la computadora está compuesta por dispositivos electrónicos que actúan como conjuntos de llaves o interruptores, que permiten el paso de ciertos niveles de tensión. La "vida" de los bits depende de que la computadora esté conectada a una fuente de alimentación, de modo que un bit *1* se puede representar con un nivel de tensión. Por lo general se dice que "un *1* lógico" se corresponde con un nivel de tensión alto o *high* y que "un *0* lógico" se corresponde con un nivel de tensión bajo o *low*.



Nibble: es una agrupación de 4 bits o medio byte. Un byte está constituido por dos nibbles, el de orden superior y el de orden inferior.

Un **byte** es una combinación de *8* bits. Así como el bit es el grano de arena, la unidad más pequeña en la construcción de un dato para procesar en la computadora, el byte, es el ladrillo, el bloque de construcción real de datos.



Palabra de CPU: es la unidad de trabajo o procesamiento de CPU expresada en bits.

Cuando se hace referencia a medio byte, o *4* bits, se utiliza el término **nibble**. Es bastante común hacer referencia a esta medida; por ejemplo, en la representación de números empaquetados se habla de nibble menos significativo y nibble más significativo (el menos significativo de un byte corresponde al grupo de *4* bits de "extrema derecha" del conjunto de ocho bits).

Asimismo, la unidad elemental de representación de un dato es el byte, con el que, por ejemplo, se puede representar "la letra F" de "femenino". La unidad elemental de representación de información es el bit; por ejemplo, un bit "uno" puede indicar que el género de una persona "es verdad" que es "femenino" y "cero" "que es falso" que lo sea. En este caso el bit se considera el valor de verdad de un dato y, por lo tanto, "nos da información". Las computadoras necesitan procesar pequeños grupos de bytes (*2*, *4*, *8*). La **unidad de trabajo o procesamiento** de CPU se denomina **palabra de CPU** (tenga en cuenta que este término nada tiene que ver con el significado que estamos acostumbrados a darle a "palabra" en español). Su longitud tiene relación con la longitud de los registros asociados a la CPU y puede tener, o no, la misma cantidad de caracteres que el contenido de una posición de memoria (denominado también "palabra de memoria"). Considere estos registros como áreas de trabajo; por ejemplo, si el tamaño de los registros asociados a la CPU permite que se almacenen *8* bits, éste será su tamaño de palabra, entonces, cualquier operación para operandos de *16* bits se resuelve en dos pasos. Una computadora se puede programar para operar con datos de tamaño superior que la capacidad de sus registros, pero su tratamiento implica una mayor cantidad de pasos, lo que afecta la velocidad global de procesamiento. Por ejemplo, usted puede realizar el cálculo de dos números decimales de *6* dígitos en una calculadora "hipotética" con un visor de tres dígitos, pero resolverá el problema en dos pasos, lo que llevará más tiempo que hacerlo en una de *8* dígitos que lo resuelve en un solo paso.

1.9 Arquitectura de una unidad central de proceso (CPU)

La CPU es el módulo físico más importante. Su capacidad de trabajo determina la capacidad de trabajo de la computadora. En la figura 1.8 vemos tres bloques funcionales en los que se puede dividir para el ejemplo una CPU Intel.

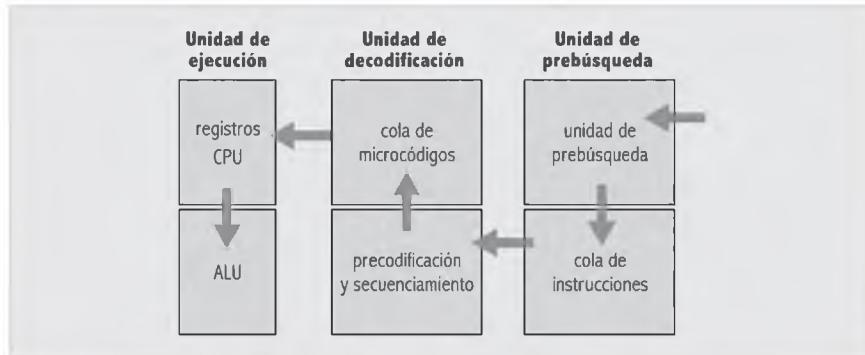


Fig. 1.8. Tres bloques funcionales de una CPU Intel.

La unidad de prebúsqueda obtiene instrucciones de la memoria principal por medio de una unidad que actúa de intermediario con el colector de datos e instrucciones o "bus" (no dibujada en el esquema). Estas instrucciones se disponen en una cola (de instrucciones).

Cada instrucción espera en la cola y es interpretada por la unidad de predecodificación, que la traduce a microcódigos. Se denomina **microcódigo** a un lenguaje de menor nivel que el código de máquina, que permite la representación de las microoperaciones organizadas de manera secuencial en el tiempo para la ejecución de la instrucción.

La ejecución se realiza en la unidad de ejecución con el soporte de la unidad aritmético-lógica y de los registros de trabajo de la CPU; además, la unidad de ejecución se comunica con los componentes que sean necesarios, aun cuando éstos no pertenezcan a la CPU.

El módulo de cálculo y comparación, denominado **unidad aritmético-lógica (ALU o Arithmetic Logic Unit)**, se encarga de operar los datos que recibe de la memoria y obtener el resultado. Las operaciones pueden ser aritméticas –suma, sustracción, desplazamiento, etc.– o lógicas –suma lógica (or), producto lógico (and), complemento (not)–.

Estas unidades funcionales forman parte de una entidad denominada **unidad central de proceso** o unidad de proceso, procesador central o simplemente procesador (**CPU** o *Central Processing Unit*).

En la CPU también están incluidos **registros** o *registers*, que forman una pequeña memoria local. Estos guardan información de manera transitoria para el procesamiento en curso (operandos, direcciones de memoria, etc.). Tienen un nombre acorde con la función que cumplen, aunque debe considerarse que en procesadores distintos asumen denominación propia y variada.

La CPU está asociada a la **memoria principal (PM o Principal Memory)** para obtener las instrucciones y los datos y almacenar los resultados; es de tecnología de semiconductores y también se denomina **memoria central, interna o memoria de trabajo**.

✓ **Unidad aritmético-lógica (ALU o Arithmetic Logic Unit):** se encarga de operar los datos que recibe de la memoria y obtener el resultado.

✓ **Memoria principal (PM o Principal Memory):** es de tecnología de semiconductores y también se denomina memoria central, interna o memoria de trabajo.

1.10 Lógica digital y componentes electrónicos

Otro enfoque en la descripción de un componente es el nivel de lógica digital. Este nivel permite la descripción con mayor detalle que el definido para un bloque funcional. La figura 1.9 muestra la diferencia de tres niveles: el de arquitectura, el de lógica digital (donde se muestra "ampliado" el diagrama de lógica para la suma de dos bits) y en un tercer nivel se explica cómo realizaría la operación el alumno.

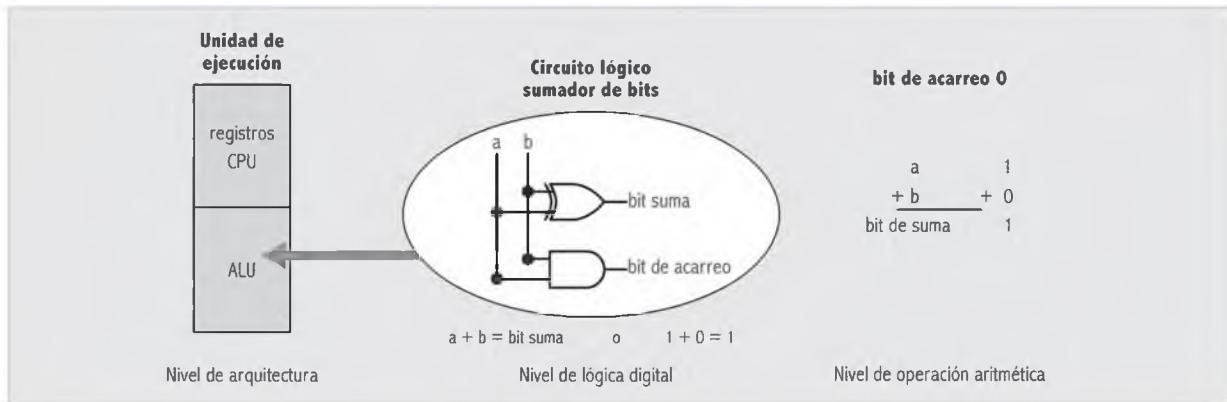


Fig. 1.9. Tres niveles de detalle.

Chip: es un circuito electrónico diseñado sobre una estructura de silicio.

En una ALU se pueden sumar muchos más de dos bits y, además, se pueden realizar muchas más operaciones. Por supuesto que describir una ALU con su complejidad actual a este nivel no es posible, por eso en el capítulo de Lógica Digital se tiene como objetivo que el lector conozca desde este enfoque los componentes más utilizados en cualquier computadora. Por ejemplo, se describe un decodificador de dos entradas y cuatro salidas para que se pueda comprender, en los capítulos posteriores, la **función** de decodificación de una dirección de n bits.

La representación del circuito lógico posibilita la comprensión del comportamiento y el estudio detallado de un componente.

Se puede incorporar mucha lógica digital en un chip. Un **chip** es un circuito electrónico diseñado sobre una estructura de silicio, material que se encuentra en la arena de las playas. Sobre la base de silicio se implementan cadenas de interruptores unidas por "pistas o conductores". En una superficie de $0,6$ cm de lado aproximadamente, se incluyen millones de componentes electrónicos. Es de bajo costo y con gran capacidad de lógica. Algunos fueron concebidos para almacenar grandes volúmenes de información y otros combinan lógica y registros para construir, por ejemplo, una CPU completa denominada desde entonces **microprocesador**. Un circuito integrado es un soporte físico que incluye en su interior conjuntos de compuertas lógicas.

En los primeros años del diseño de dispositivos electrónicos se utilizaban componentes discretos; se llama así al grupo de elementos constituidos por válvulas, resistencias y condensadores, por nombrar los más comunes.

Las válvulas pronto fueron reemplazadas por transistores y la evolución de técnicas de miniaturización permitió incluir un mayor número de transistores en una misma superficie.

Una compuerta lógica se puede armar con transistores conectados y su configuración permite satisfacer la tabla de verdad o tabla booleana a la que pertenece la compuerta. La evolución constante de la tecnología de fabricación de IC (o *Integrated Circuit*) permite agregar más y más compuertas en el mismo espacio. Según su complejidad, los IC se clasifican en:

- Integrados de pequeña escala (SSI o *Small Scale Integration*) a los circuitos, que incluyen hasta diez compuertas.
- Integrados de mediana escala (MSI o *Medium Scale Integration*) a los circuitos, que presentan menos de 100 compuertas.
- Integrados de gran escala (LSI o *Large Scale Integration*) a los circuitos, que tienen hasta 1.000 compuertas.
- Integrados de escala muy grande (VLSI o *Very Large Scale Integration*) a los circuitos, que superan las 1.000 compuertas.

Los IC vienen encapsulados en paquetes con un número de serie que informa las características del circuito que se incluyen en los catálogos dispuestos por el fabricante. El número de serie se distingue por una combinación numérica que designa básicamente el tipo de familia lógica y la función digital que realiza.

Cada bloque incluido en una cápsula de IC es un arreglo de transistores, diodos, resistencias y condensadores soldados a patas externas, o patillas o terminales, que permiten su conexión con el resto de los componentes del sistema. Por supuesto, la técnica de fabricación para lograr una mayor integración de componentes es distinta a la de menor integración y más costosa, lo que conduce a que las empresas que diseñan computadoras utilicen, en el caso de microprocesadores, algunos ya disponibles en el mercado; por ejemplo, un diseño actual de la empresa Hewlett Packard utilizó algunos microprocesadores de Intel. Las compañías que se dedican al desarrollo de microprocesadores (como Intel, AMD) los diseñan en grupos estándar para diversas aplicaciones.

La palabra de procesador es la unidad de trabajo o cantidad de información que puede procesar en un paso, expresada en bits; por ejemplo, cuántos bits como máximo pueden tener dos operandos para que se puedan sumar en forma simultánea. Las longitudes de procesador más comunes en el mercado actual son de 32 y 64 bits.

La capacidad de dirección se puede determinar por la cantidad de líneas que transfieren los bits de una dirección física, por ejemplo con 32 bits se pueden obtener 2^{32} combinaciones distintas que representan 2^{32} direcciones posibles. Este parámetro es importante, ya que determina la cantidad de memoria principal a la que puede acceder: cuanto mayor sea la cantidad de direcciones de memoria, mayor será su capacidad de almacenamiento y, por lo tanto, mayor su capacidad para almacenar programas y datos disponibles para su ejecución.

El número de instrucciones indica cuántas operaciones diferentes puede llevar a cabo una CPU; son ejemplos de operaciones los verbos sumar, restar, transferir, incrementar.

La filosofía de trabajo implementada en "máquinas CISC" otorga al programador de aplicaciones gran cantidad de instrucciones diferentes, que le permitan diseñar programas en los que la lógica compleja de un algoritmo se representa con unas pocas instrucciones de alta complejidad. La filosofía de trabajo implementada en "máquinas RISC" sostiene que la mayor parte de los programas utiliza 80% de instrucciones simples y sólo 20% de instrucciones complejas, razón por la que tienen un set de instrucciones constituido por pocas operaciones simples y rápidas. Por este motivo, programar un algoritmo con una lógica compleja requiere muchas instrucciones y, por esta razón, los programas son mucho más largos. En ambas se persigue alcanzar la mayor "velocidad de ejecución", en la primera con programas más cortos y en la segunda con instrucciones más rápidas. EPIC pretende combinar ambas,



La arquitectura de set de instrucciones se puede clasificar en tres grupos:

CISC (<i>Complex Instruction Set Computer</i>)	RISC (<i>Reduced Instruction Set Computer</i>)	EPIC (<i>Explicitly Parallel Instruction Computer</i>)
--	--	--

utilizando como estrategia "para ganar velocidad" la "ejecución de múltiples instrucciones en paralelo"

Las instrucciones operan sobre datos que están en memoria principal y que son llevados a los registros de CPU, que se utilizan para el almacenamiento temporal mientras se ejecuta una instrucción o varias de ellas. La denominación de cada uno, así como su cantidad y tamaño, dependen del procesador al que pertenezcan.

La **velocidad de ejecución** de un procesador depende en primer término de la velocidad del reloj del sistema. La ejecución de instrucciones se lleva a cabo al ritmo de señales que provienen del reloj, por lo tanto, la determinación de la velocidad está incluida en etapa de diseño de la computadora.

Denominaremos señal de reloj o *clock* a una señal que oscila a intervalos regulares entre 0 y 1. Se debe considerar que hay un tiempo de transición entre 0 y 1 o entre 1 y 0, debido a que el cambio de valor no es instantáneo. Se lo puede representar según el diagrama de la figura 1.10, en el que los grises marcan "las transiciones" mencionadas y las líneas punteadas gruesas, el intervalo que identifica un ciclo.

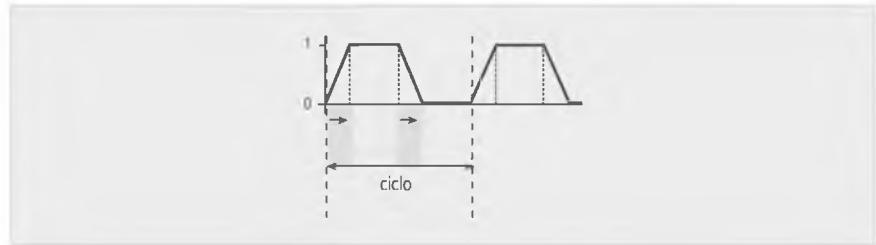


Fig. 1.10. Señal de reloj o *clock*.

La palabra **ciclo** se refiere a la señal representada en el diagrama, al instante en el que comienza la transición de "0 a 1", el tiempo que se mantiene en 1, el tiempo que transcurre hasta el final de la transición de "1" a "0", el tiempo que su valor es cero, hasta el comienzo de la nueva transición de "0" a "1".

El tiempo que transcurre en completarse un ciclo es un "período de reloj". Para generar la señal se utiliza un circuito oscilador que aplica una tensión a un cristal, por ejemplo, el cuarzo, que "vibra" a una frecuencia regular y que por ello puede utilizarse para generar señales que "oscilen" a intervalos regulares. Se denomina frecuencia a la cantidad de ciclos por segundo.

La frecuencia se expresa en Hz, con distintos multiplicadores (kilo, mega, giga, etc.), de acuerdo con la relación $T = 1/f$. Si la frecuencia es de 1 GHz = 1 000 000 000 de ciclos por segundo = 10^9 ciclos, entonces $1/10^9 = 10^{-9}$ seg = 1 nanosegundo, que es una unidad de tiempo muy utilizada en relación con la velocidad de determinadas operaciones.

El procesador y los demás componentes de una computadora operan con el suministro de corriente de una **fuente de alimentación**. Su función es convertir corriente alterna de alto voltaje (p. ej., 220 V) a corriente continua de bajo voltaje (p. ej., +5 V; -5V), además de estabilizar la tensión. La capacidad de la fuente se mide en watts, que es una medida de potencia.

El Hertz es la unidad de medida del Sistema Internacional de Unidades. Se aplica a la cantidad de veces que se repite una onda en un segundo, de modo que un Hertz quiere decir que se repite una vez.

Caso ejemplo: La placa madre de una computadora personal

Desde su aparición en el mercado, las microcomputadoras (sistemas basados en microprocesadores) incluyeron la mayor parte de sus componentes en una placa de circuito impreso, que se ha denominado de diversas maneras: placa madre o *mother board*, placa principal o placa de sistema. Esta placa sirve de soporte para la interconexión de los componentes básicos de cualquier sistema con microprocesador. Esta interconexión se realiza por medio de líneas "dibujadas" o "impresas" en la placa, que permiten la conducción de señales binarias. Estas señales "viajan" por las líneas mencionadas, que también se denominan **buses**, y que se clasifican según el tipo de agrupaciones de bits que transfieren en:

- Buses de control.
- Buses de datos.
- Buses de direcciones.

Los componentes fundamentales de una placa madre son:

- Zócalo de microprocesador o *socket*.
- Microprocesador (CPU)
- Reloj del sistema.
- Zócalo o *slot* para conectar la fuente de alimentación.
- Zócalos o *slots* para conectar los módulos de memoria, placas de memoria, etcétera
- Zócalos para buses de expansión (conexión a periféricos).

Los dispositivos que permiten la comunicación del microprocesador y las placas de memoria con el entorno de la computadora se denominan **periféricos** y para relacionar con ambos se utilizan **interfaces** (que involucran al microprocesador en la transferencia) o **unidades de acceso directo a memoria (DMA o Direct Memory Access)**, que transfieren datos entre la memoria y el periférico sin su intervención.

Los periféricos permiten el ingreso de datos (**periféricos de entrada**), la salida de resultados (**periféricos de salida**) y el almacenamiento a largo plazo (**periféricos de E/S o memorias externas**, por ejemplo, memorias soportadas en discos).

Desde el punto de vista del funcionamiento de los periféricos, los aspectos más importantes que se han de tener en cuenta son dos:

- Su conexión a la computadora.
- Los programas que definen su funcionamiento, llamados comúnmente manejadores o *drivers*.

El dispositivo periférico puede estar constituido por dos partes físicas:

- Una parte electromecánica (por ejemplo, conformada por los motores para el giro y el posicionamiento del brazo en un disco).
- El **controlador del periférico o unidad de control del periférico**, o simplemente **controlador**, que se encarga de la comunicación (de uno o varios periféricos de la misma clase) con un bus y está constituido (fig. 1.11) por:
 - Un *buffer* interno (físicamente una RAM), que permite el almacenamiento de la información que viaja desde o hacia el soporte.
 - Una lógica de control, que interpreta comandos y genera señales de control para la operación del periférico.

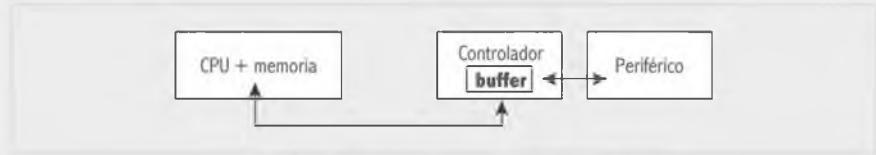


Fig. 1.11. Ubicación relativa del controlador.

Su función se puede resumir en estas operaciones básicas:

- Aislar el hardware del software que se ocupa de la transferencia y permitir un estándar de industria bien definido.
- Compatibilizar la velocidad de la transferencia de información entre los periféricos y la CPU.
- Convertir el formato de datos (estructura de datos) de la CPU al formato de datos de los periféricos y al revés.
- Amplificar las señales entre la CPU y los periféricos

Un sistema típico tiene entre sus componentes:

- Un adaptador de video.
- Un controlador de teclado
- Puertos.
- Controladores para discos rígidos y otros medios de almacenamiento masivo.

Controladores, interfaces, puertos y adaptadores son dispositivos físicos, que controlan la comunicación entre los periféricos y la computadora. Un controlador no siempre es una placa; en nuestra computadora el controlador del video es un pequeño CI integrado en la *mother board*.

Los periféricos son más lentos que la CPU, debido a que su operación involucra movimientos mecánicos, como el desplazamiento del papel en la impresora o el giro del disco alrededor de su eje. Un controlador acepta comandos propios del periférico que gobierna y establece su traducción a un "lenguaje operativo"; por ejemplo, un comando de inicialización de una transferencia en disco implica, como primera operación, el arranque del motor que permite el giro. De la misma manera, una operación de lectura implica, entre otras, una operación de selección y activación de sólo una de las cabezas de lectura de la unidad.

Por lo general, para establecer la comunicación con los periféricos se indica la dirección que identifica el periférico, denominada dirección de E/S o *port*.

Las direcciones de E/S son identificadores que se utilizan para direccionar a las placas asociadas a los periféricos (controlador, adaptador, etc.). Estas placas se insertan en la placa madre en forma directa (en los zócalos) o indirecta (en algún dispositivo que actúe de intermediario insertado, a su vez, en la placa madre). De esta forma, se establece un vínculo entre los pares CPU-memoria y controlador-periférico.

Sabemos que para leer o almacenar datos la CPU se comunica con la memoria a través de direcciones y, de la misma manera, puede establecer la comunicación con un periférico.



Puerto: es uno de los componentes de la comunicación, que permite que la CPU se enlace con un periférico.

Un puerto es uno de los componentes de la comunicación que permite que la CPU se enlace con un periférico; desde el punto de vista del software un *port* es una dirección del espacio de direccionamiento en memoria principal, que actúa de memoria temporal para el soporte de datos que van a enviarse o recibirse desde algunos tipos de periféricos.

Un puerto paralelo actúa básicamente como intermediario para el envío de una agrupación de bits en paralelo, esto quiere decir que todos los bits del grupo son enviados al mismo tiempo. Los puertos paralelos permiten la transferencia de datos en forma bidireccional.

Un puerto serie es un intermediario bidireccional, que permite el envío o la recepción de bits en serie, esto es, uno tras otro.

1.11 El Sistema Operativo. La Dinámica del Sistema

Asemeja el hardware de una computadora a los elementos que componen un juego de ajedrez. Para quien no conoce las **reglas** del juego, el tablero y las figuras representan objetos de poca utilidad. Así, el sistema operativo de la computadora indica las reglas que **coordinan** el trabajo de las piezas físicas, inútiles por sí solas.

Quien creó el juego de ajedrez relacionó de manera **coherente** figuras, tablero y reglas. Lo mismo ocurre con un sistema de computadora; el desarrollo físico se relaciona con el desarrollo de un conjunto de reglas de operación, que hacen productivo el trabajo de cada parte física. Estas reglas se describen en programas y constituyen el **sistema de operaciones o sistema operativo**. Para jugar ajedrez usted debe conocer las reglas del juego. Si su conocimiento es elemental, su juego será pobre; si su conocimiento es profundo, su juego será cada vez más interesante. El mayor conocimiento, por parte de un programador avanzado, acerca del software que utiliza su computadora, produce como resultado el mayor aprovechamiento de su herramienta de trabajo. Cuanto más conozca las posibilidades que le brinda el sistema operativo y la arquitectura de su computadora, mejor será la calidad de su trabajo.

Los programas del sistema operativo pueden estar almacenados en distintos soportes y esto tiene que ver con el tipo de servicio que ese programa presta. Así, las actividades críticas para el funcionamiento de la computadora se agrupan constituyendo un núcleo y permanecen en memoria principal la mayor parte del tiempo, de modo que intervienen con rapidez cuando es necesario durante el procesamiento de otro programa. Este núcleo toma distintos nombres según cada sistema operativo (p.ej., kernel para el sistema operativo Linux). El resto de los programas de servicio permanece en una memoria masiva (por ejemplo, un disco rígido) y son invocados por comandos del usuario, o por un proceso mediante "llamadas al sistema". Se puede afirmar que el software de base (sistema operativo y utilitarios) y de aplicación, o aplicaciones, se ejecutan alternativamente en la CPU y, de este modo, permiten hacer efectiva la tarea del procesamiento de los datos.

1.12 Resumen

El capítulo comienza con la especificación de qué se conoce como organización de computadoras y qué se entiende por arquitectura de computadoras. La relación entre organización y arquitectura es muy estrecha, ya que la organización del computador o conjunto de componentes y relaciones son parte del diseño de su arquitectura.

La función más importante de una computadora es procesar los datos. Las funciones básicas que puede llevar a cabo una computadora se pueden identificar por sus componentes:

el procesamiento de datos (CPU), la transferencia de datos y resultados (buses) y el almacenamiento de datos y resultados a corto (en memoria principal) y a largo plazo (en memorias secundarias). La unidad de control, como su nombre lo indica, controla el funcionamiento del hardware, ejecutando una instrucción en lenguaje de máquina. Una computadora transfiere los datos entre la memoria y el mundo exterior a través de periféricos de entrada y de periféricos de salida.

Se establece la clásica división en generaciones de computadoras, cuyo propósito es mostrar la incorporación de elementos de hardware y software, en función de la evolución tecnológica durante los últimos sesenta años, y de las necesidades que fueron surgiendo, primero en el ámbito científico y luego en el ámbito empresarial.

1.13 Contenido de la página Web de apoyo



El material marcado con asterisco (*) sólo está disponible para docentes.

Resumen gráfico del capítulo

Autoevaluación

Lecturas adicionales

Las comunicaciones. Conceptos básicos, de Antonio Castro Lechmaler y Rubén Fusario, es parte del libro "Telecomunicaciones para Ingenieros de sistemas" (de próxima aparición), de Alfaomega Grupo Editor (64 páginas). Agradecemos a sus autores por permitir que su escrito sea parte de las lecturas complementarias de esta obra.

Presentaciones *

2

Sistemas numéricos

Contenido

2.1 Introducción	26
2.2 Sistemas de notación posicional.....	26
2.3 Métodos de conversión de números enteros y fraccionarios	30
2.4 Operaciones fundamentales en binario.....	37
2.5 Operaciones fundamentales en octal y hexadecimal.....	39
2.6 Complemento de un número.....	42
2.7 Resumen	44
2.8 Ejercicios propuestos	46
2.9 Contenido de la página Web de apoyo.....	47

Objetivos

- Justificar la utilización de dígitos binarios como elementos del lenguaje interno de la computadora.
- Justificar el empleo de los sistemas hexadecimal y octal como medio para "empaquetar" entidades binarias.
- Incorporar las nociones de operaciones en sistema binario, suma hexadecimal, suma octal y complemento binario.

2.1 Introducción

En este capítulo abordaremos temas que nos muestran cómo se convierte un número binario a decimal, cómo se lo representa con menos símbolos en octal y en hexadecimal y los códigos más difundidos para la representación de información.

Se hará especial hincapié en el sistema binario, por ser el que utilizan las computadoras.

El sistema binario representa todos los símbolos del usuario –como letras y números– o los caracteres –como los signos de puntuación– o los comandos –como <ENTER>– con una secuencia de dígitos binarios llamados “bits” (BIT es el acrónimo de *binary digit* o dígito binario). Los bits de datos individuales pueden combinarse en agrupaciones de 8 bits llamadas “byte” (*1 byte = 8 bits*). No sólo los datos que se procesan están representados en binario, sino también las instrucciones de los programas. La computadora codifica los datos para realizar cálculos, ordenar alfabéticamente y demás transformaciones de información binaria. Por ejemplo, la letra “A” se codifica como “01000001” y la “B”, como “01000010” en el método de codificación más usado para representar caracteres alfanuméricos (tanto para almacenarlos como para mostrarlos en el monitor e imprimirlos). El valor binario correspondiente a la letra “A” es menor que el correspondiente a la “B”; esto lleva a la conclusión de que si queremos ordenar el par de letras nos basta con saber cuál es la menor para conocer cuál es la primera. Las computadoras digitales y muchos otros equipos electrónicos se basan en un sistema binario formado por dos únicos símbolos: 0 y 1. Desde que un usuario presiona las teclas de un teclado, el universo de su información se representará en binario. De este modo, debemos saber que dentro de la computadora se almacenan, transfieren y procesan sólo dígitos binarios. Desde el punto de vista físico, los bits pueden ser representados por una magnitud de voltaje o un campo magnético.

2.2 Sistemas de notación posicional

Los sistemas de notación posicional están formados por un juego de n cantidad de símbolos, cuya combinación representa valores diferentes. El concepto de criterio posicional es que cada dígito tiene un peso distinto según el lugar que ocupa; el **peso** es la base elevada a la posición que ocupa dentro del número. La suma de cada dígito multiplicado por su peso permitirá obtener el valor final del número.

En todo sistema posicional, dada una base B se tienen B dígitos posibles, de 0 a B-1.

2.2.1 Expresión generalizada de un número en potencias de su base

Dado un número N en base B, expresado por n dígitos (d), éste será igual a la sumatoria de cada dígito (d) por la potencia de la base B que le corresponda a su posición.

$$N = d_n \cdot B^n + d_{n-1} \cdot B^{n-1} + \dots + d_1 \cdot B^1 + d_0 \cdot B^0$$

$$N = d_n \cdot d_{n-1} \dots d_1 \cdot d_0 = d_n \cdot 10^n + d_{n-1} \cdot 10^{n-1} + \dots + d_1 \cdot 10^1 + d_0 \cdot 10^0$$

Más adelante podremos comprobar que la base (B) se representa con los dígitos 10 (1, 0) para cualquier sistema.

2.2.2 Sistema decimal

El **sistema de numeración decimal** es un sistema de notación posicional formado por 10 símbolos (0, 1, 2, 3, 4, 5, 6, 7, 8, 9). Entonces, el número decimal 1 023 estará conformado por la suma de las siguientes cantidades:

$$\begin{array}{r}
 1 \ 0 \ 2 \ 3_{(10)} \\
 | \quad | \quad | \quad | \\
 3 \cdot 10^0 = \quad 3 \\
 2 \cdot 10^1 = \quad 20 \\
 0 \cdot 10^2 = + \quad 0 \\
 1 \cdot 10^3 = \underline{\quad 1.000} \\
 \\
 1.023_{(10)}
 \end{array}$$

2.2.3 Sistema binario

El **sistema binario**, o de base 2, también es un sistema de notación posicional, formado por dos símbolos (0, 1) a los que se denomina **bits**. El sistema binario permite analogar sus dos símbolos (0 y 1) con dos "estados" posibles; por lo tanto, los valores 0 y 1 se pueden interpretar, por ejemplo, como:

<i>OFF</i>	<i>ON</i>
Falso	Verdadero
No	Sí

Un número binario está compuesto por un conjunto de bits y su equivalente decimal será igual a la suma que resulte de multiplicar cada bit por las potencias de 2 (la base B) que correspondan a su posición. Así, el número binario:

$$\begin{array}{r}
 1 \ 1 \ 0 \ 1 \ 1_{(2)} \\
 | \quad | \quad | \quad | \quad | \\
 1 \cdot 2^0 = \quad 1 \\
 1 \cdot 2^1 = \quad 2 \\
 0 \cdot 2^2 = + \quad 0 \\
 1 \cdot 2^3 = \quad 8 \\
 1 \cdot 2^4 = \underline{\quad 16} \\
 \\
 27_{(10)}
 \end{array}$$

El número $11011_{(2)}$ se lee: uno, uno, cero, uno, uno, en base 2.

Se debe dejar en claro que, por ejemplo, el número 1020 no pertenece al sistema binario, ya que el dígito 2 no pertenece al conjunto de símbolos binarios.

2.2.4 Sistema octal

El **sistema octal**, o de base 8, es un sistema posicional formado por ocho símbolos (0, 1, 2, 3, 4, 5, 6, 7); el peso de cada cifra son las potencias sucesivas de 8. De esta manera:

$$\begin{array}{r}
 2 \ 7 \ 5_{(8)} \\
 | \quad | \quad | \\
 5 \cdot 8^0 = \quad 5 \\
 7 \cdot 8^1 = + \ 56 \\
 2 \cdot 8^2 = \underline{\quad 128} \\
 \\
 189_{(10)}
 \end{array}$$

Sistema binario o de base 2: es un sistema de notación posicional, formado por dos símbolos (0,1) a los que se denominan bits.

Sistema octal o de base 8: es un sistema posicional formado por ocho símbolos (0, 1, 2, 3, 4, 5, 6, 7); el peso de cada cifra son las potencias sucesivas de 8.

El número $275_{(8)}$ se lee: dos, siete, cinco, en base 8

El número 948 no pertenece al sistema octal, porque los dígitos 9 y 8 no pertenecen a su conjunto de símbolos.

2.2.5 Sistema hexadecimal

El **sistema hexadecimal**, o de base 16, en general abreviado con las siglas "Hexa", "H" o "h", es un sistema posicional formado por dieciséis símbolos (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F) que representan los valores decimales de 0 a 15; el peso de cada cifra son las potencias sucesivas de 16.

Luego, si queremos hallar el equivalente decimal del número hexadecimal 1A2, lo haremos de la siguiente forma:

$$\begin{array}{r}
 1 \ A \ 2_{(16)} \\
 | \quad | \quad | \\
 2 \cdot 16^0 = 2 \\
 + 10 \cdot 16^1 = + 160 \\
 \hline
 1 \cdot 16^2 = 256 \\
 \hline
 418_{(10)}
 \end{array}$$

El número $1A2_{(16)}$ se lee: uno, a, dos, en base 16.

Es importante recalcar que un número hexadecimal suele identificarse con la letra "h" luego del dígito menos significativo, por ejemplo 1A2h. En ese caso, "h" no es el dígito menos significativo del número, sino que cumple la función de identificar un sistema de numeración. De cualquier manera, "h" no podría nunca expresar un dígito hexadecimal, ya que no pertenece al conjunto de símbolos del sistema.

En la tabla 2-1 se presentan las equivalencias entre los sistemas decimal, binario, octal y hexadecimal:

Tabla 2-1. Equivalencias entre los sistemas de bases 10, 2, 8 y 16.

Decimal	Binario	Octal	Hexadecimal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

El 10 (uno, cero) es el número que en cualquier sistema representa la base, por ejemplo en base 5

$$10_{(5)} = 5_{(10)}$$

luego $100_{(5)} = 5^2_{(10)}$ o sea, la base al cuadrado.

$$\begin{array}{r}
 1 \quad 0 \quad 0_{(5)} \\
 | \qquad \qquad \qquad \qquad | \\
 \text{---} \quad 0 \cdot 5^0 = 0 \\
 | \qquad \qquad \qquad \qquad | \\
 \text{---} \quad 0 \cdot 5^1 = 0 \\
 | \qquad \qquad \qquad \qquad | \\
 \text{---} \quad 1 \cdot 5^2 = \underline{\underline{25}} \\
 \\
 25_{(10)} = \underline{\underline{F}}_{(5)}
 \end{array}$$

Como se puede observar en la tabla 1-1, cada sistema de notación posicional construye sus cantidades de la siguiente manera:

- Primero, utiliza los números que le pertenecen, en orden creciente, desde el primero (el cero) hasta agotar el último.
 - Segundo, repite el paso anterior, pero incrementando el dígito de orden superior en una unidad.
 - Tercero, repite los dos primeros pasos, tantas veces como sea necesario, para representar el número deseado.

Para analizar los contenidos de los registros internos de la computadora, se utilizan los sistemas octal y hexadecimal como "métodos taquigráficos", que permiten representar su contenido binario compactado. Esto es posible debido a que **8 y 16 son potencias exactas de 2** ($8 = 2^3$ y $16 = 2^4$); en consecuencia, es factible convertir números del sistema binario al octal y al hexadecimal tomando agrupaciones de 3 y 4 bits, respectivamente. Por ejemplo, el número binario $11110010_{(2)} = 362_{(10)}$ y $F2_{(16)}$.

Para lograr esta conversión, la cadena binaria *11110010* se divide en grupos de 3 bits, de derecha a izquierda y, si fueran necesarios para completar el grupo, se agregan ceros a la izquierda del último bit:

011 **110** **010** y luego se le asigna el dígito octal equivalente

De la misma manera, la cadena binaria 11110010 se divide en grupos de 4 bits:

1111 **0010** y luego se le asigna el dígito hexadecimal equivalente
F **2**
 (16)

Queda claro que los sistemas octal y hexadecimal permiten “compactar” bits, de modo de hacer más sencilla la tarea de reconocerlos a simple vista. Sólo se debe asumir el esfuerzo de aprender cómo se convierte directamente desde binario a base 8 o 16, y al revés. Por esta razón, cuando un usuario pretende visualizar la información “desde adentro”, la computadora asume uno de estos dos sistemas y simplifica la tarea de leer ceros y unos. Si se comparan ambos métodos, se nota que el sistema hexadecimal compacta mejor la cadena binaria que el octal; esto es, indica lo mismo con menor cantidad de símbolos. Por lo tanto, se puede inferir que cuanto más grande es la base, más importante es la reducción. Cabe preguntarse, entonces, por qué no utilizar un sistema base 32 o 64; la razón es simple, resultaría engoroso recordar 32 o 64 símbolos diferentes y se estaría ante un problema casi mayor que el de operar con bits.

2.2.6 Número de cifras. Cantidad decimal máxima

Repase de nuevo la tabla de equivalencias. Note que con **un** bit se pueden representar los dígitos decimales 0 y 1. Con **dos** bits se pueden representar los dígitos decimales 0, 1, 2 y 3. Con **tres** bits, los dígitos decimales 0 a 7, con **cuatro** bits, los dígitos decimales 0 a 15.

Los sistemas octal y hexadecimal permiten "compactar" bits, de modo de hacer más sencilla la tarea de reconocerlos a simple vista.

Si se llama n a la cantidad de bits en juego, se puede generalizar el concepto:

con n bits se pueden representar los valores decimales comprendidos entre 0 y $2^n - 1$.

Nótese que de la misma tabla se deduce que:

con n dígitos octales se pueden representar los valores decimales comprendidos entre 0 y $8^n - 1$.

De la misma manera,

con n dígitos hexadecimales se pueden representar los valores decimales comprendidos entre 0 y $16^n - 1$.

2.3 Métodos de conversión de números enteros y fraccionarios

Ahora, es preciso establecer qué relación hay entre los distintos sistemas numéricos estudiados y el sistema decimal que estamos acostumbrados a utilizar. Despues de cierto tiempo de práctica, puede resultar bastante simple reconocer directamente que $15_{(10)}$ es igual a $1111_{(2)}$ y que ambas entidades están identificando 15 elementos; sin embargo, es factible que surjan problemas cuando la cantidad de dígitos de un sistema no decimal es muy grande, por ejemplo, $111110001010100101_{(2)}$. Para lograrlo se pueden utilizar los siguientes métodos que permiten la conversión entre sistemas de bases distintas para números enteros y fraccionarios.

2.3.1 Método de conversión de números de otras bases a decimal

A continuación, se describen los métodos que permiten convertir números de cualquier sistema de notación posicional al sistema decimal.

2.3.1.1. Binario a decimal (2) a (10)

Enteros

Sea el número $10111_{(2)}$, se obtendrá su valor decimal multiplicando cada bit por la potencia de dos que corresponda a su posición y sumando los valores finales.

$$\begin{array}{r}
 2^4 \ 2^3 \ 2^2 \ 2^1 \ 2^0 \\
 1 \ 0 \ 1 \ 1 \ 1_{(2)} \\
 \hline
 & 1 \cdot 2^0 = 1 \\
 & 1 \cdot 2^1 = 2 \\
 & 1 \cdot 2^2 = + 4 \\
 & 0 \cdot 2^3 = 0 \\
 \hline
 & 1 \cdot 2^4 = 16 \\
 \hline
 & 23_{(10)}
 \end{array}$$

Fracciones

Sea el número $0,101_{(2)}$, se obtendrá su valor decimal multiplicando cada bit por la potencia negativa de dos que corresponde a su posición, a partir de la coma y sumando los valores finales.

$$\begin{array}{ccccccc}
 & 2^0 & 2^{-1} & 2^{-2} & 2^{-3} \\
 \boxed{0,} & 1 & 0 & 1_{(2)} & & & \\
 & & & \downarrow & & & \\
 & & & 1 \cdot 2^{-3} = & 0,125 & & \\
 & & & \downarrow & & & \\
 & & & 0 \cdot 2^{-2} = +0 & & & \\
 & & & \downarrow & & & \\
 & & & 1 \cdot 2^{-1} = & 0,5 & & \\
 & & & & & & \\
 & & & & & & 0,625_{(10)}
 \end{array}$$

Como la conversión de binario a decimal será con la que usted se enfrentará más a menudo, por lo menos en este texto, es conveniente que recuerde el equivalente decimal de las primeras potencias negativas de dos.

Para recordar:

$$\begin{aligned}
 2^{-1} &= 0,5 \\
 2^{-2} &= 0,25 \\
 2^{-3} &= 0,125 \\
 2^{-4} &= 0,0625 \\
 2^{-5} &= 0,03125
 \end{aligned}$$

2.3.1.2. Octal a decimal (8) a (10)

Sea el número $17,3_{(8)}$ se obtendrá su valor decimal multiplicando cada dígito octal por la potencia de ocho que corresponda a su posición y sumando los valores finales.

$$\begin{array}{ccccccc}
 & 1 & 7 & , & 3_{(8)} & & \\
 & & & | & & & \\
 & & & 3 \cdot 8^{-1} = 3 \cdot \frac{1}{8} = 0,375 & & & \\
 & & & \downarrow & & & \\
 & & & 7 \cdot 8^0 = 7 \cdot 1 = +7 & & & \\
 & & & \downarrow & & & \\
 & & & 1 \cdot 8^1 = 1 \cdot 8 = & 8 & & \\
 & & & & & & \\
 & & & & & & 15,375_{(10)}
 \end{array}$$

2.3.1.3. Hexadecimal a decimal (16) a (10)

Sea el número $1A,0F_{(16)}$ se obtendrá su valor decimal multiplicando cada dígito hexadecimal por la potencia de dieciséis que corresponda a su posición y sumando los valores finales.

$$\begin{array}{ccccccc}
 & 1 & A & , & 0 & F_{(16)} & \\
 & & & | & & & \\
 & & & 15 \cdot 16^{-2} = 15 \cdot \frac{1}{256} = 0,058... & & & \\
 & & & \downarrow & & & \\
 & & & 0 \cdot 16^{-1} = 0 = +0 & & & \\
 & & & \downarrow & & & \\
 & & & 10 \cdot 16^0 = 10 \cdot 1 = 10 & & & \\
 & & & \downarrow & & & \\
 & & & 1 \cdot 16^1 = 1 \cdot 16 = & 16 & & \\
 & & & & & & \\
 & & & & & & 26,058..._{(10)}
 \end{array}$$

2.3.2 Método de divisiones sucesivas (para convertir un número entero decimal a otras bases)

Para convertir un número entero decimal a cualquier sistema, se divide el número por la base que corresponda hasta hallar el último cociente (o hasta que el último cociente sea cero), que formará con todos los restos anteriores (desde el último hasta el primero) el número buscado.

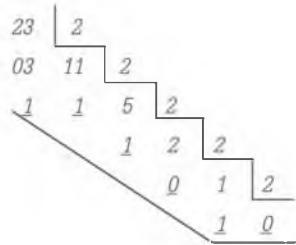


Para convertir un número entero decimal a cualquier sistema, se divide el número por la base que corresponda hasta hallar el último cociente .

El seguimiento de los ejemplos siguientes permitirá eliminar dudas.

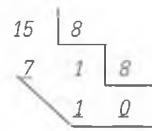
2.3.2.1. Decimal a binario (10) a (2)

$$23_{(10)} \text{ a } (2) = 10111_{(2)}$$



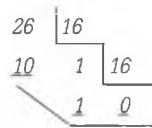
2.3.2.2. Decimal a octal (10) a (8)

$$15_{(10)} \text{ a } (8) = 17_{(8)}$$



2.3.2.3. Decimal a hexadecimal (10) a (16)

$$26_{(10)} \text{ a } (16) = 1A_{(16)}$$



En este caso, el primer resto es "10" y debe ser reemplazado por su equivalente hexadecimal "A".

2.3.3 Método de multiplicaciones (para convertir un número fraccionario decimal a otras bases)

Para convertir un número fraccionario decimal a cualquier sistema, primero se debe recordar que de una fracción pura se obtiene, después de la conversión, otra fracción pura. Por eso, cuando un número tiene parte entera y parte fraccionaria se lo separa y se aplica un método para la conversión de la parte entera y otro para la parte fraccionaria, como se describe a continuación.

Algoritmo para la conversión de un número fraccionario decimal a otras bases

1. Multiplicar la fracción por la base de conversión (en el caso de que la conversión se realice a sistema binario, la base es 2), cuyo resultado tiene una parte entera, que se considerará uno de los dígitos fraccionarios buscados.
2. Del resultado se toma sólo la parte fraccionaria y se vuelve a multiplicar por la base, para hallar el siguiente dígito fraccionario buscado. Este procedimiento se realiza tantas veces como sea necesario, hasta cancelar el método por alguno de los motivos que siguen:
 - La parte fraccionaria en una iteración se hace cero.

- En la conversión comienza a repetirse una secuencia de dígitos fraccionarios, lo que implica que es una fracción periódica.
- Se hallaron varios dígitos y se considera que la fracción obtenida tiende al resultado esperado por la conversión, y el error es infinitesimal

En consecuencia, para todos los casos, el nuevo número fraccionario estará compuesto por la parte entera resultante de cada multiplicación; éstas se toman en orden descendente, o sea, desde la primera hasta la última iteración. La flecha (\downarrow) indica el orden en que se deberán tomar los dígitos. Veamos los ejemplos.

2.3.3.1. Decimal a binario (10) a (2)

$$\text{1er. caso)} \quad 0,625_{(10)} \text{ a } (2) = 0,101_{(2)} \quad \begin{array}{l} 0,625 \cdot 2 = 1,250 \\ 0,25 \cdot 2 = 0,5 \\ 0,5 \cdot 2 = 1 \end{array} \quad \downarrow$$

Deben tomarse en cuenta las consideraciones siguientes:

- Cada parte entera así obtenida no debe superar el valor de la base del sistema; si esto se produce, hay un error en la multiplicación.
- Cada parte entera, formada por dos dígitos decimales, debe representarse con un solo símbolo del sistema al que se convierte. Así, el pasaje de decimal a hexadecimal de $0,625 \cdot 16 = 10,0$ y la cifra 10 es reemplazada por $A_{(16)}$

$$\text{2do. caso)} \quad 0,2_{(10)} \text{ a } (2) = 0,\overbrace{0011}_{(2)} \quad \begin{array}{l} 0,2 \cdot 2 = 0,4 \\ 0,4 \cdot 2 = 0,8 \\ 0,8 \cdot 2 = 1,6 \\ 0,6 \cdot 2 = 1,2 \\ 0,2 \cdot 2 = 0,4 \end{array} \quad \begin{array}{l} | \\ | \\ | \\ | \\ | \end{array} \quad \text{período}$$

El arco en el resultado señala el conjunto de números que se repiten, o sea, la periodicidad

$$\text{3er. caso)} \quad 0,122_{(10)} \text{ a } (2) = 0,000111\ldots_{(2)} \quad \begin{array}{l} 0,12 \cdot 2 = 0,24 \\ 0,24 \cdot 2 = 0,48 \\ 0,48 \cdot 2 = 0,96 \\ 0,96 \cdot 2 = 1,92 \\ 0,92 \cdot 2 = 1,84 \\ 0,84 \cdot 2 = 1,68 \\ \dots \end{array}$$

El método se cancela cuando la parte fraccionaria resultante de una multiplicación da 0, cuando se obtiene periodicidad o cuando se suspende luego de varias iteraciones, de acuerdo con la aproximación de la conversión que se crea conveniente. En este caso, se debe calcular el error cometido y considerar si éste es tolerable o afecta demasiado el resultado del cálculo en el que interviene.

2.3.3.2. Decimal a octal (10) a (8)

$$1er. \ caso) \ 0,625_{(10) \ a \ (8)} = 0,5(8)$$

$$0,625 \cdot 8 = 5,0$$

$$2do. \ caso) \ 0,2_{(10) \ a \ (8)} = 0,\overline{1463}_{(8)}$$

$$\begin{array}{r|l} 0,2 \cdot 8 = 1,6 & \\ 0,6 \cdot 8 = 4,8 & \text{periodo} \\ 0,8 \cdot 8 = 6,4 & \\ 0,4 \cdot 8 = 3,2 & \\ 0,2 \cdot 8 = 1,6 & \end{array}$$

$$3er. \ caso) \ 0,1_{(10) \ a \ (8)} = 0,\overline{06314}_{(8)}$$

$$\begin{array}{r|l} 0,1 \cdot 8 = 0,8 & \\ 0,8 \cdot 8 = 6,4 & \\ 0,4 \cdot 8 = 3,2 & \text{periodo} \\ 0,2 \cdot 8 = 1,6 & \\ 0,6 \cdot 8 = 4,8 & \\ 0,8 \cdot 8 = 6,4 & \end{array}$$

$$4to. \ caso) \ 0,12_{(10) \ a \ (8)} = 0,07534..._{(8)}$$

$$\begin{array}{r|l} 0,12 \cdot 8 = 0,96 & \\ 0,96 \cdot 8 = 7,68 & \\ 0,68 \cdot 8 = 5,44 & \\ 0,44 \cdot 8 = 3,52 & \\ 0,52 \cdot 8 = 4,16 & \end{array}$$

Se suspende con una aproximación conveniente.

2.3.3.3. Decimal a hexadecimal (10) a (16)

$$1er. \ caso) \ 0,25_{(10) \ a \ (16)} = 0,A_{(16)}$$

$$0,625 \cdot 16 = 10,0 = 0,A_{(16)}$$

El supuesto "resto" es 10 y debe ser reemplazado por el símbolo hexadecimal A.

$$2do. \ caso) \ 0,2_{(10) \ a \ (16)} = 0,\overline{3}_{(16)} \quad \begin{array}{r|l} 0,2 \cdot 16 = 3,2 & \text{periodo} \\ 0,2 \cdot 16 = 3,2 & \end{array}$$

$$3er. \ caso) \ 0,12_{(10) \ a \ (16)} = 0,1EB85..._{(16)}$$

$$\begin{array}{r|l} 0,12 \cdot 16 = 1,92 & \\ 0,92 \cdot 16 = E,72 & \\ 0,72 \cdot 16 = B,52 & \\ 0,52 \cdot 16 = 8,32 & \\ 0,32 \cdot 16 = 5,12 & \\ \dots & \end{array}$$

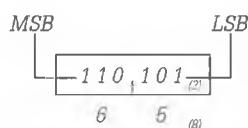
Se suspende con una aproximación conveniente.

2.3.4 Pasaje directo entre las bases 2 a 8 y 2 a 16

El pasaje directo entre los sistemas binario y octal, y entre binario y hexadecimal, se puede realizar porque 8 y 16 son potencias exactas de 2.

2.3.4.1. Binario a octal (2) a (8)

Si se observa la tabla de equivalencias entre sistemas, se ve que todo símbolo octal se representa, a lo sumo, con una combinación de 3 bits. Así, el número $110101_{(2)}$ podría representarse en octal como:



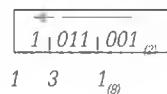
Llamaremos **LSB** (*Least Significant Bit* o de menor peso) al primer bit de la derecha y **MSB** (*Most Significant Bit* o de mayor peso) al primer bit de la izquierda del número binario. Luego, el procedimiento para convertir un número binario a octal en forma "directa" es el que se describe a continuación.

Conversión directa de binario a octal

1. Se divide el número binario en grupos de 3 bits, en el sentido siguiente: para enteros desde el LSB hacia el MSB y para fracciones en sentido inverso.
2. Luego se sustituye cada grupo de 3 bits por el correspondiente dígito octal, con cuidado de completar con ceros a la derecha la parte fraccionaria, de ser necesario para obtener el grupo de 3.

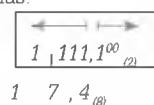
Enteros

Sea el número $1011001_{(2)}$, tomado como ejemplo para la conversión de binario a octal, el seguimiento del procedimiento da como resultado:



Fracciones

En primer término se deben formar grupos de 3 dígitos binarios, teniendo presente que en la parte fraccionaria cambia el sentido de la agrupación, que se encuentra representado por la orientación de las flechas.



El motivo de formar grupos de 3 dígitos binarios se debe a que la base es potencia exacta de dos, $8 = 2^3$, luego el exponente indica el tamaño de la agrupación.

Los dos ceros de la fracción se agregan para completar el grupo y se señalaron como superíndices para marcar el procedimiento. El agregado de ceros para completar el grupo es imprescindible para lograr la conversión correcta. Obsérvese que $0,1$ en sistema binario ($1/2$) no equivale a $0,1$ en sistema octal ($1/8$), y es muy diferente aun al resultado obtenido si se rellena el grupo con ceros a la derecha, como lo indica el procedimiento, que permite un resultado final de $0,4$ en octal.

2.3.4.2 Octal a binario (8) a (2)

En forma inversa a la enunciada antes, se puede convertir directamente un número octal a su correspondiente en base 2, reemplazando cada dígito octal por el equivalente binario en grupos de 3 bits.

Enteros

Así, si se toma el ejemplo anterior, $65_{(8)}$ se convertirá directamente a binario reemplazando cada dígito octal por el grupo de tres dígitos binarios que le correspondan.

$$\begin{array}{r} 6 \quad 5_{(8)} \\ \hline 110 \quad 101_{(2)} \end{array}$$

Fracciones

$$\begin{array}{r} \frac{1}{2} \quad \frac{4}{1} \quad \frac{2}{1} \\ \hline 10 \quad 100, \quad 010_{(2)} \end{array}$$

El sentido de las agrupaciones para números fraccionarios es siempre el mismo.

2.3.4.3 Binario a hexadecimal (2) a (16)

La conversión en forma directa de un binario a un hexadecimal sigue los mismos pasos enunciados para la conversión a un octal.

Enteros

Nótese que cada símbolo hexadecimal se representa, a lo sumo, con una combinación de 4 bits. Si se toma el número $110101_{(2)}$ y se lo divide en grupos de 4 bits empezando por el LSB, al sustituir cada grupo por su dígito hexadecimal correspondiente, se obtendrá:

$$\begin{array}{r} \leftarrow \quad \rightarrow \\ \begin{array}{|c|c|} \hline 0 & 1 & 1 & | & 0 & 1 & 0 & 1_{(2)} \\ \hline 3 & & 5 & & & & & 16 \end{array} \end{array}$$

El criterio de formar grupos de 4 bits es el mismo que se tomó para la conversión a octal, teniendo en cuenta que en hexadecimal la base es potencia exacta de $2, 16 = 2^4$, y el exponente 4 es el que indica la cantidad de dígitos binarios que se deben agrupar para esta conversión.

Fracciones

$$\begin{array}{r} \leftarrow \quad \rightarrow \\ \begin{array}{|c|c|} \hline 1 & 1 & 1 & , & 1 & 0 & 0 & 0_{(2)} \\ \hline F & , & 8 & & & & & 16 \end{array} \end{array}$$

2.3.4.4 Hexadecimal a binario (16) a (2)

De manera inversa, se podrá convertir directamente un hexadecimal a binario, reemplazando cada dígito hexadecimal por su correspondiente binario en grupos de 4 bits.

Entero

Sea el número $35_{(16)}$ la conversión directa a binario dará como resultado:

$$\begin{array}{r} \leftarrow \quad \rightarrow \\ \begin{array}{|c|c|} \hline 3 & 5_{(16)} \\ \hline 0011 \quad 0101_{(2)} \end{array} \end{array}$$

Fracciones

Si se desea convertir una fracción hexadecimal a binario en forma directa, esto se realizará de manera semejante a la conversión octal-binaria, sólo que los grupos se deberán armar de 4 dígitos binarios.

$$\begin{array}{r} F. \quad 8_{(16)} \\ 1111,1000_{(2)} \end{array}$$

De idéntica forma se pueden realizar pasajes directos entre distintas bases, donde una sea potencia exacta de la otra.

2.4 Operaciones fundamentales en binario

2.4.1 Suma

La suma entre 2 bits tiene cuatro combinaciones de operación posibles:

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 0 \text{ y me llevo 1; o acarreo 1}$$

Observemos algunos ejemplos para entender el mecanismo de la suma.

Si quiere sumar 1 más 1 en binario, el resultado será 10, o sea, la base binaria (ya que tiene sólo dos elementos). Expresado de otra manera, el resultado es 0 y me llevo 1 (o el acarreo es 1). Si ahora, para comprobar que el resultado es el correcto, se halla el equivalente de 10₍₂₎ en decimal, se obtiene 2₍₁₀₎, que es el mismo resultado que surge de sumar 1 más 1 en decimal.



Encuentre un simulador que permite realizar conversiones y operaciones entre sistemas numéricos en la Web de apoyo.

$$\begin{array}{r}
 & 1_{(2)} \\
 & + 1_{(2)} \\
 \hline
 & 10_{(2)} \\
 \text{acarreo} &
 \end{array}
 \quad
 \begin{array}{r}
 & 1_{(10)} \\
 & + 1_{(10)} \\
 \hline
 & 2_{(10)} \\
 \text{comprobación} &
 \end{array}
 \quad
 \begin{array}{r}
 10_{(2)} \\
 \hline
 0 \cdot 2^0 = 0 \cdot 1 = 0_{(10)} \\
 1 \cdot 2^1 = 1 \cdot 2 = +2_{(10)} \\
 \hline
 2_{(10)}
 \end{array}$$

$$\begin{array}{r}
 & 1_{(2)} \\
 & + 11_{(2)} \\
 \hline
 & 100_{(2)} \\
 \text{comprobación} &
 \end{array}
 \quad
 \begin{array}{r}
 & 1_{(10)} \\
 & + 3_{(10)} \\
 \hline
 & 4_{(10)} \\
 \text{comprobación} &
 \end{array}
 \quad
 \begin{array}{r}
 100_{(2)} \\
 \hline
 0 \cdot 2^0 = 0 \cdot 1 = 0_{(10)} \\
 0 \cdot 2^1 = 0 \cdot 2 = +0_{(10)} \\
 1 \cdot 2^2 = 1 \cdot 4 = 4_{(10)} \\
 \hline
 4_{(10)}
 \end{array}$$

En el caso en que se desee sumar 1 más 11 en binario, el resultado será 100₍₂₎.

2.4.2 Resta o sustracción

Es común que la operación de resta en una computadora se realice con una suma, esto es, se obtenga una resta al sumar el minuendo y el complemento a la base del sustraendo. Sin embargo, a efectos de explicar la operación manual, y teniendo en cuenta que será de gran utilidad para el lector (por ejemplo, en el seguimiento de la ejecución de un programa cuyos resultados no son los esperados), se explicarán otros mecanismos que se pueden aplicar para la resta de dos números binarios:

- Uno de ellos tiene en cuenta las combinaciones siguientes para la resta de cada par de dígitos:



La operación de resta en una computadora se calcula mediante la suma del minuendo y el complemento a la base del sustraendo.

$$0 - 0 = 0$$

$0 - 1 = 1$ le pide 1 a la posición siguiente (produce un acarreo)

$$1 - 0 = 1$$

$$1 - 1 = 0.$$

Supongamos que se debe efectuar la resta $101_{(2)} - 11_{(2)}$

Minuendo

$$\begin{array}{r} 101_{(2)} \\ \underline{- 11_{(2)}} \end{array}$$

Sustraendo

$$\begin{array}{r} 101_{(2)} \\ - 11_{(2)} \\ \hline \text{Resultado o resto} \end{array}$$

$$010_{(2)}$$

010

$$0 \cdot 2^0 = 0_{(10)}$$

$$1 \cdot 2^1 = + 2_{(10)}$$

$$0 \cdot 2^2 = 0_{(10)}$$

Comprobación $2_{(10)}$

Si se aplica este mecanismo a cada par de dígitos, en la primera columna el resultado de restar 1 menos 1 es 0. En la segunda columna, 0 menos 1 es 1, y proviene de haberle "pedido 1" a la posición siguiente (indicado con un 1 en la tercera columna). Por último, en la tercera columna, 1 menos 1 es 0. De la misma forma que se hizo para la suma, se puede comprobar que el resultado es el correcto, hallando el equivalente de $010_{(2)}$ que es 2 en decimal.

- Otro mecanismo que se utiliza, si no se puede memorizar el anterior, consiste en pre-guntar para cada par de dígitos: ¿cuál será el dígito que sumado al sustraendo me da el minuendo?, o ¿cuánto le falta al sustraendo para llegar al minuendo? Tratemos de utilizarlo, realizando las restas que se presentan a continuación.

1er. caso)

$$\begin{array}{r} 1001_{(2)} \\ \underline{- 110_{(2)}} \end{array}$$

$$0011_{(2)}$$

$$\begin{array}{r} 0011_{(2)} \\ \underline{- 120_{(2)}} \end{array}$$

$$\begin{array}{r} 0011_{(2)} \\ \underline{- 121_{(2)}} \end{array}$$

$$0 \cdot 22 = 0_{(10)}$$

$$0 \cdot 23 = 0_{(10)}$$

Comprobación $3_{(10)}$

Para realizar la resta $1001_{(2)} - 110_{(2)}$, si en la primera columna preguntamos cuánto le falta a 0 para llegar a 1, la respuesta es 1.

En la segunda columna, la pregunta ¿cuánto le falta a 1 para llegar a 0? no podrá responderse, ya que el minuendo es menor que el sustraendo y, por lo tanto, le debe "pedir 1" a la columna siguiente, con lo que la pregunta pasa a ser ¿cuánto le falta a 1 para llegar a 10? La respuesta es 1, y el pedido de una unidad a la columna siguiente se encuentra reflejado con un 1 entre el minuendo y el sustraendo en esa columna.

Es importante mencionar que "pedir 1" significa pedirle a la base siguiente una unidad, que "siempre" va a ser equivalente a 10, y sumársela al minuendo.

Para llegar al resultado de la tercera columna, en este caso, hay que realizar un paso previo antes de hacer la pregunta, que consiste en sumar al sustraendo la unidad que se había "pedido" con anterioridad y luego preguntar. De esta manera, 1 más 1 es 0 con acarreo a la columna siguiente. El resultado de esta columna es 0 y el pedido de unidad se encuentra reflejado en la columna siguiente.

La pregunta de la cuarta columna se contesta rápido: ¿cuánto le falta a 1 para llegar a 1? La respuesta es 0.

No desespere, es más difícil explicar el desarrollo de estos métodos que aplicarlos

2do. caso)

$$\begin{array}{r} 11011_2 \\ - 1110_2 \\ \hline 01101_2 \end{array}$$

$$\begin{array}{r} 27_{10} \\ - 14_{10} \\ \hline 13_{10} \end{array}$$

$$\begin{array}{r} 01101_2 \\ | \\ 1 \cdot 2^0 = 1_{10} \\ | \\ 0 \cdot 2^1 = 0_{10} \\ | \\ 1 \cdot 2^2 = + 4_{10} \\ | \\ 1 \cdot 2^3 = 8_{10} \\ | \\ 0 \cdot 2^4 = 0_{10} \\ \hline \text{Comprobación} & 13_{10} \end{array}$$

En la resta $11011_2 - 1110_2$, si preguntamos, para la primera columna, ¿cuánto le falta a 0 para llegar a 1?, la respuesta es simple: 1.

En la segunda columna, si hacemos la pregunta ¿cuánto le falta a 1 para llegar a 1?, la respuesta es 0.

En la tercera columna, para la pregunta ¿cuánto le falta a 1 para llegar a...?, ahora ya sabemos que tiene que ser 10, la respuesta es 1 y en la columna siguiente se indicó con un 1 la solicitud de una unidad para hacer efectiva la resta.

En la cuarta columna, un paso previo estipula que se debe sumar el sustraendo con la unidad que se había “pedido” y el resultado restarlo del minuendo. Luego, 1 más 1 es 10 y la pregunta para hacer es ¿cuánto le falta a 10 para llegar a...?, en este caso 11. Usted podrá preguntarse ¿por qué 11? El mecanismo es siempre el mismo: si se le pide una unidad a la columna siguiente, entrega el equivalente a la base (siempre 10) y al sumárselo, en este caso a 1, queda 11. No olvide reflejar el pedido en la columna de orden superior.

Para finalizar, la última columna es sencilla. A la pregunta ¿cuánto le falta a 1 para llegar a 1?, respondemos 0.

Los mecanismos propuestos para la suma y la resta quizás resulten un tanto tediosos, sobre todo si usted ya aprendió a operar con binarios; no obstante, es probable que le den un porcentaje de seguridad en la operatoria que otros mecanismos no poseen.

2.5 Operaciones fundamentales en octal y hexadecimal

La técnica para las operaciones en octal o hexadecimal es la misma que para cualquiera de las operaciones en un sistema posicional, por lo tanto, sólo se desarrollará la suma para ambos sistemas. Tenga en cuenta que lo único que cambia es la base del sistema. Además, en relación con la aplicación que usted pueda darle en su ámbito laboral, lo más probable es que tenga que sumar o restar direcciones de memoria expresadas en estos sistemas. Es muy probable que jamás tendrá que realizar una multiplicación u operaciones más complejas; por lo tanto, su estudio se considera innecesario para los fines prácticos de este libro.

Para analizar los ejemplos que se brindan, siga en forma detallada los esquemas presentados, que le permitirán reconocer el mecanismo empleado.

2.5.1 Suma octal

Suponga las siguientes sumas y su comprobación en decimal:

$$\begin{array}{r}
 12_{(8)} \\
 + 2_{(8)} \\
 \hline ?
 \end{array}
 \quad
 \begin{array}{r}
 1 \cdot 8^1 = 8_{(10)} \\
 2 \cdot 8^0 = + 2_{(10)} \\
 \hline 10_{(10)}
 \end{array}$$

$$\begin{array}{r}
 12_{(8)} \\
 + 2_{(8)} \quad \longrightarrow + 2_{(10)} \\
 \hline 14_{(8)} \quad \boxed{12_{(10)}}
 \end{array}$$

$$\begin{array}{r}
 4 \cdot 8^0 = 4_{(10)} \\
 1 \cdot 8^1 = + 8_{(10)} \\
 \hline \boxed{12_{(10)}}
 \end{array} \quad \text{Comprobación}$$

$$\begin{array}{r}
 17_{(8)} \\
 + 1_{(8)} \\
 \hline ?
 \end{array}
 \quad
 \begin{array}{r}
 1 \cdot 8^1 = 8_{(10)} \\
 7 \cdot 8^0 = + 7_{(10)} \\
 \hline 15_{(10)}
 \end{array}$$

$$\begin{array}{r}
 17_{(8)} \\
 + 1_{(8)} \quad \longrightarrow \boxed{+ 1_{(10)}}
 \end{array}$$

$$\begin{array}{r}
 20_{(8)} \quad \boxed{16_{(10)}}
 \end{array}$$

$$\begin{array}{r}
 0 \cdot 8^0 = 0_{(10)} \\
 2 \cdot 8^1 = + 16_{(10)} \\
 \hline \boxed{16_{(10)}}
 \end{array} \quad \text{Comprobación}$$

2.5.2 Técnica para sumar números grandes en cualquier base

Si, por ejemplo, se desea realizar la siguiente suma, $7_{(8)} + 7_{(8)} + 7_{(8)}$, la técnica para sumar números grandes en cualquier base es la que se describe a continuación:

- Poner en columnas los números y sumarlos, en base decimal:

$$\begin{array}{r}
 7 \\
 + 7 \\
 \hline
 7 \\
 \hline
 27_{(10)}
 \end{array}$$

- Al resultado de la suma de la columna anterior ($27_{(10)}$), restarle la base de origen (8) las veces necesarias hasta que el resto sea menor que esa base:

$$\begin{array}{r}
 21 \\
 - 8 \\
 \hline
 13 \\
 - 8 \\
 \hline
 5
 \end{array}
 \quad
 \begin{array}{l}
 \text{el número de veces (2) que se restó la base} \\
 \text{es el acarreo para la columna siguiente} \\
 \hline
 + 7 \\
 \hline
 7
 \end{array}
 \quad
 \begin{array}{r}
 + 7 \\
 \hline
 7 \\
 \hline
 25
 \end{array}$$

el resto (5), menor que la base,
 es el dígito que corresponde a la suma de
 la columna en la base de origen

Luego, el resultado de la suma, en la base de origen, se obtendrá de la siguiente manera:

- El último resto (dígito menor que la base) será el dígito que corresponda a la resta final de las sucesivas restas, en este caso 5.
- La cantidad de veces que se haya restado la base será el acarreo para la columna siguiente, en este caso 2.

Si los números para sumar tienen más de una columna, la técnica se repetirá para cada una de ellas y el acarreo se sumará a la columna siguiente.

2.5.3 Suma hexadecimal

Suponga las siguientes sumas y su comprobación en decimal:

$$\begin{array}{r}
 12_{(16)} \\
 + 2_{(16)} \\
 \hline
 14_{(16)}
 \end{array}
 \quad
 \begin{array}{l}
 1 \cdot 16^1 = 16_{(10)} \\
 2 \cdot 16^0 = + 2_{(10)} \\
 \hline
 18_{(10)}
 \end{array}$$

$12_{(16)}$
 $+ 2_{(16)}$ ————— $+ 2_{(10)}$
 $14_{(16)}$ $20_{(10)}$
 $4 \cdot 16^0 = 4_{(10)}$
 $1 \cdot 16^1 = + 16_{(10)}$
 $20_{(10)}$

Comprobación

$$\begin{array}{r}
 17_{(16)} \\
 + 1_{(16)} \\
 \hline
 ?
 \end{array}
 \quad
 \begin{array}{l}
 1 \cdot 16^1 = 16_{(10)} \\
 7 \cdot 16^0 = + 7_{(10)} \\
 \hline
 23_{(10)}
 \end{array}$$

$17_{(16)}$
 $+ 1_{(16)}$ ————— $+ 1_{(10)}$
 $18_{(16)}$ $24_{(10)}$
 $8 \cdot 16^0 = 8_{(10)}$
 $1 \cdot 16^1 = + 16_{(10)}$
 $24_{(10)}$

Comprobación

$$\begin{array}{r}
 17_{(16)} \\
 + 1_{(16)} \\
 \hline
 ? \\
 \\[1ex]
 \boxed{17}_{(16)} \\
 + \boxed{1}_{(16)} \quad + \boxed{1}_{(16)} \\
 \boxed{18}_{(16)} \quad \quad \quad \boxed{24}_{(16)} \\
 \\[1ex]
 1 \cdot 16^1 = 16_{(10)} \\
 7 \cdot 16^0 = + 7_{(10)} \\
 \hline
 23_{(10)} \\
 \\[1ex]
 8 \cdot 16^0 = 8_{(10)} \\
 1 \cdot 16^1 = + 16_{(10)} \\
 \hline
 \boxed{24}_{(10)} \\
 \end{array}
 \text{Comprobación}$$

$$\begin{array}{r}
 1F \\
 + 1 \\
 \hline
 ? \\
 \\[1ex]
 \boxed{1F}_{(16)} \\
 + \boxed{1}_{(16)} \quad + \boxed{1}_{(16)} \\
 \boxed{20}_{(16)} \quad \quad \quad \boxed{32}_{(16)} \\
 \\[1ex]
 1 \cdot 16^1 = 16_{(10)} \\
 15 \cdot 16^0 = + 15_{(10)} \\
 \hline
 31_{(10)} \\
 \\[1ex]
 0 \cdot 16^0 = 0_{(10)} \\
 2 \cdot 16^1 = + 32_{(10)} \\
 \hline
 \boxed{32}_{(10)} \\
 \end{array}
 \text{Comprobación}$$

Utilizando la técnica de la suma para números grandes:

$$\begin{array}{r}
 F_{(16)} \\
 + F_{(16)} \\
 \hline
 \boxed{F}_{(16)} \\
 \boxed{2D}_{(16)} \\
 \text{la base se} \\
 \text{restó 2 veces} \\
 \\[1ex]
 + 15_{(10)} \\
 \hline
 15_{(10)} \\
 45_{(10)} \\
 - 16_{(10)} \\
 \hline
 29_{(10)} \\
 - 16_{(10)} \\
 \hline
 \boxed{13} \\
 \end{array}
 \begin{array}{l}
 15_{(10)} \\
 + 15_{(10)} \\
 \hline
 15_{(10)} \\
 45_{(10)} \\
 - 16_{(10)} \\
 \hline
 29_{(10)} \\
 - 16_{(10)} \\
 \hline
 \boxed{13} \\
 \end{array}
 \begin{array}{l}
 \text{equivalente a "D" para} \\
 \text{la base hexadecimal} \\
 \end{array}$$

2.6 Complemento de un número

Dado que muchas computadoras digitales basan su operatoria aritmética sobre todo en la suma, la mayoría de las operaciones aritméticas debe transformarse a una técnica que la incluya. Por eso, es necesario inferir la idea de complemento de un número, que permitirá, entre otras cosas hacer restas mediante sumas, así como representar números signados. Hay dos métodos que permiten representar el complemento de un número y se detallan a continuación.

2.6.1 Complemento a la base, a la raíz o auténtico

En todo sistema numérico de notación posicional, para un número x de n dígitos existe un número x' , también de n dígitos, que es su complemento a la base; tal que

$$x + x' = 10^n, \quad \text{de lo cual se deduce que}$$

$$x' = 10^n - x, \quad \text{siendo } 10 \text{ la base en cualquier sistema.}$$

Otra forma de definirlo es indicar que el complemento a la base de un número N es el resultado de elevar la base a la cantidad de cifras del número, menos el número dado:

$$C_b N = B^n - N$$

En el sistema binario el complemento a la base suele llamarse "complemento a 2", lógicamente por ser 2 la base del sistema, y suele abreviarse C_2 .

Si, por ejemplo, queremos obtener el complemento a la base de los números $0_{(2)}$, $1_{(2)}$, $10_{(2)}$ y $11_{(2)}$, debemos restar estos números de sus respectivas bases, elevadas a la cantidad de dígitos que ellos poseen, o sea de $10^1 = 10$ y de $10^2 = 100$.

$$\begin{array}{r} 10 \\ - 0 \\ \hline 10 \end{array} \quad \begin{array}{r} 10 \\ - 1 \\ \hline 1 \end{array} \quad \begin{array}{r} 100 \\ - 10 \\ \hline 10 \end{array} \quad \begin{array}{r} 100 \\ - 11 \\ \hline 01 (*) \end{array}$$

Obsérvese que en todos los casos el complemento de un número es su inverso más 1, o sea, el número inverso del sustraendo + 1.

2.6.2 Su utilización para la representación de binarios negativos complementados a “2”

Si se desea convertir el decimal -3 a un binario con signo de 3 bits, se debe considerar la siguiente estructura:

- El bit de extrema izquierda representa con un “1” el signo negativo.
- Los 2 bits restantes representan el complemento a 2 del valor binario real, que es 11, que invertido es 00 y más 1 es 01.



La cantidad de bits n , utilizados para la representación de un número binario con signo, siempre **incluye** el signo.

Para descomplementar un número, o sea, para hallar de nuevo su valor absoluto, el método es el mismo que para hallar el complemento (complemento de complemento).

2.6.3 Complemento a la base -1 o restringido

En todo sistema numérico de notación posicional para un número y de n dígitos, existe un número y' , también de n dígitos, que es su complemento restringido; tal que

$$y + y' = 10^n - 1, \quad \text{de lo cual se deduce que}$$

$$y' = (10^n - 1) - y, \quad \text{siendo } 10 \text{ la base en cualquier sistema.}$$

Otra manera de definirlo sería indicar que el complemento a la base -1 de un número N es el resultado de elevar la base a la potencia dada por la cantidad de cifras del número, menos 1, y luego restarle el número dado.

$$C_{b-1} N = (B^n - 1) - N$$

En el sistema binario el complemento a la base -1 es el complemento a 1 ; obviamente si la base es 2 , la base -1 es 1 y suele abreviarse C_1 . Coincide con el complemento booleano, o sea que para hallarlo se cambian los 1 por 0 y al revés.

*Si al complemento restringido se le suma 1
se obtiene el complemento auténtico.*

Si se quiere hallar el complemento restringido de los números $0_{(2)}$ y $11_{(2)}$ resulta:

$$\begin{array}{r} 1 & \quad 11 \\ - 0 & \quad -11 \\ \hline 1 & \quad 00 \end{array} \quad \begin{array}{l} \text{nótese que es el} \\ \text{inverso del sustraendo} \end{array}$$

2.6.4. Su utilización para la representación de binarios negativos complementados a “1”

Si se desea convertir el decimal -3 a un binario con signo de 3 bits, se debe considerar la estructura siguiente:

- El bit de extrema izquierda representa con un “1” el signo negativo.
- Los 2 bits restantes expresan el complemento a 1 del valor binario real.

1	0	0
---	---	---

Signo *Complemento (calculado en para 11)*

Regla práctica para hallar el complemento de un número binario

Restringido (C1): *se invierten todos los dígitos.*

$$C_{(1)} 0 = 1 \text{ el complemento restringido de } 0 \text{ es } 1.$$

$$C_{(1)} 11 = 00 \text{ el complemento restringido de } 11 \text{ es } 00.$$

Auténtico (C2): *Para hallar el complemento a 2 se invierten todos los dígitos y se le suma 1.*

$$C_{(2)} 11 = 00 + 1 = 01$$

$$C_{(2)} 101 = 010 + 1 = 011$$

Comprobación: 1000

$$- \underline{101}$$

$$011$$

2.7 Resumen

El sistema de números binarios ofrece muchas ventajas en el manejo interno de la información en las computadoras, respecto de la representación decimal que utilizamos los usuarios. Esto se debe en primer término, a la mayor simplicidad de diseño de una unidad aritmética binaria o a la facilidad de transferir y almacenar magnitudes físicas que representen bits. No obstante, los números decimales son esenciales para la comunicación entre el usuario y la máquina, y, por lo tanto, se justifica el aprendizaje de los métodos de conversión entre ambos sistemas.

Revisar la información textual del contenido de las largas cadenas binarias alojadas en una memoria puede resultar complicado, por lo tanto, para abreviar secuencias de ceros y unos, se utilizan los sistemas numéricos octal y hexadecimal. Esto es posible debido a que el pasaje directo entre las bases 2 y 8, y el pasaje directo entre las bases 2 y 16 son factibles, ya que 8 es potencia exacta de 2 ($2^3 = 8$) y 16 también lo es ($2^4 = 16$); por lo tanto, en el primer caso una cadena de tres bits puede ser reemplazada por un único dígito octal, y en el segundo, una cadena de cuatro bits, por un único dígito hexadecimal. Un informático debe poder leer información hexadecimal y pensarla en binario en forma automática. Por ejemplo, una impresora "X" puede enviar el siguiente código hexadecimal, correspondiente a su estado actual "**D0**", que debe ser interpretado por el software que lee ese byte de estado como **11010000**; por lo tanto, el envío de un byte puede "programarse" para su envío o no, de acuerdo con la interpretación que se establezca para cada uno de los bits. Por ejemplo, si:

D				O			
1	1	0	1	0	0	0	0
7	6	5	4	3	2	1	0

El bit **7** significa encendida.

El bit **6** significa no ocupada.

El bit **5** significa sin papel.

El bit **4** significa seleccionada entre otras impresoras.

El bit **3** significa atasco de papel.

El bit **2** significa bajo nivel de tinta.

El bit **1** significa error de dispositivo.

El bit **0** significa tiempo excedido para enviar el byte.

Entonces, dada la combinación obtenida del pasaje directo de hexadecimal a binario, se interpreta que la impresora está encendida, no ocupada y seleccionada entre otras impresoras. A partir de allí el programa puede enviar el byte para imprimir, puesto que las condiciones establecidas le permiten la impresión. Por esta razón, se justifica el esfuerzo de aprender ambos sistemas numéricos y los métodos de sus conversiones a binario.

Otro ejemplo es la manera en que se utilizan ambos sistemas –el decimal y el hexadecimal– en una configuración de acceso a la red:

```
Adaptador Ethernet Conexión de área local : 
Sufijo de conexión específica DNS : cpe.telecentro.net.ar
Descripción. . . . . : Adaptador Fast Ethernet compatible V
IA : 00-0C-61-00-00-00
Dirección física : 00-0C-61-00-00-00
DHCP habilitado. : No
Autoconfiguración habilitada. : Sí
Dirección IP. . . . . : 200.115.237.194
Máscara de subred : 255.255.252.0
Puerta de enlace predeterminada : 200.115.236.1
Servidor DHCP : 200.115.236.1
Servidores DNS : 200.115.192.30
                           200.115.192.28
                           200.115.192.29
Concesión obtenida . . . . . : Miércoles, 28 de Enero de 2009 13:28
Concesión expira . . . . . : Viernes, 30 de Enero de 2009 13:18:3
:28
7
C:\Documents and Settings\Usuario>
```

donde la dirección física del adaptador está indicada en hexadecimal y la dirección IP fue convertida a decimal por la propia computadora. Si esto no es así, hay que convertir los hexadecimales de esa dirección IP a decimal y obtener, de esta manera, lo que se muestra como "200", "115", "237" y "194" en **Dirección IP 200.115.237.194**.

Por último, el complemento de un número binario permite operar restas mediante sumas de una forma muy simple para los componentes electrónicos.

2.8 Ejercicios propuestos

- 1) Defina qué significan las siglas LSB y MSB.
- 2) Explique por qué cada dígito tiene un valor distinto y relativo a su posición en un sistema numérico de notación posicional.
- 3) Convierta los números binarios siguientes a sus correspondientes en sistema decimal:
 - a) 10111
 - b) 11111
 - c) 10000001
 - d) 10,1
 - e) 10,101
 - f) 100,011
 - g) 1011,0101
 - h) 1100,011
- 4) Convierta los números decimales siguientes a sus correspondientes en sistema binario:
 - a) 55
 - b) 48
 - c) 204
 - d) 237
 - e) 255,75
 - f) 10,4
 - g) 83,45
- 5) Sin hacer ningún cálculo, convierta, si es posible, los números binarios siguientes a sus equivalentes en sistemas decimal, octal y hexadecimal:
 - a) 0001
 - b) 0101
 - c) 1000
 - d) 1011
- 6) ¿La notación hexadecimal se usa ampliamente en el ámbito de la computación como método taquigráfico para representar números binarios, decimales o de ambos sistemas?
- 7) Convierta los números hexadecimales siguientes a sus correspondientes en sistema decimal:
 - a) 7E
 - b) DB
 - c) 12A3
 - d) 34CF
- 8) Convierta los números decimales siguientes a sus correspondientes en sistema hexadecimal:
 - a) 48.373
 - b) 217
 - c) 16
 - d) 3,25
 - e) 101,55
- 9) ¿Cuántos enteros positivos se pueden expresar con K dígitos, usando números en base r?
- 10) Convierta "directamente", si es posible, los números siguientes. Justificar verbalmente en ambos casos.
 - a) $310,10_{(16)-(2)}$
 - b) $310,10_{(2)-(16)}$
 - c) $310,10_{(8)-(2)}$
 - d) $310,10_{(8)-(10)}$
 - e) $103_{(16)-(10)}$
 - f) $3,2_{(3)-(24)}$
 - g) $10210_{(5)-(25)}$

- | 1) Convierta "directamente" los números hexadecimales siguientes a sus equivalentes binarios.
- 2EA,F65
 - 57,24
 - 3A,B
 - 10,10
 - F,003
- | 2) Convierta los números hexadecimales siguientes a sus equivalentes binarios, pasando previamente por la base 10 (base 16 base 10 base 2), tanto los enteros como las fracciones.
- E2
 - 3D
 - A0
 - 2,1
 - F,01
 - 1,F
 - H,A
- | 3) Convierta:
- 19,75_{(10)→(2)}
- 19,3_{(10)→(2)}
- 13,9_{(10)→(16)}
- | 4) Indique los números enteros anterior y posterior, en el mismo sistema:
- 59F₍₁₆₎
 - 777₍₈₎
- | 5) Calcule el complemento auténtico y el restringido de los números 99₍₁₀₎ y 101₍₂₎.
- | 6) Dado el complemento auténtico siguiente, si es posible, indicar x para las bases 9, 3 y 16:
- | | |
|------|-----|
| 1000 | () |
| - x | () |
| 681 | () |
- | 7) Deduzca cómo efectuarla la resta, por complemento a la base del sustraendo, de los números siguientes. Resuelva.
- 210₍₁₀₎ - 311₍₁₀₎
- | 8) Convierta a base 2 el número decimal 83,1. Reconviértalo a base 10, teniendo en cuenta 6 bits fraccionarios. ¿A qué conclusiones arriba respecto de la parte fraccionaria?
- | 9) 10101001 es un número negativo expresado en complemento a 2. Convierta este número en binario positivo y su equivalente decimal y luego exprese el número negativo binario en decimal y hexadecimal.
- | 10) Deduzca cómo representar el número negativo 10011110 con una longitud de 16 bits. Expréselo en sistemas binario y hexadecimal.

2.9 Contenido de la página Web de apoyo

El material marcado con asterisco (*) sólo está disponible para docentes.



Resumen gráfico del capítulo

Simulación

Herramienta interactiva que permite realizar conversiones y operaciones entre sistemas numéricos.

Autoevaluación

Evaluaciones Propuestas*

Presentaciones*

3

Representación de datos en la computadora

Contenido

3.1 Introducción	50
3.2 Flujo de datos dentro de una computadora	50
3.3 Códigos de representación de caracteres alfanuméricos	53
3.4 Códigos de representación decimal (BCD)	56
3.5 Códigos de representación numérica no decimal.....	59
3.6 Representaciones redundantes	68
3.7 Resumen.....	71
3.8 Ejercicios propuestos	71
3.9 Contenido de la página Web de apoyo.....	72

Objetivos

- Reconocer los códigos de representación de caracteres (alfanuméricos y numéricos).
- Reconocer los formatos numéricos: Representaciones decimales, representaciones numéricas para números enteros o para números reales.
- Reconocer sólo algunas representaciones redundantes que permiten la detección y corrección de errores.
- Comprender los convenios de representación en Punto Flotante Exceso IEEE-P754.



En la página Web de apoyo encontrará un breve comentario de la autora sobre este capítulo.

3.1 Introducción

En el desarrollo de este capítulo se describe la representación interna de los formatos de "tipos de datos" que los lenguajes de programación permiten declarar, para las distintas estructuras de dato. Por ejemplo: una variable *string* se almacenará en un código de representación alfanumérico, como UNICODE o ASCII; una variable *unsigned integer*, como un número entero no signado, conocido como "de coma fija" o "de punto fijo"; un número como el $7,5_{(10)}$ se almacenará como un número de "punto flotante" o "coma flotante". La mayoría de ellos se ajusta a convenios estandarizados, de los cuales en este capítulo se consideraron los más difundidos en los diseños de las computadoras actuales. Todos pasamos por la experiencia de comprar un teclado para nuestra computadora personal; un comprador no especializado desconoce que los teclados de distintas marcas responden a un estándar que independiza la compra del dispositivo respecto de la marca del fabricante de su propia computadora, ya que el ingreso de datos por teclado responde al estándar de representación de códigos alfanuméricos.

En cuanto a datos "operables", los especialistas del área deben conocer cuál es la diferencia entre la representación de un entero y la de una fracción o de un número real.

En esta introducción hacemos referencia a que utilizaremos los términos "coma fija" o "punto fijo" en forma indistinta en relación con el término en inglés *fixed-point*; de la misma manera se utilizarán los términos "coma flotante" o "punto flotante" respecto de las representaciones *floating-point*. Esto se debe a que la "coma" se utiliza en nuestro sistema numérico como el separador de decimales, y en el caso de la palabra "punto" corresponde a la traducción literal del término en inglés *point*. Se considera que ambos son aceptables, dado que aparecen de manera indistinta en la bibliografía consultada.

3.2 Flujo de datos dentro de una computadora

Una computadora está constituida por una memoria, que almacena programas y datos en forma temporal, y un órgano ejecutor de estos programas conocido como unidad central de proceso (CPU).

Si se la analiza desde un ejemplo más cotidiano, pero práctico, la memoria cumple la función de una hoja en la que se escribieron los pasos para la resolución de un problema y los datos que intervienen en los cálculos y donde se volcarán los resultados obtenidos. A su vez, usted interpreta los pasos, toma los datos y los vuelca en una calculadora de bolsillo. Dentro de la computadora, quien realiza esta actividad es la unidad de control (CU) y la de la calculadora de bolsillo es la unidad aritmético-lógica (ALU). Así, el conjunto CU-ALU constituye la ya mencionada CPU.

Las variaciones en el diseño interno de las distintas computadoras actuales no modifican este esquema global, pero sí afectan el procesamiento de los programas y los datos que deben transformarse en el interior para adaptarse a la estructura de cada computadora. Sin embargo, el programador no es el responsable de implementar estas transformaciones; por ejemplo, la compilación de un programa en lenguaje simbólico permite adaptar su estructura y generar el programa en el código de máquina que la computadora puede entender. Entonces hay un "mediador" entre los programas y los datos que vienen "de afuera" y las necesidades internas para procesarlos. Este vínculo que pasa de lo estándar a lo particular en cada equipo es una rutina de conversión enlazada al código del programa.

En resumen, tanto las instrucciones de un programa como los datos que ingresan en la memoria desde el exterior lo hacen en un código alfanumérico de representación de caracteres, por ejemplo, el ASCII. Las instrucciones son transformadas por un programa de traducción (que puede ser un ensamblador, un traductor o un compilador) a código de máquina, que es el que entiende el procesador, para luego ejecutarse. Los encargados de la transformación de los datos, respetando el formato definido en el programa, suelen ser rutinas previstas por

bibliotecas estándar del lenguaje e incluidas en el programa. Por lo tanto, aunque los datos ingresen desde el teclado en un código alfanumérico de caracteres, éstos se almacenarán en la memoria de distintas formas, según los requerimientos determinados por la declaración de los datos del programa; éste es el único que puede reconocer si el byte de memoria accedido es, por ejemplo, un operando o un carácter alfanumérico (fig. 3.1).

Por otra parte, una computadora necesita comunicarse con el medio externo por medio de dispositivos de entrada o salida, y éstos se diseñan para conectarlos a una amplia gama de computadoras diferentes entre sí, lo que induce a pensar que los periféricos utilizan códigos estándar para la representación de información. La forma de ingreso o egreso de datos y programas en una computadora respeta estos códigos de representación de caracteres que abastecen las necesidades de **intercambio** de información a nivel general. Esto es independiente de las transformaciones que se realizan para su tratamiento interno.

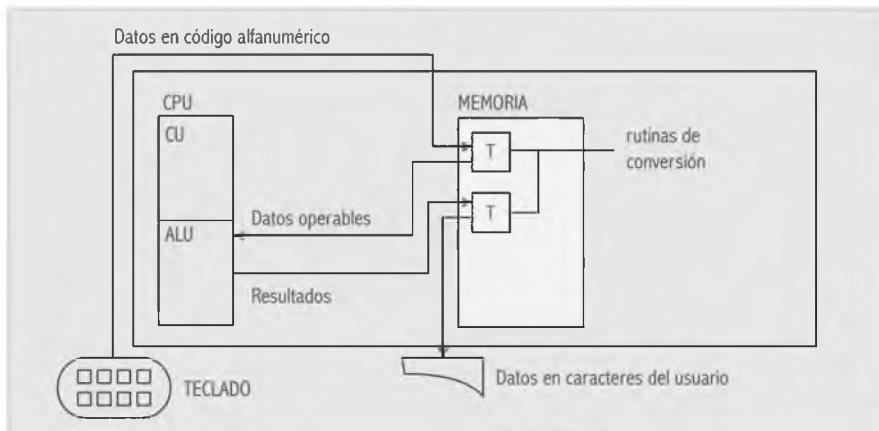
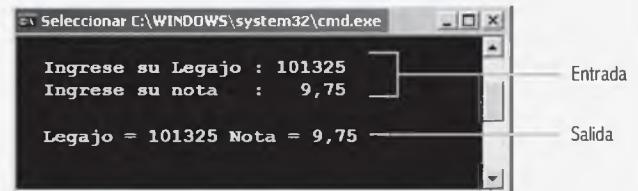


Fig. 3.1. Conversión de los datos durante el procesamiento.

Para ilustrar el flujo de datos utilizaremos un programa que permite ingresar dos datos de diferentes tipos: uno entero y uno real, utilizando la especificación de formato en lenguaje C. La función `scanf` permite el ingreso de los datos, su sintaxis es la siguiente: `scanf("Formato", &identificación de la variable)`. `scanf("%d", &x);` permite el almacenamiento de un número entero en la variable x, `scanf("%f", &m);` permite el almacenamiento de un número en coma flotante en la variable m. La variable representa un lugar en memoria en la que se almacenará un valor que el usuario ingresará por el teclado. Luego de cada ingreso se presiona la tecla Enter.

```
#include< stdio.h > //incluye la biblioteca estándar que contiene a scanf y a printf
void main()
{
// Definición de la variables
int legajo;
float nota;
clrscr();
//Ingreso de Datos con función scanf
printf("Ingrese su legajo:");
scanf("%d",&legajo);
printf("Ingrese su nota con dos decimales:");
scanf("%f",&nota);
// Salida de datos ingresados con función fprintf
printf("\n\n Legajo=%d Nota=%g",legajo, nota);
}
```



Un **código alfanumérico** establece la relación necesaria para que una computadora digital, que procesa solamente dígitos binarios, interprete el lenguaje que utiliza el usuario (caracteres numéricos, alfábéticos o especiales). Un código es una convención que determina una única combinación binaria para cada símbolo que se ha de representar. Por ejemplo, con 7 bits se obtienen $2^7 = 128$ combinaciones distintas, que son suficientes para asignar un símbolo alfábético, numérico o especial a cada una de ellas.

Si bien se puede establecer infinidad de códigos alfanuméricos de "entendimiento" entre el usuario y las distintas computadoras, se desarrollaron convenios de uso internacional, cuya finalidad es concretar la compatibilidad entre dispositivos de distinto origen.

Cada carácter emitirá una serie de señales binarias al "digitarse" sobre el teclado, que quedará alojada internamente como una combinación de bits, de acuerdo con alguna representación de caracteres alfanuméricos preestablecida.

Supóngase que se ingresa el dato alfanumérico "+105", el proceso de codificación hará que éste quede registrado en la memoria, por ejemplo, en código ASCII, como se muestra en la figura 3.2.



Registros de memoria	
0010 1011	+
0011 0001	1
0011 0000	0
0011 0101	5

Fig. 3.2. Símbolos del usuario y su representación binaria en ASCII

Formato de una entidad binaria: es la estructura y la cantidad de bits de un determinado tipo de dato para su tratamiento en la computadora.

Se denomina **formato** de una entidad binaria a la estructura y la cantidad de bits de un determinado tipo de dato para su tratamiento en la computadora. Los formatos pueden ser de longitud fija o variable, según la categoría de dato que represente. Es importante destacar que el que interpreta un grupo de bits bajo un formato determinado es un programa. Esto significa que una cadena de bits en un registro de la ALU puede ser interpretada por el programa que la utiliza como operando, como un binario sin signo o con él, o simplemente puede representar una parte del operando en sí, como la mantisa de un operando definido como real. Del mismo modo, un byte alojado en la memoria puede ser un carácter alfanumérico, un operando o incluso una instrucción, según quién lo "interprete".

En la tabla 3-1 se detallan distintos tipos de representaciones de datos que una computadora puede manejar y que más adelante se explicarán. Debe quedar claro que, en general, del conjunto de posibilidades para cada una de las representaciones, sólo se utilizará una de ellas. Por ejemplo, para una computadora las representaciones alfanuméricas siempre se interpretarán en código ASCII, las representaciones decimales se manejarán en BCD puro, los números enteros se expresarán en complemento a la base y los números reales utilizarán una representación en IEEE P754. De manera independiente, la misma computadora puede utilizar formatos de números enteros en punto fijo sin signo o con él, ya que ello depende de la definición del tipo de dato que el programa de usuario necesite.

Tabla 3-1. Distintos tipos de representaciones de datos.

Representaciones alfanuméricas	ASCII UNICODE	
Representaciones decimales (BCD)	BCD puro o natural (8421) BCD exceso tres BCD 2421 o AIKEN	
Representaciones binarias	Números enteros	Coma o punto fijo sin signo (enteros-positivos) Coma o punto fijo con signo (enteros) Coma o punto fijo con complemento a la base (enteros) Coma o punto fijo con complemento restringido (enteros)
	Números reales	Coma o punto flotante (entera y fraccionaria) Coma o punto flotante con mantisa normalizada Coma o punto flotante IEEE P754 Coma o punto flotante "exceso 64"
Representaciones redundantes (detección de errores)	Códigos de paridad	Paridad vertical o a nivel carácter Paridad vertical o a nivel bloque Paridad entrelazada Código de Hamming

3.3 Códigos de representación de caracteres alfanuméricicos

Si bien una computadora es una "calculadora aritmética" por excelencia, gran parte de sus aplicaciones consiste en el tratamiento de cadenas de caracteres llamadas *strings*. Un *string* es una unidad formada por caracteres representados según un código alfanumérico preestablecido para esa computadora. Uno de los códigos de representación de caracteres más usado es el "ASCII".



El código ASCII de 7 bits permite determinar $2^7 = 128$ combinaciones posibles que representan 128 símbolos distintos.

3.3.1 Código ASCII

El código **ASCII** de 7 bits (*American Standard Code for Information Interchange* o Código Estándar Americano para Intercambio de Información) representa cada carácter con 7 bits, que permite determinar $2^7 = 128$ combinaciones distintas, suficientes para establecer una única relación carácter-combinación binaria, que se representa en la tabla 3-2.

Tabla 3-2. Tabla de código ASCII (de 7 bits).

HEX. N°	0	1	2	3	4	5	6	7	
HEX. N°	BITS 654 3210	000	001	010	011	100	101	110	111
0	0000	NUL	DLE	SP	0	@	P	-	p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	\$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	'	7	G	W	g	w
8	1000	BS	CAN	(8	H	X	h	x
9	1001	HT	EM)	9	I	Y	i	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESC	+	:	K	[k	{
C	1100	FF	FS	,	<	L	\	l	
D	1101	CR	GS	-	=	M]	m	}
E	1110	SO	RS	.	>	N	^	n	-
F	1111	SI	US	/	?	O	_	o	DEL



Encuentre un simulador sobre codificación ASCII en la página Web de apoyo.

Significado en inglés de los caracteres de control de las columnas 0 y 1		
NUL, null	VT, vertical tabulation	SYN, synchronous idle
SOH, start of heading	FF, form feed	ETB, end of transmission block
STX, start of text	CR, carriage return	CAN, cancel
ETX, end of text	SO, shift out	EM, end of medium
EOT, end of transmission	SI, shift in	SUB, substitute
ENQ, enquiry	DEL, data link escape	ESC, escape
ACK, acknowledge	DC1, device control 1	FS, file separator
BEL, bell (audible signal)	DC2, device control 2	GS, group separator
BS, backspace	DC3, device control 3	RS, record separator
HT, horizontal tabulation (punched card skip)	DC4, device control 4	US, unit separator
LF, line feed	NAK, negative acknowledge	DEL, delete

Este patrón se puede definir como “oficial” y en todos los textos en los que se incluya esta tabla no habrá variaciones que den lugar a pensar que “hay varios ASCII”.

Los primeros 3 bits (6, 5 y 4) de mayor significación en cada combinación se denominan bits de **zona** y en la tabla de doble entrada numeran las columnas 0 a 7.

Los últimos 4 bits (3, 2, 1 y 0) en cada combinación se denominan bits de **dígito** y en la tabla numeran las filas de 0 a 15, o de 0 a F en hexadecimal.

Los primeros 32 caracteres pueden parecer “raros”, sin embargo, son de gran utilidad porque permiten **cierto control** de las actividades de la computadora. Por ejemplo, para tabular hay una tecla o dos en el teclado que permiten “ocupar” un lugar en la pantalla con la combinación 0000 1001; aunque no se visualice, “marcan” ese lugar para que el cursor salte allí. El carácter BEL permite emitir un sonido que puede utilizarse como aviso al operador de la computadora. Los códigos SO y SI, shift activo o inactivo, se utilizan para ordenar a la impresora que imprima caracteres estándar o condensados. El resto de los caracteres es familiar para el usuario: Letras mayúsculas y minúsculas, dígitos, signos de puntuación, etcétera. Sin embargo, hay ciertas cuestiones para tener en cuenta. Por un lado, las mayúsculas se consideran distintas a las minúsculas, aun cuando se trate de la misma letra, debido a que su forma de representación **simbólica** es diferente por completo (A vs. a). Las combinaciones que representan letras están numéricamente ordenadas, esto es, la combinación de “A” es menor que la de “B”, lo que permite ordenar alfabéticamente un conjunto de datos. Por último, el espacio en blanco está definido por la combinación 0010 0000 porque el blanco o el espacio “ocupan lugar aunque no se vean”.

Para terminar, es importante saber que la combinación binaria que representa un carácter también se expresa en sus formas hexadecimal y decimal de la siguiente forma:

$$\text{“A”} = 100\ 0001 = 41h = 65d \quad h = \text{hexadecimal} \quad d = \text{decimal}$$



ASCII de 8 bits: a causa de la necesidad de agregar caracteres, el código se amplió para poder representar 128 símbolos más.

3.3.2 Código ASCII ampliado

Si bien el ASCII se concibió como código de 7 bits, su uso generalizado dio lugar a una ampliación a causa de la necesidad de agregar caracteres. Ésta es la razón por la que se incluye la tabla ASCII de 8 bits, 4 bits de zona y 4 de dígito (tabla 3-3). Los primeros 128 caracteres son los originales. Los segundos 128 son los agregados y aquí se puede hablar de caracteres “no oficiales”, dado que las tablas no guardan uniformidad; en su mayoría, representan caracteres gráficos y no son esenciales. Algunos ejemplos son el símbolo del Yen japonés –D9h–, el franco –F9h–, el signo de interrogación de apertura que utiliza el idioma español –8Ah–, caracteres de contorno (doble línea, línea simple), caracteres de brillo o intensidad que se superponen en una pantalla sobre otros caracteres, etcétera.

Tabla 3-3. Tabla de código ASCII ampliado (de 8 bits).

	HEX. N°	0	1	2	3	4	5	6	7
HEX. N°	BITS 7654 3210	0000	0001	0010	0011	0100	0101	0110	0111
0	0000	NUL		SP	0	@	P	'	p
1	0001		DC1	!	1	A	Q	a	q
2	0010		DC2	-	2	B	R	b	r
3	0011		DC3	#	3	C	S	c	s
4	0100		DC4	\$	4	D	T	d	t
5	0101			%	5	E	U	e	u
6	0110			&	6	F	V	f	v
7	0111	BEL		'	7	G	W	g	w
8	1000	BS	CAN	(8	H	X	h	x
9	1001	HT	EM)	9	I	Y	i	y
A	1010	LF		*	:	J	Z	j	z
B	1011	VT	ESC	+	:	K	[k	{
C	1100	FF		,	<	L	\	l	:
D	1101	CR		-	=	M]	m	}
E	1110	SO			>	N	^	n	
F	1111	SI		/	?	O	-	o	DEL

	HEX. N°	8	9	A	B	C	D	E	F
HEX. N°	BITS 7654 3210	1000	1001	1010	1011	1100	1101	1110	1111
0	0000	ç	é	á	í	â	ð	a	=
1	0001	ü	æ	í	í	â	ñ	ß	±
2	0010	é	æ	ó	í	â	ò	g	=
3	0011	â	ó	ú	³	â	ó	p	=
4	0100	â	ó	ñ	'	ä	ö	s	f
5	0101	à	ò	ñ	µ	â	ö	s)
6	0110	â	û	á	¶	æ	ö		÷
7	0111	ç	ù	·	·	ç	×	t	-
8	1000	ê	ÿ	ë	·	ë	ø	f	º
9	1001	ë	ö	¬	í	é	û	ø	
A	1010	è	Ü	¬	º	é	ú	o	-
B	1011	í	¢	½	»	ë	í	d	✓
C	1100	î	£	¼	¼	í	—	8	n
D	1101	ì	¥	í	½	í	í	f	z
E	1110	Ã	þ	«	¾	í	í	e	¡
F	1111	À	f	»	¿	í	—	n	□

3.3.3 Delimitación de strings

Las cadenas de caracteres alfanuméricas (*strings*), procesadas y almacenadas en la computadora, pueden ser:

- De **longitud fija**: cada dato ocupa un número determinado de bytes fijo. Por lo tanto, el comienzo y el final de cada dato puede conocerse con rapidez.
- De **longitud variable**: para determinar el final de cada dato hay dos métodos:
 - Cada dato tiene en su inicio un campo en el que se indica la longitud en bytes, como lo muestra la figura 3.3.



Fig. 3.3. Longitud de bytes.

- Cada dato se separa de los colindantes mediante un símbolo específico o separador, que se utiliza para marcar el final de la cadena de caracteres, pero no se considera parte de ésta. Hay dos separadores muy usados:
- 1) El "byte 0" o NUL.
 - 2) El byte con un código identificado en la tabla ASCII como de "retorno de carro" (CR), que por lo general se utiliza para el final de una línea de texto, que en el teclado equivale a la tecla <ENTER>. En la figura 3.4 se observa el caso que se menciona.



Fig. 3.4. CR (carriage return).

Para dar un ejemplo, el *string* "PRIMER AÑO" se representa en ASCII de 8 bits (en hexadecimal) como:

P R I M E R A Ñ O CR
50 52 49 4D 45 52 20 41 A5 4F 0D



Los códigos BCD son convenciones que permiten la representación de números decimales (0 a 9) en bloques binarios de 4 bits.

3.4 Códigos de representación decimal (BCD)

Los **códigos BCD** son convenciones que permiten la representación de números decimales (0 a 9) en bloques binarios de 4 bits. Son códigos de representación de números y se los denomina códigos ponderados, porque adjudican cierto peso a los 1 binarios, según la posición que ocupan en el bloque, por lo que se debe verificar que la suma de los pesos de cada combinación sea igual al número decimal representado.

Aquí se mencionarán tres clases de códigos BCD, el BCD puro y dos derivaciones de él, cuya importancia radica en su capacidad para ser códigos autocomplementarios; ellos son el BCD exceso tres y el BCD 2421.

3.4.1 BCD puro o natural

Se denomina así porque los pesos en cada bloque coinciden con el valor de los 4 primeros pesos del sistema binario puro. Ésta es la razón por la que al BCD puro se lo suele llamar BCD 8421. En la tabla 3-4 se encuentran las equivalencias entre decimal y BCD puro.

Tabla 3-4. BCD puro.

Valor decimal	BCD puro 8421	Valor decimal	BCD puro 8421
0	0000	5	0101
1	0001	6	0110
2	0010	7	0111
3	0011	8	1000
4	0100	9	1001

Si, por ejemplo, se quiere expresar el número $5_{(10)}$ en BCD puro u 8421 , su equivalente es $0101_{(BCD)}$; esto es correcto, ya que si se suman los pesos correspondientes a los 1 contenidos dentro del número 0101 ($4 + 1$) se obtiene el valor 5.

Algunos ejemplos permitirán que se observen las diferencias de representación de un número en los distintos sistemas

Valor decimal	Binario puro	BCD puro
$15_{(10)}$	$1111_{(2)}$	$0001\ 0101_{(BCD)}$
$256_{(10)}$	$2^8 = 100000000_{(2)}$	$0010\ 0101\ 0110_{(BCD)}$
$1_{(10)}$	$1_{(2)}$	$0001_{(BCD)}$

Nótese que al eliminar los dígitos de zona de "un número" en el código ASCII (zona 3), se obtiene la combinación de bits que representa el dígito BCD correspondiente.

3.4.1.1 BCD empaquetado

La misma ALU que opera en binario puro puede operar en BCD. Lo que cambia es la forma en que se producen las distintas etapas para llevar a cabo la operación. Estas etapas están definidas por instrucciones en el programa, que permiten la gestión de la operación, considerando como base del sistema la decimal y no la binaria.

Los dispositivos electrónicos capaces de realizar la operatoria aritmética permiten almacenar los datos decimales en una forma codificada en sistema binario. Como ya se indicó, los datos del exterior suelen introducirse en la computadora en códigos alfanuméricicos con formato de caracteres. En ASCII los números se representan con octetos divididos en dos partes: zona y dígito. Los números así representados se denominan "**zoneados**" (*unpacked*). Si estos datos numéricos se operaran en la ALU como fueron ingresados, los bits de zona ocuparían, en principio, un lugar innecesario y provocarían resultados incorrectos al operarlos aritméticamente, lo que complicaría el cálculo.

En la tabla de código alfanumérico la zona distingue, por ejemplo, números de letras. A una cadena de caracteres que va a tratarse como un número en la ALU "le sobra la zona", por lo tanto, una secuencia de instrucciones en el programa la elimina.

El producto final al eliminar la zona y agrupar los dígitos en forma reducida se conoce como "empaquetado" y cada dígito queda representado empleando sólo 4 bits de "dígito", definido en el byte que representa el carácter numérico.

Un grupo de dígitos BCD constituye un número u operando empaquetado; el signo se incluye en la cabecera o la cola del número, por lo general, en la parte sobrante del octeto que representa el último octeto. Los operandos empaquetados ocupan un número entero de octetos y, por lo tanto, un operando signado, con cantidad par de dígitos decimales, deberá rellenarse con un cero en la posición más significativa.

Este tipo de convenio permite el tratamiento aritmético considerando la base decimal, y si bien permite una conversión simple y rápida (eliminar la zona y considerar el signo), la capacidad de la ALU admite que sólo un grupo reducido de dígitos decimales pueda operarse en forma aritmética cada vez; por lo tanto, el tratamiento de dos operandos largos requiere varios accesos a la ALU (hipotéticamente, sumar con una calculadora de sólo dos dígitos dos operandos de 10, implica 5 sumas parciales con el control correspondiente de acarreos).

Un número desempaquetado ocupa un byte, en el que los bits de orden superior son los correspondientes a la zona del código alfanumérico, y se utiliza para el ingreso o para que pueda "mostrarse" al usuario.

+3456	<i>desempaquetado:</i>	2B 33 34 35 36	
	<i>empaquetado:</i>	03456B	Signo + (B)
+279	<i>desempaquetado:</i>	2B 32 37 39	
	<i>empaquetado:</i>	279B	
-3456	<i>desempaquetado:</i>	2D 33 34 35 36	
	<i>empaquetado:</i>	03456D	Signo - (D)
-279	<i>desempaquetado:</i>	2D F2 F7 F9	
	<i>empaquetado:</i>	279D	

Las operaciones "empaquetar" y "desempaquetar" se indican en forma explícita en el programa con instrucciones que las definan.

3.4.2 BCD exceso tres

Se obtiene a partir del BCD puro, sumando un 3 binario a cada cifra decimal. Así se logra un código simétrico o autocomplementario, que facilita la operación de hallar el complemento de un número (complemento restringido decimal, complemento lógico o negación) sólo con invertir sus dígitos.

En la figura 3.5 se muestran las equivalencias entre decimal y BCD exceso tres. Obsérvese que 0111 (4) es el complemento restringido de 1000 (5), o que 0011 (0) es el complemento restringido de 1100 (9).

Valor decimal	BCD exceso tres	
0	0011	
1	0100	
2	0101	
3	0110	
4	0111	
5	1000	
6	1001	
7	1010	
8	1011	
9	1100	"Código autocomplementario"

Fig. 3.5. Código BCD exceso tres.

3.4.3 BCD AIKEN o 2421

Este código adjudica a cada dígito "1" binario el peso que resulta de la combinación 2421, en lugar del BCD puro que impone un peso igual a 8421. Por ejemplo, el número 7 en BCD 2421 es "1101" y surge de la suma de los dígitos "1" multiplicados por sus pesos, o sea, $2 + 4 + 1 = 7$.

También se usa en gran medida por su simetría, que permite hallar con facilidad el complemento de un número, o sea que cuando una operación BCD involucra el complemento restringido de un dígito, éste se obtiene invirtiendo los bits de la combinación de la misma forma que se hace en sistema binario. En la figura 3.6 se observan las equivalencias entre decimal y BCD 2421.

Valor decimal	BCD 2421
0	0000
1	0001
2	0010
3	0011
4	0100
5	1011
6	1100
7	1101
8	1110
9	1111

“Código autocomplementario”

Fig. 3-6. Código BCD Aiken.

Continuando con los ejemplos presentados, se pueden observar las diferencias de representación de un número en estos últimos dos sistemas.

Valor decimal	BCD exceso tres	BCD 2421
$15_{(10)}$	0100 1000 _(BCD EXC-3)	0001 1011 _(BCD 2421)
$256_{(10)}$	0101 1000 1001 _(BCD EXC-3)	0010 1011 1100 _(BCD 2421)
$1_{(10)}$	0100 _(BCD EXC-3)	0001 _(BCD 2421)

3.5 Códigos de representación numérica no decimal

Estos códigos permiten la operación aritmética de datos en sistema binario o en otras bases no decimales. Los operandos ingresan primero en código alfanumérico para luego convertirse a alguno de estos convenios. Si bien la conversión es más compleja que cuando se opera en sistema decimal, los datos que intervendrán en cálculos reducen su tamaño en grado considerable, con lo que se operan con mayor rapidez. A modo de ejemplo, se puede recordar cuántos dígitos ocupa el número $15_{(10)}$ en un convenio decimal, en contraposición al binario:

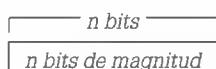
$$0001\ 0101_{(BCD)} \quad 1111_{(2)}$$

3.5.1 Coma o punto fijo sin signo (enteros positivos)

Es el formato más simple para representar números enteros positivos, expresados como una secuencia de dígitos binarios:

$$X_{n-1}, X_{n-2}, \dots, X_2, X_1, X_0$$

Como este formato sólo permite números sin signo, para la representación de un número se utiliza la totalidad de bits del formato.

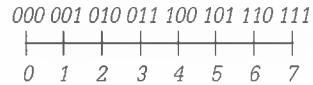


El rango de variabilidad para un número x en este formato es:

$$0 \leq x \leq 2^n - 1$$

donde "n" es la cantidad de dígitos que componen el número; también se la conoce como tipo de dato "ordinal sin signo".

Por ejemplo, para un formato de $n = 3$ bits, el rango de variabilidad estará comprendido entre 0 y 7. Expresado de otra manera, permite representar los números comprendidos en ese rango, como se observa en la recta de representación siguiente:



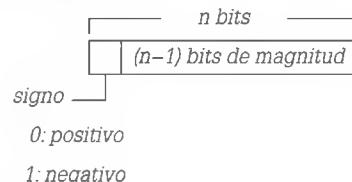
Por lo tanto, si se suman dos números cuyo resultado supere el rango de variabilidad establecido, este resultado será erróneo, ya que perderá el dígito más significativo. El ejemplo siguiente muestra esta condición para $n = 3$:

Comprobación	
$ \begin{array}{r} 101 \\ + 111 \\ \hline 1100_{(2)} \end{array} $	$ \begin{array}{r} 5 \\ + 7 \\ \hline 12_{(10)} \end{array} $

Las unidades de cálculo en una CPU actualizan "señalizadores", llamados *flags* o banderas, que indican, entre otras, esta circunstancia de desborde u *overflow* que, para este ejemplo en particular, implica la pérdida del bit de orden superior.

3.5.2 Coma o punto fijo con signo (enteros)

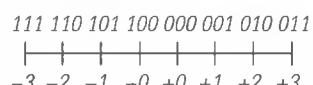
Es igual al formato anterior, pero reserva el bit de extremo izquierda para el signo, de manera que si ese bit es igual a 0 indicará que el número es positivo; asimismo, si, por el contrario, es igual a 1 indicará que el número es negativo. A este tipo de representación también se la llama "magnitud con signo".



El rango de variabilidad para los binarios puros con signo es:

$$-(2^{n-1} - 1) \leq X \leq + (2^{n-1} - 1)$$

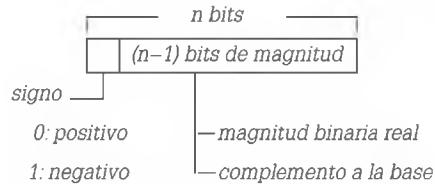
Para el ejemplo de $n = 3$ bits, el rango de variabilidad estará comprendido entre -3 y $+3$, según la recta de representación siguiente:



Nótese la presencia de un cero positivo y otro negativo. Este último surge del cambio de signo, 0 por 1, para esa misma magnitud.

3.5.3 Coma o punto fijo con signo con negativos complementados a “2” (enteros)

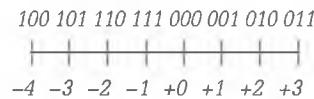
Es igual al formato anterior, reservando el bit de extremo izquierda para el signo, de manera que si ese bit es igual a 0, indicará que el número es positivo; por el contrario, si ese bit es igual a 1, indicará que el número es negativo. La diferencia radica en que los $n - 1$ bits restantes se representan con su magnitud binaria real para el caso de los positivos y en el complemento a la base de la magnitud para el caso de los negativos.



El rango de variabilidad para los binarios con signo en complemento a la base es:

$$-(2^{n-1}) \leq X \leq + (2^{n-1} - 1)$$

Para $n = 3$ bits, el rango de variabilidad estará comprendido entre -4 y $+3$, como se muestra en la recta de representación siguiente:



La representación de los números negativos surge de aplicar la definición para hallar el complemento a la base de un número; por lo tanto, el complemento del número $+2$ con un formato de 3 bits es:

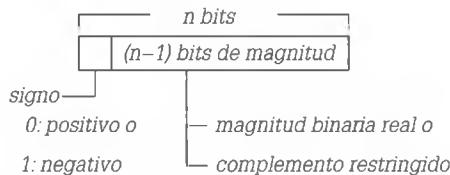
$$\begin{array}{r} 1000 \\ - 010 \\ \hline 110 \end{array}$$

De forma similar, el complemento del número $+3$ es:

$$\begin{array}{r} 1000 \\ + 011 \\ \hline 101 \end{array}$$

3.5.4 Coma o punto fijo con signo con negativos complementados a “1” (enteros)

Es igual al formato anterior, reservando el bit de extrema izquierda para el signo, de manera que si ese bit es igual a 0, indicará que el número es positivo; por el contrario, si ese bit es igual a 1, indicará que el número es negativo. No obstante, los $n - 1$ bits restantes se representan con su magnitud binaria real para el caso de los positivos y en complemento restringido de la magnitud para el caso de los negativos.

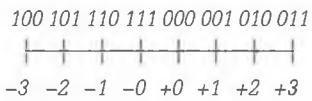


El rango de variabilidad para los binarios puros con signo en complemento restringido o complemento a 1, es:

$$-(2^{n-1} - 1) \leq X \leq + (2^{n-1} - 1)$$

En esta codificación también hay dos “ceros”, uno positivo y otro negativo.

Luego, para $n = 3$ la recta representativa del rango de variabilidad será:



3.5.5 Reales en coma o punto flotante (números muy grandes y números reales)

Los números analizados hasta ahora se pueden clasificar como enteros sin signo y enteros signados, en general limitados por un formato de 8, 16, 32 o 64 bits; sin embargo, a veces es necesario operar datos mayores (por ejemplo, para programas que resuelvan cálculos científicos). La primera solución para tratar números muy grandes es ampliar el formato a tres, cuatro u ocho octetos y obtener así valores decimales que oscilan entre alrededor de los 8 millones y los 9 trillones (considerando enteros signados). De todas maneras, en los cálculos no siempre intervienen valores enteros y es entonces donde aparece el inconveniente de cómo representar el punto para manejar cantidades fraccionarias. Por un lado, el punto debería expresarse con un dígito "0", o bien, con un dígito "1", dentro de la cadena de bits que representa al número binario; por otro lado, el punto no asume una posición fija sino variable dentro de la cadena, o sea que si el punto se representa con un bit "1" no se podrá diferenciar en una secuencia "1111" el binario 1111 del binario 111,1, o bien del binario 1,111. La solución para este problema es eliminar el punto de la secuencia e indicar de alguna manera el lugar que ocupa dentro de ella. De esta forma, los números se representan considerando dos partes. La primera indica los bits que representan al número y se llama **mantisa**, la otra indica la posición del punto dentro del número y se llama **exponente**. Esta manera de tratar números fraccionarios tiene cierta semejanza con lo que se conoce en matemática como notación científica o exponential. Veamos cómo se puede representar un número decimal:

$$3,5_{(10)} = \begin{cases} 35 \cdot 10^{-1} & \text{o bien, es igual a} \\ 0,35 \cdot 10^{+1} & \end{cases}$$

En el primer caso, 35 es un número entero que, multiplicado por la base 10 y elevado a la -1 , expresa el mismo número inicial sin representar la coma dentro de él. Así, "35" es la mantisa y " -1 ", el exponente, que se puede leer como "el punto está **un** lugar a la **izquierda** del dígito menos significativo de la mantisa".

En el segundo caso, 0,35 es un número fraccionario puro (se indica entonces que es una fracción **normalizada**, dado que se supone que el punto está a la izquierda del dígito más significativo y distinto de cero de la mantisa). Luego, 0,35 multiplicado por la base 10 y elevado a la $+1$ expresa el mismo número inicial, sin representar la coma dentro de él. En este caso, " $+1$ " indica que el punto está **un** lugar a la **derecha** del dígito más significativo de la mantisa.

Cuando se trata de números binarios con punto fraccionario se utilizan, entonces, dos entidades numéricas, una para la mantisa (por lo general una fracción normalizada y con signo) y otra para el exponente. La base del sistema es siempre conocida, por lo tanto, no aparece representada. Este nuevo formato de números se conoce como binarios de punto (coma) flotante, ya que el punto varía su posición ("flota") según el contenido binario de la entidad exponente.

La fórmula

$$\pm M \cdot B^{\pm p}$$

representa el criterio utilizado para todo sistema posicional, donde "M" es la mantisa, que podrá tratarse como **fracción pura** o como **entero**, según se **suponga** la coma a **izquierda** o a **derecha** de "M". "B" es la base del sistema; "P" es el exponente que indica la posición del punto en "M" y los signos "+" y "-" que la anteceden indican su desplazamiento a dere-

cha o izquierda respecto del **origen**. En la memoria sólo se almacenarán la mantisa "M" y la potencia "P" en binario. La base "B" y el **origen** del punto siempre son determinados con anterioridad por el convenio y conocidos por los programas que utilizan esta representación.

La representación de datos en punto flotante incrementa el rango de números más allá de los límites físicos de los registros. Por ejemplo, se sabe que en un registro de 16 bits el positivo mayor definido como entero signado que se puede representar es el +32767. Si se considera el registro dividido en partes iguales, 8 bits para la mantisa entera y 8 bits para el exponente, el entero mayor se calcula según la fórmula siguiente:

$$+(2^7 - 1) \cdot 2^{+(2^7 - 1)} = +127 \cdot 2^{+127}$$

que implica que al binario representativo del entero 127 se le agregan 127 ceros a derecha; de modo que enteros muy grandes encuentran su representación en esta modalidad.

Si ahora se representa el valor $3,5_{(10)}$ en binario 11,1 resulta:

$$+3,5_{(10)} = +11,1_{(2)} \text{ expresado en notación científica es igual a}$$

$$+111,0_{(2)} \cdot 10_{(2)}^{-1(2)} = +7_{(10)} \cdot 2_{(10)}^{-1(10)} = +7/2 = +3,5_{(10)} \text{ mantisa entera}$$

$$+0,111_{(2)} \cdot 10_{(2)}^{+10(2)} = +0,875_{(10)} \cdot 2_{(10)}^{+2(10)} = +3,5_{(10)} \text{ mantisa fraccionaria}$$

Dado que los registros que operen estos datos tendrán una longitud fija, se asume un formato de "m" bits para la mantisa y "p" bits para el exponente, entonces se puede definir el rango de representación de números reales, que es siempre finito. O sea que determinados reales son "representables" en el formato y otros quedan "afuera" de él.

Cuando el resultado de una operación supera los límites mayores definidos por el rango, entonces es incorrecto. El error se conoce como **overflow de resultado** y actualiza un bit (*flag de overflow*) en un registro asociado a la ALU, conocido como registro de estado o *status register*.

Cuando el resultado de una operación cae en el hueco definido por los límites inferiores del rango (del que se excluye el cero), esto es

$$0 > x > 0,1 \quad o \quad 0 < x < 0,1$$

es incorrecto. El error se conoce como *underflow* de resultado.

3.5.5.1 Convenios de representación en punto flotante con exponente en exceso

Para eliminar el signo del exponente se utilizan convenios que agregan al exponente un exceso igual a 2^{p-1} . Esta entidad nueva se denomina característica y es igual a:

$$\begin{array}{c} \text{CARACTERÍSTICA} = \boxed{\pm P} + \boxed{2^{p-1}} \\ \text{EXPONENTE} \qquad \qquad \qquad \text{EXCESO} \end{array}$$

Siguiendo con el ejemplo de $+3,5_{(10)}$ pero en este caso en un formato de $m = 8$ (cantidad de bits de la mantisa, incluido el signo) y $p = 8$ (cantidad de bits de la potencia), el exceso será:

$$2^{p-1} = 2^7 = 128 \quad \text{donde "p" es la cantidad de bits de la característica.}$$

En este convenio la mantisa se representa en la forma de signo y magnitud, o sea, **no se utiliza el complemento para los negativos**. Ahora bien, la forma en que la computadora "ve" la mantisa puede ser entera o fraccionaria y sin normalizar o normalizada. Analicemos las diferentes formas que puede adoptar la mantisa y su manera de representación, siempre para el mismo ejemplo.



Overflow: es un error que se produce cuando el resultado se encuentra fuera de los límites superiores del rango.

Para una mantisa **entera** será:

Formato alojado en una locación de memoria

$$3,5_{(10)} = 11,1_{(2)} = 111 \cdot 10^{-1}$$

$$\text{Característica} = 128 - 1 = 127$$

exc. exp.

CARÁCTER	S	MANTISA
01111111	0	0000111.

↓
+ (positivo)

Si la mantisa fuese **fraccionaria** la forma de representarla sería:

$$3,5_{(10)} = 11,1_{(2)} = .0000111 \cdot 10^{+10}$$

$$\text{Característica} = 128 + 6 = 134$$

CARÁCTER	S	MANTISA
10000110	0	.0000111

En general, la manera de representar la mantisa es normalizada; por lo tanto, si se necesita expresar el número anterior con mantisa **fraccionaria y normalizada**, éste será:

$$3,5_{(10)} = 11,1_{(2)} = .111 \cdot 10^{+10}$$

$$\text{Característica} = 128 + 2 = 130$$

CARÁCTER	S	MANTISA
10000010	0	.1110000

En la tabla 3-5 se muestran las correspondencias entre el exponente y la característica.

Tabla 3-5. Correspondencias entre el exponente y la característica.

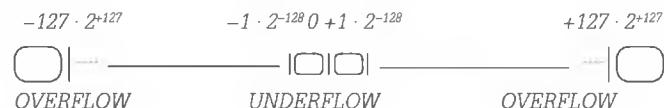
Si el exponente es	la característica será
-128	0
-1	127
0	128
127	255

Analicemos la tabla 3-6, en la que se observan los valores máximos y mínimos que se pueden representar con $m = 8$ y $p = 8$, con una **mantisa entera sin normalizar** (excedida en 128).

Tabla 3-6. Valores máximos y mínimos representados con $m=8$ y $p=8$

	Mantisa	Carácter	Mant.	Poten.	Resulta de
- el mayor nro. positivo es:	0 1111111	11111111	+127	+127	$255 - 128 = +127$
- el menor nro. positivo es:	0 0000001	00000000	+1	-128	$0 - 128 = -128$
- el menor nro. negativo es:	1 0000001	00000000	-1	-128	$0 - 128 = -128$
- el mayor nro. negativo es:	1 1111111	11111111	-127	+127	$255 - 128 = +127$

Los valores máximos y mínimos de la tabla se ven reflejados en la recta de representación siguiente:



Genéricamente:

- El número mayor positivo queda definido como $+(2^{m-1}-1) \cdot 2^{\frac{p(p-1)}{2}}$
- El número mayor negativo queda definido como $-(2^{m-1}-1) \cdot 2^{\frac{p(p-1)}{2}}$
- El número menor positivo queda definido como $+2^{\frac{p(p-1)}{2}}$
- El número menor negativo queda definido como $-2^{\frac{p(p-1)}{2}}$

En resumen, los valores máximos y mínimos que se pueden representar con una mantisa entera sin normalizar son los que se visualizan en la tabla 3-7

Tabla 3-7. Límites de overflow y underflow.

$$\pm(2^{m-1}-1) \cdot 2^{2^{p-1}-1} \quad \pm1 \cdot 2^{\lceil 2^{p-1} \rceil}$$

En los rangos de variabilidad la recta no es siempre continua, porque a medida que el exponente "flota" fuera del límite del registro se "agregan ceros" a derecha o a izquierda de la mantisa y, por lo tanto, se expresan valores enteros o fracciones puras muy pequeñas, por lo que quedan sin representación los valores intermedios

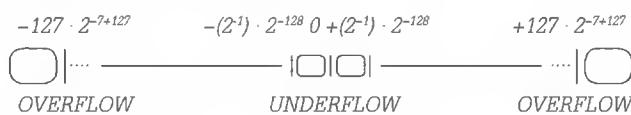
Con una **mantisa normalizada fraccionaria** donde $m = 8$ y $p = 8$ (excedida en 128), los valores máximos y mínimos que se pueden llegar a representar son los que se muestran en la tabla 3-8. Recuerde que 127 es el valor correspondiente a una mantisa entera, entonces, para calcular el valor 0,1111111 se lo debe multiplicar por 2^{-7} , lo que provoca el desplazamiento del punto de extrema derecha a extrema izquierda:

$$+127 \cdot 2^{-7} = \frac{+127}{+128} = 0.99218..$$

Tabla 3-8. Valores máximos y mínimos.

	Mantisa	Carácter	Mant.	Poten.
- el mayor nro. positivo es:	0 1111111	11111111	$127 \cdot 2^{-7}$	+127
- el menor nro. positivo es:	0 1000000	00000000	$+.1$	-128
- el menor nro. negativo es:	1 1000000	00000000	$-.1$	-128
- el mayor nro. negativo es:	1 1111111	11111111	$-127 \cdot 2^{-7}$	+127

Los valores máximos y mínimos de la tabla, para mantisa fraccionaria normalizada, se reflejan en la recta de representación siguiente:



Por último, los valores máximos y mínimos que se pueden representar con una mantisa fraccionaria normalizada son los que se visualizan en la tabla 3-9.

Tabla 3-9. Límites de overflow y underflow.

$$\pm(2^{m-1}-1) \cdot 2^{(m-1)} \cdot 2^{2^{p-1}-1} \quad \pm(2^{-1}) \cdot 2^{\lceil 2^{p-1} \rceil}$$

Los sistemas de representación de datos en punto flotante analizados hasta ahora nos permiten introducirnos en el estudio de otros, cuya comprensión hubiese resultado confusa sin este primer acercamiento.

3.5.5.2. Punto flotante "exceso 127"

En esta convención el número se representa en formato de 4 octetos:

1 bit 8 bits m = 23 bits mantisa

S	$\pm p + 127$	
---	---------------	--

1. (posición supuesta del bit implícito)

El cálculo de la característica o el exponente responde al criterio empleado en la convención anterior:

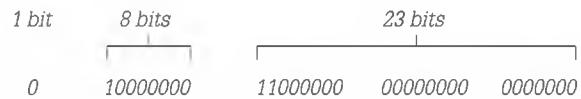
$$\pm P + (2^{8-1} - 1) = \pm P + (2^7 - 1) = \pm P + 127_{(10)}$$

El primer bit corresponde al signo de la mantisa (0 positiva, 1 negativa) y en los 31 bits restantes se representa la característica y su magnitud binaria real. El lugar "supuesto" dado que no se representa para su almacenamiento, es la posición entre la característica y los bits de mantisa. Como la mantisa es una fracción pura normalizada, el primer bit luego de la coma debe ser 1. Por esta razón este "1" tampoco se almacena y "se supone" antes de la coma. Por lo tanto, la mantisa está constituida por los bits a la derecha de ella hasta llegar a 23.

$1.b_2b_1b_0\dots\dots\dots b_3b_2b_1b_0$ [23 bits de mantisa]

Si ahora representamos $+3,5_{(10)} = +11,1_{(2)} = +1,11_{(2)} \cdot 10_{(2)}^{+10_{(2)}}$ calculamos la característica como $(+1) + 01111111 = 10000000$

El signo es positivo, por lo tanto, el formato queda:



El seguimiento de otros ejemplos facilitará su entendimiento:

$$+4,0_{(10)} = +100_{(2)} = 1,0_{(2)} \cdot 10_{(2)}^{+10_{(2)}}$$

0 10000000 00000000 00000000 00000000

El exponente $+10_{(2)}$ significa "desplazar la coma a la derecha dos lugares".

$$+16,0_{(10)} = +1000_{(2)} = 1,0_{(2)} \cdot 10_{(2)}^{+10_{(2)}}$$

0 10000011 00000000 00000000 00000000

El exponente $+100$ significa "desplazar la coma a la derecha cuatro lugares"

$$-1,0_{(10)} = -1_{(2)} = -1,0_{(2)} \cdot 10_{(2)}^{+100_{(2)}}$$

1 01111111 00000000 00000000 00000000

El exponente 0 significa "no desplazar la coma"

$$-0,5_{(10)} = -0,1_{(2)} = -0,1_{(2)} \cdot 10_{(2)}^{+0_{(2)}}$$

1 01111110 00000000 00000000 00000000

El exponente -1 significa "desplazar la coma a la izquierda un lugar".

$$+0,125_{(10)} = +0,001_{(2)} = 1,0_{(2)} \cdot 10_{(2)}^{-1_{(2)}}$$

El exponente -11 significa "desplazar la coma a la izquierda tres lugares".

Estos dos últimos ejemplos representan casos de números menores que cero, por lo tanto, la coma sufre un “desplazamiento a izquierda”, que se representa en convenio con el signo del exponente negativo y la característica menor que 127.

De la observación de los últimos cinco ejemplos se pueden sacar las conclusiones siguientes:

- Los números negativos sólo se diferencian de los positivos por el valor del bit de signo asociado a la mantisa.
- Toda potencia de 2 se representa con mantisa $1,00_{(2)}$ y toma su valor según el desplazamiento del punto.
- Con una misma mantisa y distintas características se representan diferentes valores decimales.
- El punto puede “flotar” aun fuera del límite físico que impone el formato.

Pasos para la representación de un número en este formato

1. Convertir el número decimal a binario.
2. Normalizar, eliminar el bit denominado “implícito” y calcular el valor de “P” para determinar el desplazamiento del punto.
3. Calcular y representar la característica.
4. Acomodar el signo y la mantisa, considerando sólo los dígitos fraccionarios hasta completar el formato.

Ahora considérese el ejemplo siguiente:

$$-0,2_{(10)} = -0,\overline{0011}_{(2)} = -0,1100110011\ldots_{(2)} \cdot 10_{(2)}^{-11_{(2)}} = -1,\overline{001} \cdot 10_{(2)}^{-11_{(2)}}$$

Esta conversión no es exacta, o sea que la fracción binaria hallada es periódica y, por lo tanto, se aproxima a la fracción decimal original.

El error varía en relación con la cantidad de períodos que se consideren. Cuando se normaliza, los ceros a derecha del punto desaparecen, se indica su existencia con el valor del exponente $-11_{(2)}$ (tres lugares a izquierda del origen de la coma). El número representado queda así:

$$\begin{array}{c} 1\ 01111100\ 10011001\ 10011001\ 1001100 \text{ truncado} \\ \qquad\qquad\qquad 110011001\ldots \end{array}$$

En la mantisa se repite el período tantas veces como el formato lo permite hasta que se trunca.

Con este último ejemplo se pretende que quede claro que si la cantidad de bits de la mantisa aumenta, mejor será la precisión del número.

A continuación, se presenta un ejemplo que servirá de ejercitación. Representemos en punto flotante convenio exceso -127 los números $-5001,25_{(10)}$ y $+10,1_{(10)}$

$$\begin{aligned} -5001,25_{(10)} &= -1001110001001,01 = -1,00111000100101_{(2)} \cdot 10_{(2)}^{+1100} \\ &\quad 1\ 11110011\ 00111000100101000000000 \\ &\quad \boxed{-\ 115 \ \text{bit implícito}} \end{aligned}$$

Precisión simple y precisión doble

En esta convención la cantidad de bits que representan el número admite dos variantes:

- Formato de precisión simple: cuatro octetos.
- Formato de precisión doble: ocho octetos.

Los ejemplos aportados pertenecen al formato de precisión simple. La determinación de los límites sobre la recta real nos permite observar la mejora en la precisión del formato mayor. Queda a criterio del programador cuándo utilizar una u otra, al momento de definir en sus programas variables reales; se debe tener en cuenta que una mejora en la precisión implica el almacenamiento de mayor cantidad de bits, y esto tendrá desventajas que deberán evaluarse.

3.6 Representaciones redundantes

3.6.1 Códigos de detección y/o corrección de errores. Introducción

El cambio de un bit en el almacenamiento o la manipulación de datos origina resultados erróneos. Para detectar e incluso corregir estas alteraciones se usan códigos que agregan "bits redundantes". Por ejemplo, los bits de paridad se agregan al dato en el momento de su envío y son corroborados en el momento de su recepción por algún componente de la computadora, y lo mismo sucede al revés. De todas formas, la cantidad de errores que puedan producirse son mensurables probabilísticamente. El resultado de "medir" la cantidad de "ruido" que puede afectar la información se representa con un coeficiente conocido como **tasa de error**. De acuerdo con el valor de la tasa de error, se selecciona un código, entre los distintos desarrollados, para descubrir e incluso corregir los errores. Estos códigos reciben el nombre de códigos de paridad.

3.6.2 Paridad vertical simple o a nivel carácter

Al final de cada carácter se incluye un bit, de manera que la suma de "unos" del carácter completo sea par (paridad par) o impar (paridad impar). Este tipo de codificación se denomina paridad vertical. Suele acompañar la transmisión de octetos en los periféricos y la memoria.

Los ejemplos muestran el bit de paridad que se agrega al octeto, en el caso de usar paridad par o impar:

00110010 1	<i>se agrega un uno que permita obtener paridad par</i>
01001011 0	<i>se agrega un cero para lograr paridad par</i>

Este código de detección de errores sólo se utiliza si la tasa de error en la cadena de bits que se han de transmitir no es superior a uno; esto es, si hay dos bits erróneos no permite detectarlos.

3.6.3 Paridad horizontal a nivel de bloque

Por cada bloque de caracteres se crea un byte con bits de paridad. El bit 0 será el bit de paridad de los bits 0 del bloque; el bit 1, el de paridad de los bits 1 del bloque, y así sucesivamente.

3.6.4 Paridad entrelazada

Utilizando en conjunto el código de paridad vertical con el horizontal se construye el código de paridad entrelazada, que además de detectar errores permite corregirlos.



Bit de paridad: es un bit redundante agregado a una cadena de bits en la que se pretende detectar un posible error.

Supongamos una transmisión de cuatro caracteres en código ASCII de 7 bits, con paridad par. El primer grupo indica cómo deberían llegar los caracteres a su lugar de destino, de no haberse producido errores en la transmisión. El segundo grupo indica con un recuadro el bit que tuvo un error en la transmisión.

PH	PH
0 1 0 1 0	0 1 0 1 0
0 1 1 1 1	0 [0] 1 1 1 ——
1 1 0 1 1	1 1 0 1 1
1 0 1 1 1	1 0 1 1 1
1 1 1 0 1	1 1 1 0 1
1 0 1 1 1	1 0 1 1 1
0 1 1 0 0	0 1 1 0 0
<i>PV</i> – 0 1 1 1 1	<i>PV</i> – 0 1 1 1 1
<i>sin error</i>	<i>con error</i>

La forma de corregir el error es invertir el bit señalado como erróneo. Si hay más de un error en la misma fila o columna, quizás se pueda detectar, pero no se podrá determinar con exactitud cuál es el error. En la práctica este código no se considera óptimo, debido a la cantidad de bits de paridad que se deben agregar, por lo que suele utilizarse el código de Hamming.

3.6.5 Código de Hamming

Este código permite detectar y corregir los errores producidos en una transmisión con sólo agregar p bits de paridad, de forma que se cumpla la relación siguiente:

$$2^p \geq i + p + 1$$

donde i es la cantidad de dígitos binarios que se han de transmitir y p es la cantidad de bits de paridad.

Supongamos que se desean transmitir 4 dígitos binarios. Según el método de Hamming a éstos deberá agregarse la cantidad de bits de paridad necesarios para satisfacer la relación enunciada antes:

$$\begin{array}{ll} i = 4 & \text{para que la relación } 2^p \geq i + p + 1 \text{ se cumpla debe ser} \\ p = ? & p = 3, \text{ luego} \quad 2^3 \geq 4 + 3 + 1 \end{array}$$

De esta relación se deduce que la transmisión será de 4 bits de información más 3 de paridad par. Se debe tener en cuenta que con 2 bits de paridad se puede corroborar la transmisión de 1 dígito de información, con 3 de paridad se corroboran hasta 8 bits de información, con 4 de paridad se corroboran hasta 11 de información, y así sucesivamente.

En la cadena de 7 bits del ejemplo: b_1, b_2, \dots, b_7 son bits de paridad los que ocupan las posiciones equivalentes a las potencias de dos y el resto son bits de datos.

La tabla siguiente es la guía para determinar la distribución y el cálculo de los bits de paridad y su corroboración luego de la transmisión de hasta 11 dígitos binarios.

$p_1 = b_1$	$p_2 = b_2$	$p_3 = b_4$	$p_4 = b_8$
$p_1 = b_1$	*		
$p_2 = b_2$		*	
$i_1 = b_3$	*	*	
$p_3 = b_4$			*
$i_2 = b_5$		*	*
$i_3 = b_6$			*
$i_4 = b_7$	*	*	*
$p_4 = b_8$			*
$i_5 = b_9$		*	*
$i_6 = b_{10}$		*	*
$i_7 = b_{11}$	*	*	*
$i_8 = b_{12}$			*
$i_9 = b_{13}$	*		*
$i_{10} = b_{14}$		*	*
$i_{11} = b_{15}$	*	*	*

Por lo tanto, para corroborar la transmisión de cuatro bits de información, deberán verificarse las igualdades siguientes:

$$p_1 = b_1 + b_3 + b_5 + b_7$$

$$p_2 = b_2 + b_3 + b_6 + b_7$$

$$p_3 = b_4 + b_5 + b_6 + b_7$$

Si llegara a producirse algún error en la transmisión, (por ejemplo, se transmite el número binario 1101 y se recibe el 1100) el método de Hamming indicará el número del bit donde se produjo el error, mediante el proceso de verificación que realiza en el punto receptor. Veamos:

Se transmiten: $i_4 \ i_3 \ i_2 \ p_3 \ i_1 \ p_2 \ p_1$
 $b_7 \ b_6 \ b_5 \ b_4 \ b_3 \ b_2 \ b_1$
 $1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1$

Se reciben: 0 0 1 0 1 0 1

Verificación: $p_3 \ p_2 \ p_1$
 $1 \ 1 \ 1 = \text{el bit } 7_{(10)} \text{ es el erróneo.}$

Una vez detectado el error la corrección es sencilla, basta con invertir el bit erróneo.

3.7 Resumen

Como vimos, toda información interna se representa en forma binaria. En este capítulo se describieron en particular algunas de las representaciones de datos; veamos, entonces, algunos ejemplos de su utilización desde la perspectiva del lenguaje de programación y el diseño de hardware, considerando a los fabricantes de software y hardware más populares para ilustrar lo expuesto hasta ahora.

En el caso de números de coma flotante, Microsoft utiliza las versiones de *IEEE 754* en el formato de 4 bytes, 8 bytes y 10 bytes, definidas en lenguaje ensamblador con las directivas "DD", "DQ" y "DT", respectivamente. En relación con el hardware, podemos dar como ejemplo su origen en el coprocesador 8087 (ya fuera de uso) para los primeros procesadores Intel de la industria 80x86 y en todas las unidades de coma flotante incluidas en los procesadores actuales del mismo fabricante. Respecto de los lenguajes de programación, para dar un ejemplo de identificación de los tipos de datos podemos utilizar el "C", en el que una variable de coma flotante de 4 bytes se declara como *float*, la variable coma flotante de 8 bytes se declara como *double* y la variable de coma flotante de 10 bytes, como *long double*.

Algunos lenguajes de programación como C, C++ y Java soportan tipos de datos decimales. Este soporte también se encuentra en los procesadores IBM Power 6 y en el *mainframe* de IBM System Z10.

3.8 Ejercicios propuestos

1) La siguiente es una sentencia en lenguaje de alto nivel, $C = A - B$.

Indicar un código que la represente en el programa fuente y establecer cuántos octetos ocuparía en memoria.

2) Para un formato de 32 bits, indicar los rangos de variabilidad en decimal, para números representados en ASCII sin signo y en binario signado en complemento a 2, e indicar cómo se representa el número +25₁₀ en los dos sistemas.

3) Dado el número 7910₁₀, expresarlo en:

- a) Decimal codificado en binario.
- b) BCD exceso -3.
- c) BCD exceso -2421
- d) Binario natural.

4) Si D2 es la representación hexadecimal de un número, exprese cuál es ese número en sistema decimal, si se considera un entero:

- a) No signado.
- b) Signado.

5) Expresar en decimal los límites de representación para un formato de 16 bits, si éste contiene:

- a) Un entero no signado.
- b) Un entero signado.

6) Representar (-18)₁₀ y (-64)₁₀ con un formato de 16 bits, en punto fijo en complemento restringido y en punto fijo en complemento a la base.

7) Determinar el rango de variabilidad para un formato de punto fijo con negativos complementados a dos en:

- a) 6 bits.
- b) 32 bits.
- c) 64 bits.

8) Dados los siguientes números, expresarlos en punto flotante, convención exceso 127.

- a) +32₁₀
- b) +3.25₁₀
- c) (-0.4)₁₀
- d) (-10.1)₁₀
- e) +0.5₁₀

9) La siguiente representación hexadecimal determina una cadena binaria: 945A.

- a) Indicar el valor decimal que representa si la cadena se considera binario natural.
- b) Indicar qué valor decimal representa si la cadena se considera un binario signado en sus tres formas de representación.

- c) Suponiendo que la cadena es la representación de un número en punto flotante en exceso con mantisa entera, donde $m = 8$ y $p = 8$, indicar:
- ¿Cuál es el exceso utilizado?
 - ¿Cuál es el número decimal que representa en la forma $\pm M \cdot B^p$?
- 10) Para un formato que permita una mantisa fraccionaria con $m = 8$ y $p = 8$:
- a) Determine los límites de representación, a partir de los cuales se produce overflow y underflow de resultado.
 - b) ¿Cuál es el exceso que permite emplear este formato?
 - c) ¿Qué permite la utilización de una mantisa con mayor cantidad de bits?
 - d) ¿Cuál sería la consecuencia si "p" tuviera mayor cantidad de bits?
- 11) Determine los límites, a partir de los cuales se produciría overflow y underflow de resultado, en precisión simple y en precisión doble, para el formato conocido como "exceso-1 27".
- 12) Suponga que recibe el número $87_{(10)}$ en el buffer de la impresora para ser listado en representación ASCII y que la transmisión utiliza como método de control el código de Hamming:
- a) Desarrolle las funciones para determinar el cálculo de los bits de paridad.
 - b) Suponga que se producen errores en la transmisión en el bit número:
 - 10 para el dígito $8_{(10)}$
 - 13 para el dígito $7_{(10)}$
 - c) Exprese cómo se desarrolla la detección y la corrección de ambos errores.

3.9 Contenido de la página Web de apoyo

El material marcado con asterisco (*) sólo está disponible para docentes.



Resumen gráfico del capítulo

Simulación

Permite ingresar un texto y lo codifica en ASCII.

Autoevaluación

Video explicativo (02:44 minutos aprox.)

Audio explicativo (02:44 minutos aprox.)

Evaluaciones Propuestas*

Presentaciones*

4

Aritmética de la computadora

Contenido

4.1 Introducción	74
4.2 Aritmética binaria.....	74
4.3 Aritmética decimal	79
4.4 Resumen.....	84
4.5 Ejercicios propuestos.....	85
4.6 Contenido de la página Web de apoyo.....	86

Objetivos

- Formular los convenios para la representación de cantidades signadas.
- Determinar los rangos de representación en formato de n bits.
- Evaluar la validez de las operaciones aritméticas.



En la página Web de apoyo encontrará un breve comentario de la autora sobre este capítulo.

4.1 Introducción

El desarrollo de este capítulo permite el análisis de las operaciones básicas de la aritmética binaria, implementadas en la unidad de coma fija para números enteros. Las operaciones en BCD corresponden a la aritmética decimal o de "base diez codificada en binario", implementada en los microprocesadores en la unidad de coma flotante (aun cuando no se trata de la misma forma de operatoria) y en unidades específicas de coma flotante decimal en *mainframes*. Los números "decimales codificados en binario" (BCD o *Binary Coded Decimal*) se utilizan en aplicaciones donde deba considerarse importante un error por conversión; por ejemplo, si $0,2_{(10)}$ es igual en binario a $0,0011$ periódico en las cuatro cifras fraccionarias, significa que la conversión a este binario se aproxima pero nunca será igual al original en base 10. Es de destacar que la aritmética decimal es más compleja que la binaria y que la representación interna de los datos ocupa mucho mayor espacio de almacenamiento.

4.2 Aritmética binaria

Físicamente, una unidad aritmética es un circuito asociado con uno o más registros, que contienen los operandos en la operación. Este circuito tiene un diseño distinto de una computadora a otra, que depende fundamentalmente de las operaciones que realice. Algunas computadoras sólo tienen implementadas en hardware operaciones muy simples, como la suma, el desplazamiento y el complemento. Las operaciones más complejas, como la multiplicación o la radicación, se realizan por medio de la ejecución de un software, que logra la solución con operaciones simples implementadas en hardware. Estas rutinas constituyen lo que se denomina **implementación en software** de operaciones aritméticas, de modo que el programador no debe preocuparse por implementarlas cuando las necesite. Por ejemplo, dada una sentencia de un lenguaje de programación que defina una multiplicación, se llamará a la rutina correspondiente que puede resolverla por sumas sucesivas y desplazamientos. El proceso insumirá cierto tiempo de ejecución, por lo que se deduce que las implementaciones en software simplifican el diseño de hardware a expensas de mayor tiempo de ejecución.

Recordemos que una ALU puede operar en base 2 cuando los datos se definen como enteros (signados o no), o como reales, y en base 10 cuando se pueden definir como decimales codificados en binario (BCD).

En una computadora cada operando será transferido de una o varias locaciones de memoria a uno de los registros de la CPU, que tiene una capacidad fija y limitada. La ALU no reconoce el formato de los operandos, sino que ellos se establecen de acuerdo con convenciones definidas para cada tipo de dato en los programas que los utilizan y sólo estos últimos son los encargados de interpretarlos, resultando transparentes para los componentes físicos de la ALU.

Describiremos ahora los algoritmos para la resolución de los problemas aritméticos elementales que puede utilizar la ALU en la operatoria de datos binarios enteros, signados, en su representación de punto fijo, y para datos binarios reales, en su representación de punto flotante.

4.2.1 Representación de datos en punto fijo (binarios enteros)

Para un operando declarado como entero, la coma mantiene una posición fija a extrema derecha de su magnitud y los números se representan en un convenio denominado representación de datos en punto fijo. El punto no está presente en el registro donde se almacena el operando, sino que su presencia se supone manteniendo esta posición fija.

Los números positivos se representan en la forma de magnitud con signo y los negativos pueden representarse de tres maneras diferentes. En una misma computadora, sólo los negativos se tratarán de una de estas formas. Por ejemplo, sea el número $-3_{(10)}$ podemos representarlo en un formato de 8 bits así:

- | | |
|-----------------------------------|-----------|
| 1. en magnitud con signo | 1 0000011 |
| 2. en complemento a 1 (con signo) | 1 1111100 |
| 3. en complemento a 2 (con signo) | 1 1111101 |

4.2.2 Operaciones aritméticas con enteros signados

Considere que los registros que contendrán los operandos que han de intervenir en la operación almacenan 8 bits y los negativos se representan en su forma de complemento a 2. Consideremos los cuatro casos de sumas posibles:

1. $(+A) + (+B)$
2. $(+A) + (-B)$
3. $(-A) + (+B)$
4. $(-A) + (-B)$

En todos los casos se realizará una suma binaria entre los pares de bits de igual peso y, en ocasiones, estas sumas parciales afectarán el par de bits de la siguiente columna con un acarreo (*carry*) igual a 1, como se puede observar en los ejemplos que se presentan a continuación.

$$\begin{array}{cccc}
 (+A) + (+B) & (+A) + (-B) & (-A) + (+B) & (-A) + (-B) \\
 (+3) + (+3) = +6 & (+10) + (-7) = +3 & (-7) + (+1) = -6 & (-64) + (-32) = -96 \\
 00000011 & 00001010 & 11111001 & 11000000 \\
 + \underline{00000011} & + \underline{11111001} & + \underline{00000001} & + \underline{11100000} \\
 00000110 & 00000011 & 11111010 & 10100000
 \end{array}$$

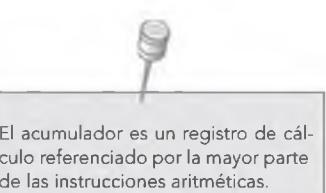
En este procedimiento hay algunos conceptos importantes que destacar:

- Los operandos negativos están complementados a la base y, por lo tanto, los resultados negativos que se obtienen están también complementados.
- El signo forma parte de la operación, sumándose como cualquier otro bit.
- En las operaciones se consideran todos los bits del formato, aun cuando sean 0 no significativos (a izquierda).

En el primer caso los operandos (ambos positivos) están en su forma de magnitud binaria real y signo. Se obtuvo el resultado esperado sin lugar a dudas.

En el segundo caso el primer operando es positivo y el segundo, negativo (complementado); se obtuvo el resultado esperado y el acarreo final se despreció.

En el tercer caso el primer operando es negativo (complementado) y el segundo es positivo; sin embargo, el resultado obtenido no aparenta ser -6 . La razón es que al ser negativo, el valor obtenido es su complemento a la base, en este caso el número de 7 bits obtiene su complemento de la base $2^7 = 128$. Si se convierte el valor obtenido a decimal se obtiene 122, que es precisamente lo que le falta a 6 para llegar a 128, por lo tanto, el resultado es correcto. La ALU entrega como resultado la secuencia de bits obtenida y es función del programa interpretar que cuando el signo es negativo el valor está enmascarado en su complemento. Si quiere realizar la verificación, simplemente calcule el complemento y obtendrá la magnitud buscada.



<i>s</i>	Magnitud complementada
1	1111010
<i>s</i>	Magnitud binaria real
1	0000110

En el cuarto caso ambos operandos son negativos y, por lo tanto, están complementados; sin embargo, puede parecer que el primero no lo está, eso ocurre porque la magnitud complementada coincide con la magnitud real. El complemento a la base de 64, en 7 bits, es lo que le falta a 64 para llegar a $2^7 = 128$, que es exactamente 64.

<i>s</i>	Magnitud binaria real
1	1000000
<i>s</i>	Magnitud complementada
1	1000000

Nótese que en este caso también hay acarreo final que se desprecia.

Las sumas vistas pueden efectuarse en un sumador con las características que se presentan en la figura 4.1.

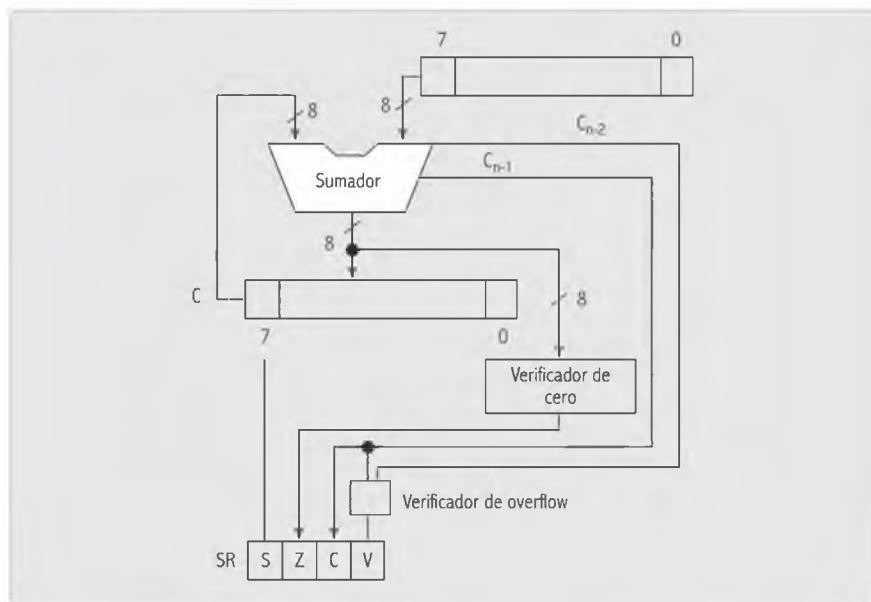


Fig. 4.1. Esquema de registros para suma de 8 bits.

El registro de estado almacena las banderas o flags aritméticas y otras que representan ciertas condiciones que pueden ser verdaderas o falsas.

El esquema presentado en la figura 4.1 es un bosquejo parcial de una unidad aritmética formado por:

- Un registro de 8 bits acumulador (AC), que almacenará el primer operando antes de la operación y el resultado después de ella.
- Un registro de 8 bits, que almacenará el segundo operando.

- Un registro de control del que consideraremos, sólo por ahora, 4 flags o banderas, llamado Registro de Estado o *Status Register* (SR).

Los flags son:

flag V: indica si hubo o no overflow o desborde de registro, después de haberse llevado a cabo la operación. Si almacena un valor "1" significa que hubo overflow; si almacena un valor "0" significa que no hubo overflow.

flag C: indica el acarreo o *carry* que se produce en la operación. Se pone en "1" si C_7 es igual a "1", o se pone en "0" si C_7 es igual a "0".

flag S: indica el signo del resultado de la operación. Se pone en "1" si el resultado es negativo, o se pone en "0" si el resultado es positivo.

flag Z: indica si el resultado de la operación es 0 o distinto de 0. Se pone en "1" si el resultado es 0, o se pone en "0" si el resultado es distinto de 0.

El acarreo o *carry* es el 1 producido cuando se suma 1+1 en base 2.

4.2.2.1 Overflow, sobreflujo o bit de desborde

Se dice que hay un sobreflujo cuando se excede la capacidad de un registro de almacenamiento, lo que provoca la invalidez del resultado obtenido. El overflow se verifica cuando, después de realizada una operación, se determina que el último acarreo es distinto del penúltimo.

Para un formato de 8 bits, el primero es para el signo y los 7 restantes para la magnitud; por lo tanto, el rango de variabilidad está comprendido entre -128 y +127. Para que se produzca overflow, bastaría con sumar +1 al operando +127. No obstante, observemos los casos extremos:

$\begin{array}{r} C_7 C_6 \\ \hline 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ \hline +127 & 01111111 \end{array}$	$\begin{array}{r} C_7 C_6 \\ \hline 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline -128 & 10000000 \end{array}$
$+$ $+127$ $+254$ <i>nueva magnitud</i>	$+$ -128 -256 <i>nueva magnitud</i>

La primera es la suma de los positivos más grandes que admite el formato. Siempre, cuando hay overflow de positivos, $C_7 = 0$ y $C_6 = 1$; el resultado correcto se logra si se considera que C_7 es signo del resultado y los ocho bits obtenidos de resultado son la magnitud binaria real. La segunda es la suma de los negativos más grandes que admite el formato. Cuando hay overflow de negativos, $C_7 = 1$ y $C_6 = 0$; el resultado correcto se logra si se considera que C_7 es signo del resultado y los ocho bits obtenidos son el complemento de la magnitud binaria real. En este caso es lógico analizar que si la combinación de 10000000 (8 bits) es -128, entonces la combinación de 100000000 (9 bits) corresponde a -256, que es la base siguiente.

Se debe destacar que el overflow sólo puede producirse con operandos del mismo signo, ya que en el caso contrario (un positivo y un negativo), el resultado obtenido siempre "entrará" en el formato utilizado (en realidad, uno se resta del otro).

Como ya dijimos, una ALU que tiene sólo un circuito sumador, como en este caso, realiza las restas binarias transformándolas en sumas de la siguiente manera:

$$\begin{aligned}
 (+A) - (+B) &= (+A) + (-B) \\
 (+A) - (-B) &= (+A) + (+B) \\
 (-A) - (+B) &= (-A) + (-B) \\
 (-A) - (-B) &= (-A) + (+B)
 \end{aligned}$$

O sea que, para efectuar una resta, la ALU debe recibir el complemento del segundo operando calculado previamente; así, un sustraendo positivo pasa a ser un operando negativo y un sustraendo negativo pasa a ser un operando positivo, quedando la resta implementada como una suma.

Una rutina software de resta debe ordenar las siguientes operaciones simples:

1. Invertir los bits del sustraendo (cálculo del complemento restringido).
2. Incrementarlos en una unidad (cálculo del complemento auténtico)
3. Sumarles el minuendo.

Delimitemos los pasos con el siguiente ejemplo:

$$\begin{array}{r}
 (+A) - (-B) \\
 (+10) - (-7) = +17 \\
 +10 = 00001010 \quad -7 = 11111001 \\
 \hline
 \begin{array}{l}
 \text{1.} \quad 00000110 \\
 + \\
 \text{2.} \quad 00000111 \\
 \hline
 \text{3.} \quad \underline{00000111} \\
 \hline
 00010001
 \end{array}
 \end{array}$$

4.2.3 Operaciones aritméticas en punto flotante

Habiendo dejado en claro que todas las operaciones para realizar en una ALU elemental serán transformadas en sumas, falta aclarar que, para sumar dos operandos en representación de punto flotante (RPFte), se debe tener en cuenta el alineamiento del punto, esto es, que ambos exponentes sean iguales. Si este procedimiento lo realiza el usuario escribiendo sobre un papel, alinear parte entera con parte entera y parte fraccionaria con parte fraccionaria es bastante sencillo, pero para el procesador no lo es.

Cuando un operando en punto flotante está desalineado respecto del otro, los exponentes son distintos, por lo tanto, la operación de suma se ve complicada con un paso previo. Una rutina software, semejante a la que se describe a continuación, servirá de apoyo para que el hardware de la ALU lleve a cabo la suma:

- Enviar a la ALU las entidades representativas de los exponentes de ambos operandos.
- Comparar los exponentes, por ejemplo, restando uno de otro. Si el resultado es igual a 0, significa que los exponentes son iguales, por lo tanto, las mantisas tienen su punto alineado. Cuando ambos exponentes son distintos, significa que se debe desplazar una de las mantisas para alinearla con la otra; el criterio que sigue el programa para decidir cuál de las dos se desplaza depende del convenio de punto flotante utilizado.
- Enviar a la ALU las mantisas para ser sumadas.

La unidad aritmética está capacitada para desplazar la información de algunos registros asociados con ella, de manera que se puede cargar en ellos la mantisa de uno de los operandos y, así, desplazarla los lugares necesarios hasta alinearla con la otra.

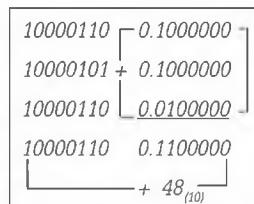
En los convenios en exceso las características de ambos operandos son binarias sin signo. Si se comparan, la menor característica indicará que la correspondiente mantisa se debe desplazar a la derecha y el valor que se obtiene del resultado de la comparación señalará cuántos lugares se debe

desplazar. El hecho de desplazar el operando que tiene menor característica responde a que un desplazamiento a derecha implica la pérdida de los dígitos menos significativos y, a lo sumo, el resultado se verá afectado por un redondeo. En caso contrario, un desplazamiento a izquierda representará la pérdida de los dígitos más significativos de la mantisa y, por ende, el resultado será erróneo.

A continuación, se presentan dos ejemplos para dar una "idea" de la operatoria de la ALU, dejando aclarado que las operaciones se realizan en binario signado, en punto flotante, con un exceso igual a 2^{p-1} , con mantisa fraccionaria normalizada, siendo $m = 8$ y $p = 8$.

$$1) \quad (+32)_{(10)} + (+16)_{(10)} =$$

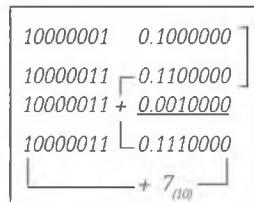
$$\begin{array}{r} 32_{(10)} = 00100000_{(2)} = 0.1000000 \cdot 10^{+10} \\ + 16_{(10)} = 00010000_{(2)} = 0.1000000 \cdot 10^{+10} \\ \hline 48_{(10)} \end{array}$$



En este ejemplo se muestra el desplazamiento de la mantisa, correspondiente al decimal 16, un lugar a la derecha para igualar las características.

$$2) \quad (+1)_{(10)} + (+6)_{(10)} =$$

$$\begin{array}{r} 1_{(10)} = 00000001_{(2)} = 0.1000000 \cdot 10^1 \\ + 6_{(10)} = 00000110_{(2)} = 0.1100000 \cdot 10^1 \\ \hline 7_{(10)} \end{array}$$



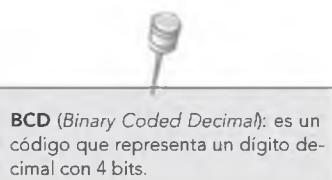
En este ejemplo el desplazamiento de la mantisa, correspondiente al decimal 1, es de dos lugares a la derecha para llegar a la misma característica.

La suma de dos mantisas normalizadas suele provocar un *overflow* de resultado y éste puede subsanarse con facilidad si se realiza un corrimiento del acarreo-signo-mantisa a derecha y se le suma un 1 a la característica. Si la suma no produjo *overflow*, pero el resultado se encuentra desnormalizado, se deberá efectuar un corrimiento a izquierda y restar del exponente el número representativo de los corrimientos efectuados. Si como consecuencia de los corrimientos, y a causa de representar la mantisa números muy grandes, se produjera sobreflujo en la característica, éste activará la bandera de *overflow* del SR.

4.3 Aritmética decimal

Como se indicó antes, una computadora digital puede realizar operaciones aritméticas en BCD (decimal codificado en binario). Los operandos pueden ser números con signo o sin él; la diferencia que hay entre ambos es que los números signados agregarán a su formato un dígito BCD, destinado a contener en el último bit el signo del operando. Es obvio que si se trabaja con un formato fijo se deberá tener en cuenta que la capacidad de almacenamiento se verá restringida a un dígito BCD menos, dedicado a contener el signo.

Desarrollaremos aquí el algoritmo utilizado por la rutina que se encarga de operar números BCD sin signo, dejando asentado que la operatoria para el manejo de los números BCD signados es similar a la de los binarios signados.



4.3.1 Operaciones con operandos BCD

4.3.1.1 Suma

Consideremos dos números decimales A y B , siendo a_i y b_i los dígitos respectivos. Si queremos efectuar la suma de los dígitos de a pares, debemos tener en cuenta, también, un posible acarreo C_{i+1} (carry), que a lo sumo tendrá valor "1"

$$\begin{array}{r} C_n \quad C_{n-1} \quad C_{i-1} \quad C_0 \\ A \quad \quad a_{n-1} \dots a_i \dots a_0 \\ + B \quad \quad b_{n-1} \dots b_i \dots b_0 \\ \hline S \quad C_n \quad S_{n-1} \dots S_{i-1} \dots S_0 \end{array}$$

La situación más desfavorable que se puede presentar es que

$$\begin{aligned} a_i &= 9_{(10)} = 1001_{(BCD)}, \\ \text{que } b_i &= 9_{(10)} = 1001_{(BCD)}, \\ \text{y que } C_{i+1} &= 1_{(10)} = 0001_{(BCD)}, \quad \text{cuyo resultado sería } S_i = 19_{(10)}. \end{aligned}$$

Tabla 4-1. Tabla de conversión

Decimal	Binario	BCD
0	0000	0000
1	0001	0001
2	0010	0010
3	0011	0011
4	0100	0100
5	0101	0101
6	0110	0110
7	0111	0111
8	1000	1000
9	1001	1001
10	1010	0001 0000
11	1011	0001 0001
12	1100	0001 0010
13	1101	0001 0011
14	1110	0001 0100
15	1111	0001 0101
16	10000	0001 0110
17	10001	0001 0111
18	10010	0001 1000
19	10011	0001 1001

Si observamos los valores de la tabla 4-1 y los consideramos posibles S_i , vemos que mientras no excedan $9_{(10)}$ la correspondencia binaria-BCD es idéntica.

¿Qué pasa entonces con los $S_i > 9_{(10)}$?

Nótese que la tabla binaria está desfasada de la tabla S_i BCD en 6 unidades. Por lo tanto, si $S_i > 9_{(10)}$ entonces varía entre 10 y 19, pues la suma máxima es $9 + 9 + 1$.

$$10 \leq S_i \leq 19,$$

$$\text{o cuando } S_i \leq 9 \text{ pero } C_i = 1$$

se suma 6 (codificado en BCD) para obtener el resultado correcto; además, se debe tener en cuenta que se genera un acarreo $C_i = 0001_{(BCD)}$ para la suma del siguiente par de dígitos de los operandos BCD.

Observemos la operación para la suma en BCD en los siguientes dos casos:

Decimal	BCD
123	0001 0010 0011
+ 456	+ 0100 0101 0110
579	0101 0111 1001
456	0100 0101 0110
+ 789	+ 0111 1000 1001
1245	1011 1101 1111
	+ 0110 0110 0110
	+ 0001 0001 0001 10101
	0001 10010 10100 5
	1 2 4

4.3.1.2 Resta

Caso en que el minuendo (M) > sustraendo (S)

Siendo la resta $M - S = R$ (resto o resultado), demostraremos que si sumamos el minuendo al complemento del sustraendo obtenemos el mismo resultado incrementado en 10^n , siendo n la cantidad de dígitos de M y de S :

$$M + C_S = R + 10^n$$

$$\text{Dada la resta} \quad 70 - 40 = 30$$

$$\text{entonces} \quad 70 + (10^2 - 40) = 30 + 10^2$$

$$70 + 60 = 130$$

eliminando 10^2 de R obtenemos 30, que es el resultado correcto.

$$\text{De la misma manera: } M + C'_S + 1 = R + 10^n$$

Caso en que el minuendo < sustraendo

Si el minuendo es menor que el sustraendo, el resultado obtenido es el "complemento de la diferencia real", luego:

$$M + C_S = C_{R(\text{resultado})}^B$$

$$\text{Dada la resta} \quad 20 - 30 = -10$$

$$20 + (10^2 - 30) = 10^2 - 10$$

$$20 + 70 = 90$$

y 90 es el complemento de la magnitud del resultado. Por lo tanto, 90 representa -10, que es la diferencia real.

$$\text{También: } M + C_S + 1 = C'_{R(\text{resultado})}.$$

Utilizando el complemento a la base o a la base -1, evitamos la implementación de un circuito que efectúe la resta dentro de la unidad aritmética, dado que se puede transformar en una suma.

El código BCD 8421 no es autocomplementario, ya que no obtenemos el complemento a 9 de un número (complemento restringido BCD) simplemente invirtiendo sus bits (procedimien-

to utilizado en el sistema de base 2, sino que debemos incluir una corrección. Para hallar el complemento de un dígito BCD, la rutina de tratamiento de la operación en curso, se puede utilizar el algoritmo que enunciamos a continuación

Algoritmo para el cálculo del complemento restringido de un dígito BCD natural:

Al dígito BCD se le suman 6 unidades (0110)

y al resultado se le invierten los bits.

Este algoritmo basa su lógica en considerar que la base de un operando BCD es 10, por lo tanto, el complemento restringido del mismo se obtendrá de la base -1, o sea, 9. Si al dígito BCD lo llamamos N se calculará como $9 - N$.

A su vez, un dígito BCD se representa en una computadora con cuatro dígitos binarios y, entonces, su complemento restringido se obtendrá de $1111_{(2)}$ [equivalente a $15_{(10)}$], o sea, $1111_{(2)} - N$, que es lo mismo que expresar $15_{(10)} - N$.

Para equiparar las dos relaciones enunciadas antes, $(9 - N)$ y $(15 - N)$, es necesario introducir una corrección al segundo enunciado, que se detalla a continuación:

$$9 - N = (15 - 6) - N, \text{ operando podemos indicar que}$$

$$9 - N = 15 - (6 + N).$$

La última ecuación define el algoritmo enunciado precedentemente. Como ya se demostró, esta resta equivale al complemento binario de $6 + N$, razón por la cual una vez que ésta se llevó a cabo, se invierten los bits resultantes.

Veamos la operatoria utilizada para la resta:

Decimal	BCD 8421	Regla de resta
456	0100 0101 0110	$0 - 0 = 0$
- 123	- 0001 0010 0011	$0 - 1 = 1$ con acarreo
333	001100110011	$1 - 0 = 1$
		$1 - 1 = 0$

Proponemos:

999	456	suma del minuendo más
- 123	+ 876	el complemento del sustraendo
	876	1 332
	+ 1	más 1 para llegar a la base por ser
		complemento restringido
		se desprecia
		el exceso de 10^n

Según el algoritmo, se le suma 6 (0110) a cada dígito y luego se invierten:

$C'0001 =$	$0001 + 0110 = 0111$	$\boxed{\begin{array}{l} \text{suma de } 0110 \\ 0001 + 0110 = 0111 \end{array}}$	$=$	1000	$\boxed{\begin{array}{l} \text{inversión} \\ 1000 = 8_{(10)} \end{array}}$
$C'0010 =$	$0010 + 0110 = 1000$	$\boxed{\begin{array}{l} 0010 + 0110 = 1000 \\ 0010 + 0110 = 1000 \end{array}}$	$=$	0111	$\boxed{\begin{array}{l} \text{inversión} \\ 0111 = 7_{(10)} \end{array}}$
$C'0011 =$	$0011 + 0110 = 1001$	$\boxed{\begin{array}{l} 0011 + 0110 = 1001 \\ 0011 + 0110 = 1001 \end{array}}$	$=$	0110	$\boxed{\begin{array}{l} \text{inversión} \\ 0110 = 6_{(10)} \end{array}}$

luego, se realiza la suma en BCD 8421:

$$\begin{array}{r}
 0100 \quad 0101 \quad 0110 \\
 + \underline{1000} \quad \underline{0111} \quad \underline{0110} \\
 1100 \quad 1100 \quad 1100 \\
 + 0110 \quad 0110 \quad \underline{0110} \\
 \underline{\underline{0001}} \quad \underline{\underline{0001}} \quad 10010 \\
 1\ 0011 \ 10011 \\
 \\
 + \qquad \qquad \qquad 0001 \text{ para llegar a la base } \\
 \\
 1 | 0011 \quad 0011 \quad 0011 \\
 | \quad \quad \quad | \quad \quad \quad | \\
 1 \quad 3 \quad \quad 3 \quad \quad 3 \\
 \\
 \text{ja}
 \end{array}$$

A continuación, resolvemos algunos ejercicios para afianzar los procedimientos:

$$\begin{array}{r}
 1) \quad 25 + 2 = \\
 & 25 & 0010 & 0101 \\
 + & 2 & + & \underline{0000} & \underline{0010} \\
 & 27 & 0010 & 0111 \\
 & & & \boxed{2} & \boxed{7}
 \end{array}$$

$$\begin{array}{r}
 2) \quad 662 + 1045 = \\
 & \begin{array}{r} 662 \\ + 1045 \\ \hline 1707 \end{array} \quad \begin{array}{r} 0110 \\ + 0001 \\ \hline 0001 \end{array} \quad \begin{array}{r} 0110 \\ 0000 \\ \hline 0100 \end{array} \quad \begin{array}{r} 0010 \\ 0101 \\ \hline 0111 \end{array} \\
 & \quad \boxed{1} \quad \boxed{7} \quad \boxed{0} \quad \boxed{7}
 \end{array}$$

3) $234 - 152 =$

$$\begin{array}{r}
 234 & 234 & C'0001 = & 0001 + 0110 = 0111 & = 1000 & = 8_{(10)} \\
 - 152 & + 847 & C'0101 = & 0101 + 0110 = 1011 & = 0100 & = 4_{(10)} \\
 082 & 1081 & C'0010 = & 0010 + 0110 = 1000 & = 0111 & = 7_{(10)}
 \end{array}$$

$$\begin{array}{r}
 0010 & 0011 & 0100 \\
 + 1000 & 0100 & 0111 \\
 \hline
 1010 & 0111 & 1011 \\
 + 0110 & & 0110 \\
 \hline
 1\ 0000 & 0001 & 10001
 \end{array}$$

$$\begin{array}{r}
 1000 \\
 + 0001 \\
 \hline
 1\ 0000 & 1000 & 0010 \\
 | & | & | \\
 0 & 8 & 2
 \end{array}$$

$$4) \quad 623 - 54 =$$

$$\begin{array}{r}
 C'0000 = 0000 + 0110 = 0110 = 1001 = 9_{(10)} \\
 623 \quad 623 \quad C'0101 = 0101 + 0110 = 1011 = 0100 = 4_{(10)} \\
 - 54 \quad 945 \quad C'0100 = 0100 + 0110 = 1010 = 0101 = 5_{(10)} \\
 569 \quad 1568 \\
 + \underline{1} \\
 1569 \quad 0110 \ 0010 \ 0011 \\
 + \underline{1001} \ \underline{0100} \ \underline{0101} \\
 1111 \ 0110 \ 1000 \\
 + \underline{0110} \quad + \underline{0001} \\
 \hline
 1 | \underline{0101} \ \underline{1} \ \underline{1001} \\
 \quad 5 \quad 6 \quad 9
 \end{array}$$

4.4 Resumen

La denominación de “número de punto o coma fija” obedece a que se trata de números enteros. Por lo tanto, la “coma separadora de decimales” se supone alojada en la posición derecha extrema del dígito de orden inferior. Son ejemplos de ello los números $2,0_{(10)}$ o $35,0_{(10)}$ o $234\,768,0_{(10)}$. En cualquiera de estos casos “la parte decimal” es 0 , por lo que se elimina junto con la coma “que queda” (en el supuesto lugar) “fija”. En representación binaria de coma fija, un entero sumando o restando de otro entero da un número de igual cantidad de bits que los operandos. En el caso de la multiplicación, se debe considerar que si los operandos son de n bits, el resultado puede llegar a ocupar $2 \cdot n$, o sea que se expande al doble. En el caso de la división de dos enteros de n bits, el cociente se considerará otro entero de n bits y el resto, otro también de n bits.

En el mundo, la mayoría de la gente utiliza aritmética decimal (base 10) y, cuando los valores son muy grandes o muy pequeños, se emplean los exponentes en potencias de diez, mientras que los números en coma flotante (base 2) sólo pueden aproximar números decimales comunes. En aplicaciones financieras o comerciales, donde no se admiten errores cuyo origen sea la conversión entre sistemas, caso presentado en el número $0,2_{(10)}$ aproximadamente igual a $0,0011_{(2)}$, se deberán utilizar lenguajes que permitan declarar variables “tipo decimal”. Los cálculos sobre este tipo de datos se operan usando la aritmética decimal, que es más eficiente también con respecto a la precisión, ya que para los reales utiliza un entero como mantisa y un entero que representa una potencia de diez como exponente.

En Java la “Clase BigDecimal” representaría el número $8.999999E-10$ con esta precisión, mientras que el mismo número para el tipo de dato “Java float” se representaría $9E-10$. Es preocupación del diseñador de la aplicación ver si la diferencia es significativa o no para la tarea en cuestión.

Como contrapartida, los números decimales utilizan 20% más de almacenamiento que la representación en binario puro. Recuérdese que el número $255_{(10)}$ en formato de 8 bits en representación de coma fija sin signo es “11111111” (se almacena en un byte), mientras que el mismo número en BCD desempaquetado ocupa tres bytes: el primero “0000 0010” para el 2, el segundo “0000 0101” para el primer 5 y el tercero “0000 0101” para el último 5.

Los cálculos en decimal requieren 15% más de “circuitería”, debido a las correcciones que surgen de operar “en binario pero en base diez”, y por esta razón son más lentos. Recuérdese que la suma de “9+1” en BCD requiere la corrección con el número $6_{(10)}$ o “0110”, pues el código BCD está definido sólo hasta el dígito decimal 9.

4.5 Ejercicios propuestos

- 1) Realizar en representación de punto fijo, tal como lo haría la ALU, las siguientes sumas para un formato de 6 bits. Verificar el resultado en decimal, teniendo en cuenta que los negativos están representados en la forma de complemento a 2.

- $00110_{(2)} + 110010_{(2)}$
- $010101_{(2)} + 000011_{(2)}$
- $111001_{(2)} + 001010_{(2)}$
- $101011_{(2)} + 111000_{(2)}$

- 2) Realizar las siguientes restas en representación de punto fijo, tal como lo haría la ALU, para un formato de 6 bits y utilizar negativos complementados a la base. Verificar el resultado en decimal.

- $010101_{(2)} - 000111_{(2)}$
- $001010_{(2)} - 111001_{(2)}$
- $111001_{(2)} - 001010_{(2)}$
- $101011_{(2)} - 100110_{(2)}$

- 3) Efectuar la siguiente operación como lo haría la ALU en representación de punto fijo con formato de 8 bits y negativos complementados a la base. Indicar el valor final de los flags S, Z, V, C y qué representa cada uno de ellos.

$$\{-85\}_{(10)} - \{+53\}_{(10)}$$

- 4) Considerar que los siguientes datos hexadecimales están representando operandos binarios enteros signados. Realizar su suma. Comprobar en decimal.

20; F9

- 5) Dadas las siguientes operaciones en decimal, efectuarlas como lo haría la ALU, en un formato de 8 bits, con negativos complementados a la base. Indique el valor del resultado y el de los flags S, Z, V y C. Compruebe el resultado obtenido convirtiéndolo a decimal.

- $101 + 29$
- $101 + \{-29\}$
- $\{-101\} + 29$
- $\{-101\} + \{-29\}$
- $101 - 29$
- $101 - \{-29\}$
- $\{-101\} - 29$
- $\{-101\} - \{-29\}$

- 6) Sea la siguiente sentencia de un lenguaje de alto nivel (llamado X):

$$C = A - B$$

- Indicar un código que la represente en el programa fuente y cuántos octetos ocuparía en memoria.

- Si $A = +3_{(10)}$ y $B = -54_{(10)}$ indicar cómo llegarían los datos a la ALU si fueran definidos como binarios enteros signados.

- Operar en formato de 16 bits, en complemento a la base, e indicar el estado de los flags S, Z, V, C.

- Indicar los valores a partir de los cuales hay *overflow* para enteros naturales y para enteros signados en C_2 , según el formato especificado en el párrafo anterior.

- 7) Suponga que desea realizar la siguiente operación $C = A + B$ en binario de punto fijo en complemento a la base, como lo haría una ALU de 16 bits, siendo:

$$A = +102_{(10)}$$

$$B = -48_{(10)}$$

- ¿Cuál es el resultado de la operación?

- ¿Cuál es el estado de los flags?

- Realice la comprobación en decimal del resultado.

- Exprese el resultado en binario de punto flotante, con un exceso igual a $2^p - 1$, siendo $m = 32$ y $p = 8$ bits, y con mantisa fraccionaria normalizada.

- 8) Realizar la siguiente operación en BCD puro y comprobar en decimal:

$$2400_{(10)} + 1625_{(10)}$$

- 9) Realizar las siguientes sumas en decimal codificado en binario y comprobar en decimal:

$$23 + 4$$

$$680 + 1027$$

- 10) Realizar las siguientes restas en BCD y comprobar en decimal:

$$a) 234 - 152$$

$$b) 725 - 435$$

- 11) Multiplicación en complemento a dos. Algoritmo de Booth

El algoritmo de Booth utiliza cuatro elementos de almacenamiento. Éstos pueden pensarse como variables o registros de cálculo y se denominan A para contener el multiplicando, M para contener el multiplicador y Q como variable o registro auxiliar (que además de contener la misma cantidad de bits que A y M, utiliza un bit más al que identificaremos como Q-1, dado que se relaciona con Q0 en los desplazamientos). Para el ejemplo utilizaremos un formato de 4 bits, donde el bit de orden superior es siempre el signo del operando y los números

para multiplicar serán -4 y +2. El resultado esperado es -8 y, obviamente, estará complementado por ser negativo; se debe considerar que el resultado de la multiplicación queda alojado en A y Q. A continuación, se describen los pasos del algoritmo.

1. Se inicializan A y Q-1 con 0 y una variable denominada cuenta con 4, que es la cantidad de bits del formato.
2. Se carga Q con el multiplicando y M con el multiplicador.
3. Se compara Q0 bit menos significativo de Q y Q-1.
4. Si son iguales a "0" "1", se suma A con M y se almacena en A; luego se salta al paso 7.
5. Si son iguales a "1" "0", se resta M de A y se almacena en A. Cuidado, recuerde que se debe realizar la resta con el complemento a la base de M. Luego se salta al paso 7.
6. Si los pasos 4 y 5 fueron falsos, significa que el resultado de la comparación fue "1" "1" o "0" "0", en cuyo caso se salta en forma directa al paso 7.

7. Se desplazan a la derecha los bits del grupo A, Q y Q-1, manteniendo el signo en A3, y se pierde el valor actual de Q-1 a causa del desplazamiento. Esto se conoce como desplazamiento aritmético a derecha (SAR o *Shift Arithmetic Right*), dado que se conserva el signo.

8. Se decrementa cuenta en una unidad.
9. Si cuenta no llegó a 0, se vuelve al paso 3, caso contrario se termina la secuencia de pasos, ya que se logró el objetivo. Se pide:

 - A. Realizar un diagrama de flujo que represente los pasos enunciados.
 - B. Indicar en cada paso cuál es el verbo *assembler 80x86* que se utilizaría si se deseara programar el algoritmo.
 - C. Observar la tabla descripta a continuación para comprender el funcionamiento del algoritmo.
 - D. Realizar una tabla idéntica para los operandos -3 y +2.

Tabla de secuencia de una multiplicación con el algoritmo de Booth.

Cuenta	A	Q	Q ₋₁	M
1 y 2	4	0000	1100	0 0010 Inicialización de los registros o las variables
3				Comparar Q0 y Q-1
7 y 8	3	0000	0110	0 0010 Sólo desplazamiento Primer ciclo
7 y 8	2	0000	0011	0 0010 Sólo desplazamiento Segundo ciclo
5	1110	0011	0	0010 A A-M Tercer ciclo
7 y 8	1	1111	0001	1 0010 Desplazar A, Q, Q-1
7 y 8	0	1111	1000	1 0010 Sólo desplazamiento Cuarto ciclo
9	1111	1000	1	0010 Fin del algoritmo resultado 1111000 igual -8

4.6 Contenido de la página Web de apoyo

El material marcado con asterisco (*) sólo está disponible para docentes.



Resumen gráfico del capítulo

Simulación

Resuelve el algoritmo de Booth paso a paso

Autoevaluación

Vídeo explicativo (02:13 minutos aprox.)

Audio explicativo (02:13 minutos aprox.)

Evaluaciones Propuestas*

Presentaciones*

5

Álgebra de Boole

Contenido

5.1 Introducción	88
5.2 Álgebra de los circuitos digitales	88
5.3 Álgebra de Boole	89
5.4 Función booleana.....	92
5.5 Compuertas lógicas o <i>gates</i>	93
5.6 Circuito lógico.....	99
5.7 Circuito sumador-binario en paralelo	102
5.8 Formas normales o canónicas de una función.....	104
5.9 Circuitos equivalentes.....	107
5.10 Minimización de circuitos.....	108
5.11 Resumen.....	110
5.12 Ejercicios propuestos.....	111
5.13 Contenido de la página Web de apoyo.....	112

Objetivos

- Brindar una introducción al Álgebra proposicional y al Álgebra de Boole.
- Formular, resolver y minimizar en la confección de los circuitos digitales que intervienen en la implementación de una computadora.



En la página Web de apoyo encontrará un breve comentario de la autora sobre este capítulo.



La palabra "digital" describe la tecnología que genera, almacena y procesa datos en términos de 0 y 1, denominados bits. Antes, la transmisión electrónica se denominaba analógica; en esta tecnología los datos se convertían en una señal, de frecuencia o amplitud, variable sobre una onda portadora de una frecuencia determinada. La radio y el teléfono utilizaban tecnología analógica.

5.1 Introducción

En este capítulo se presentan los temas del Álgebra de Boole requeridos en el diseño de circuitos lógicos del capítulo Lógica Digital en el que se analizarán y diseñarán otros sistemas digitales sencillos. Se presentan los operadores lógicos básicos y los dispositivos que los implementan, las compuertas lógicas. Los circuitos que se presentan en este capítulo son "combinacionales", o sea que el valor de las salidas sólo depende del valor de las entradas. Para representar un circuito se utiliza el diagrama de bloque, que sólo indica las entradas y las salidas del circuito, y el diagrama de lógica. Luego se explica el método basado en mapas de Karnaugh para simplificación de funciones. Para la representación de circuitos se utiliza la norma establecida por el ANSI (*American National Standard Institute*).

5.2 Álgebra de los circuitos digitales

5.2.1 Elementos de Álgebra proposicional

El Álgebra proposicional estudia la operación entre elementos denominados **proposiciones**. Los siguientes enunciados son ejemplos de ellos:

"Los gatos son animales"

"Los perros son animales"

Una proposición es una sentencia declarativa de la que tiene sentido decir si es verdadera o falsa. Esto implica que es posible dar a una proposición un **valor de verdad**. Sin embargo, el uso del lenguaje puede llevar a conclusiones absurdas. Si se reemplaza "son" por "=", "Los gatos" por "a", "Los animales" por "b" y "Los perros" por "c", se puede escribir: $a = b, c = b$ y, por lo tanto, se puede concluir erróneamente que $a = c$, es decir que los gatos son perros.

Por esta razón, el Álgebra proposicional vacía las proposiciones de contenidos semánticos y sólo estudia su relación utilizando una serie de operadores denominados **operadores lógicos**.

Una proposición constituida por dos o más proposiciones simples relacionadas por operadores lógicos se denomina **proposición compuesta**, y se le puede adjudicar un valor de verdad. Algunos operadores lógicos son:

- La conjunción (\wedge).
- La disyunción inclusiva (\vee).
- La disyunción exclusiva ($\vee\bar{v}$).

Dadas dos proposiciones, vacías de contenido semántico y simbolizadas como p y q , se puede confeccionar una **tabla de verdad** que represente las cuatro combinaciones de valores de verdad:

- Si ambas son falsas.
- Si p es falsa y q verdadera.
- Si p es verdadera y q es falsa.
- Si ambas son verdaderas.

Para cada una de estas combinaciones se describe un valor de verdad para cada operador. Considerese a F como el literal que representa el valor de verdad falso y a V como el literal que representa el valor de verdad verdadero. En la tabla 5-1 se muestra el valor de verdad de cada proposición compuesta para los operadores mencionados:

Tabla 5-1. Tabla de verdad para dos proposiciones.

<i>p</i>	<i>q</i>	<i>p</i> ∧ <i>q</i>	<i>p</i> ∨ <i>q</i>	<i>p</i> ∨ <i>q</i>
F	F	F	F	F
F	V	F	V	V
V	F	F	V	V
V	V	V	V	F

El operador **negación** se aplica a una sola proposición, simple o compuesta. El valor de verdad *F* (falso) es la negación de *V* (verdadero) y el valor de verdad *V* (verdadero) es la negación de *F* (falso). Como se observa en la tabla 5-2 el símbolo $\sim p$ representa la negación de la proposición *p*.

Boole (1815-1864). Matemático británico que es considerado uno de los fundadores de las Ciencias de la Computación, debido a su creación del Álgebra booleana, la cual es la base de la Aritmética computacional moderna.

Tabla 5-2. Tabla de verdad de negación.

<i>p</i>	$\sim p$
F	V
V	F

5.3 Álgebra de Boole

George Boole, en el siglo XIX, se dedicó a formalizar y mecanizar el proceso del pensamiento lógico y desarrolló una teoría lógica que utilizaba variables en lugar de proposiciones. Los valores que puede tomar una variable "booleana" son *0* o *1*.

En 1938, Claude Shannon propuso la utilización del Álgebra de Boole en el análisis de circuitos eléctricos, lo que años más tarde permitió profundizar en el análisis y el desarrollo de computadoras electrónicas.

El Álgebra de Boole es un ente matemático constituido por los elementos *0* y *1* y las operaciones suma lógica, producto lógico y complemento.

Relacionando lo expresado antes, las proposiciones del Álgebra proposicional son variables en el Álgebra de Boole y los valores de verdad que se asignaban a proposiciones son los que pueden tomar las variables. El *0* se relaciona con el falso y el *1* con el verdadero. El Álgebra de Boole constituye el fundamento teórico para el diseño de circuitos digitales. El valor del Álgebra de Boole, plasmado en su "Investigación sobre las leyes del pensamiento", consiste en lograr un análisis lógico con un soporte matemático desconocido para su época. Su obra fue interpretada como base teórica del desarrollo de la Cibernética.

Shannon (1916-2001). Ingeniero eléctrico y matemático estadounidense, considerado el fundador de la teoría de la información.

Demostró que el Álgebra booleana se podía utilizar en el análisis y la síntesis de la comutación de los circuitos digitales, idea que fue calificada como una de las aportaciones teóricas fundamentales que ayudó a cambiar el diseño de los circuitos digitales.

5.3.1 Operadores

Los operadores "booleanos" son:

- Producto lógico (\cdot).
- Suma lógica ($+$).
- Complemento (\sim).

Otra vez, haciendo referencia al Álgebra proposicional podemos relacionar los operadores "booleanos" con los operadores lógicos, ya vistos en la sección anterior.

Desde el punto de vista semántico, los nombres de las variables utilizan las primeras letras del abecedario en minúscula, por eso, reemplazamos "*p*" por "*a*" y "*q*" por "*b*".

El "producto lógico" o "booleano" está relacionado con la tabla de verdad de la conjunción y se simboliza con un punto (\cdot).

La "suma lógica" o "booleana" está relacionada con la tabla de verdad de la disyunción inclusiva y se simboliza con un signo más ($+$).

El "complemento" es un operador equivalente a la negación. En las tablas 5-3 y 5-4 se representan los valores de verdad para cada operador en términos "booleanos".

Tabla 5-3. Tabla para dos variables.

a	b	$a \cdot b$	$a + b$
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

Tabla 5-4. Tabla de verdad.

a	\bar{a}
0	1
1	0



Encuentre un simulador que permite crear el diagrama lógico de una expresión booleana en la Web de apoyo.

5.3.2 Tablas de verdad

Una tabla de verdad es: "Una lista ordenada de las 2^n combinaciones distintas de ceros y unos, que se pueden obtener de la combinación del valor de n variables binarias".

Para 3 variables, y considerando que los valores que puede tomar cada una son sólo 0 o 1, la cantidad de combinaciones binarias distintas es de $2^3 = 8$, para 4 variables la cantidad de combinaciones es, entonces, $2^4 = 16$. En términos generales, con n variables se pueden obtener 2^n combinaciones diferentes.

5.3.3 Propiedades del Álgebra de Boole

Los enunciados que se expresan a continuación pertenecen a las propiedades que todo elemento o toda operación del Álgebra de Boole deben cumplir.

Leyes comutativas

Ambas operaciones binarias (suma y producto) son comutativas, esto es que si a y b son elementos del Álgebra se verifica que:

$$a + b = b + a$$

$$a \cdot b = b \cdot a$$

Leyes Distributivas

Cada operación binaria (suma y producto) es distributiva respecto de la otra:

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

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

Leyes de Identidad

Dentro del Álgebra existen dos elementos neutros, el 0 y el 1, que cumplen la propiedad de identidad con respecto a cada una de las operaciones binarias:

$$a + 0 = a$$

$$a \cdot 1 = a$$

Leyes De Complementación

Para cada elemento a del Álgebra, existe un elemento denominado a negada, tal que:

$$a + \bar{a} = 1 \quad a \text{ y } \bar{a} \text{ no pueden ser cero al mismo tiempo,}$$

$$a \cdot \bar{a} = 0 \quad a \text{ y } \bar{a} \text{ no pueden ser uno al mismo tiempo.}$$

Estas dos leyes definen el **complemento** de una variable.

5.3.4 Teoremas del Álgebra de Boole

Sobre la base de los postulados anteriores, se deduce una serie de teoremas. La demostración de cada teorema se puede realizar en forma algebraica o mediante la tabla de verdad.

Teorema 1

De las expresiones anteriores se deduce el "principio de dualidad", que determina que estas expresiones permanecen válidas si se intercambian las operaciones $+$ por \cdot y los elementos 0 por 1 .

Teorema 2

Para cada elemento del Álgebra de Boole se verifica que:

$$a + 1 = 1$$

$$a \cdot 0 = 0$$

Según este teorema y las "leyes de identidad" citadas anteriormente, se deduce que:

$$0 + 0 = 0 \quad 0 \cdot 0 = 0$$

$$0 + 1 = 1 \quad 0 \cdot 1 = 0$$

$$1 + 1 = 1 \quad 1 \cdot 1 = 1$$

Teorema 3

Para cada elemento a de un Álgebra de Boole se verifica que:

$$a + a = a \quad \text{Ley de idempotencia}$$

$$a \cdot a = a$$

Teorema 4

Para cada par de elementos del Álgebra de Boole, a y b , se verifica que:

$$a + (a \cdot b) = a \quad y \quad \text{Ley de absorción}$$

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

Confeccionando la tabla de verdad se puede demostrar, por ejemplo, para la primera igualdad (tabla 5-5):

Tabla 5-5			
a	b	$a \cdot b$	$a + a \cdot b$
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

Teorema 5

En el Álgebra de Boole la suma y el producto son asociativos:

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

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

Teorema 6

Para todo elemento a de un Álgebra de Boole, se verifica que:

$$\bar{\bar{a}} = a \quad \text{Ley de involución}$$

Comprobación:

a	\bar{a}	$\bar{\bar{a}}$
0	1	0
1	0	1

Teorema 7

En todo Álgebra de Boole se verifican las siguientes igualdades, conocidas como "Leyes de De Morgan", que permiten transformar sumas en productos y productos en sumas:

$$\begin{aligned} \overline{a + b + c + \dots} &= \bar{a} \cdot \bar{b} \cdot \bar{c} \cdots \\ \overline{a \cdot b \cdot c \cdots} &= \bar{a} + \bar{b} + \bar{c} + \dots \end{aligned}$$

5.4 Función booleana

Una función "booleana" es: "Una expresión algebraica constituida por variables binarias, los operadores binarios suma lógica y producto lógico, el operador complemento, el paréntesis y el signo igual".

La descripción del valor que asume una función se representa en una tabla de verdad y se denomina representación estándar; por ejemplo, en la tabla 5-6 se describe el valor de las funciones $f_2 = a \cdot \bar{b}$; $f_4 = \bar{a} \cdot b$; $f_{12} = a + \bar{b}$ y $f_{15} = 1$.

Tabla 5-6. Descripción del valor que asume una función.

a	b	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
		0	and	a		b	xor	or	nor	xnor	\bar{b}	\bar{a}	nan		1		

"Para n variables se pueden definir 2^{2^n} funciones distintas". En la tabla 5-7 se describen las diecisésis funciones que se pueden definir para dos variables.

Tabla 5-7. Funciones que se pueden definir para dos variables.

a	b	$a \cdot b$	$a + a \cdot b$
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

Cada una de estas funciones se puede expresar algebraicamente así:

$$\begin{aligned}
 f_0 &= 0 \\
 f_1 &= a \cdot b \\
 f_2 &= a \cdot \bar{b} \\
 f_3 &= a \\
 f_4 &= \bar{a} \cdot b \\
 f_5 &= b \\
 f_6 &= a \cdot \bar{b} + \bar{a} \cdot b \\
 f_7 &= \overline{a + b} \\
 f_8 &= a + b \\
 f_9 &= a \cdot b + \bar{a} \cdot \bar{b} \\
 f_{10} &= \bar{b} \\
 f_{11} &= a + \bar{b} \\
 f_{12} &= \bar{a} \\
 f_{13} &= \bar{a} + b \\
 f_{14} &= a \cdot \bar{b} \\
 f_{15} &= 1
 \end{aligned}$$

5.5 Compuertas lógicas o gates

Una compuerta lógica es la "representación" de una red de conmutadores. Cada conmutador se controla con una señal binaria identificada como a, b, c, \dots, n . Cada una de estas señales constituye una "entrada" de la compuerta o "terminal de entrada". La estructura de la red de conmutadores genera una señal binaria de salida, o "terminal de salida", identificada como f , que es una función de a, b, c, \dots, n . Para cada combinación de entradas, la función $f(a, b, c, \dots, n)$ asume un valor 0 o 1. La tabla de verdad de la función f representa el comportamiento de la compuerta para cada combinación de valores.

El Álgebra de Boole enunciada en este libro en su forma binaria es equivalente, por definición, al Álgebra de conmutadores.



Una compuerta lógica es el bloque elemental que permite la implementación de circuito digital. La mayoría tiene dos entradas y una salida, que también se denominan "terminales"; cada terminal puede estar en uno de dos estados, 0 o 1, representados por diferentes niveles de voltajes.

Comutador

Es un dispositivo físico que permite controlar el flujo de un elemento (corriente eléctrica, agua, etc.). En la vida cotidiana vemos innumerables ejemplos de conmutadores, por ejemplo, en una habitación la luz se prende o apaga con un conmutador manual.

Comutador ideal

Es un modelo teórico que representa el comportamiento de un modelo real. El estudio del funcionamiento de un conmutador ideal es independiente de la tecnología que se utilice para su implementación. En un conmutador ideal se utilizan los dígitos 0 y 1 para representar el estado de una entrada o una salida y, también, para representar el estado del conmutador abierto o cerrado.



Encuentre una animación sobre cómo trabajan los interruptores no mecánicos en la Web de apoyo.



Los terminales de entrada de un circuito aceptan señales binarias, dentro de tolerancias de voltajes permitidas, y generan en los terminales de salida señales binarias que cumplen también el convenio de voltaje establecido para ese circuito.



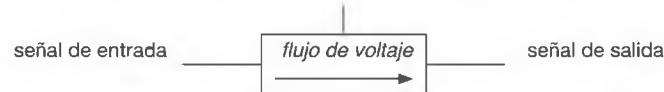
La información binaria se representa en un sistema digital por cantidades físicas denominadas señales.

Circuitos de conmutadores

A una agrupación de varios conmutadores relacionados entre sí se la denomina circuito de conmutación. El Álgebra de Boole constituye el fundamento teórico para su diseño. Los primeros circuitos de conmutación se diseñaron con contactos, el elemento 0 representa un contacto que está abierto y el elemento 1 representa un contacto que está cerrado.

Señal

En lugar de cambiar el estado de un conmutador manualmente, como ocurre con el interruptor de la luz de una habitación, en un conmutador electrónico se utiliza una señal binaria, que se denomina señal de control o **señal de entrada**. Esta señal regula el flujo de un voltaje por medio del conmutador, que es, a su vez, otra señal binaria que se denomina señal de dato o **señal de salida** del conmutador.



En la figura anterior se muestra un típico símbolo gráfico de conmutador. La flecha muestra el sentido de circulación de las señales binarias y las líneas más gruesas representan los **terminales** de acceso o egreso de las señales.

Lógica positiva

Los componentes electrónicos binarios operan solamente con dos niveles de tensión de señales, llamados nivel lógico "1" y nivel lógico "0".

Cuando se indica que un conmutador opera con **lógica positiva**, la tensión lógica "1" es más positiva que la tensión lógica "0", lo que significa que la tensión lógica "0" está próxima a la tensión real cero y la tensión lógica "1" se sitúa en un cierto nivel de tensión positiva. Si, por el contrario, el conmutador trabaja con lógica negativa, la tensión lógica "1" es menos positiva que la tensión lógica "0". Lo que determina el tipo de lógica es la asignación de los valores lógicos de acuerdo con las amplitudes relativas de las señales.

Valor lógico	Valor de la señal	Valor lógico	Valor de la señal
1	High + 5V	0	High - 0,3V
0	Low + 0,2V	1	Low - 4V

Lógica positiva

Lógica negativa

Convenios de voltaje

Como la tensión es una magnitud continua sujeta a fluctuaciones, se establece un rango de variabilidad para cada estado lógico. Cuando una señal cambia de uno a otro estado se encuentra, en determinado momento, fuera de los rangos permitidos; esta etapa se denomina período de transición y no es tomada en cuenta como valor lógico para la señal.

En la figura 5.1 se observa un ejemplo de señal binaria en un convenio de voltaje que utiliza muchos **circuitos integrados**:

- Las zonas A y C corresponden a los intervalos considerados aceptables para una señal de salida.

- La zona B corresponde al intervalo de transición, o sea, cierto intervalo en el que el valor de la señal pasa de un estado lógico al otro y no es tomado en cuenta

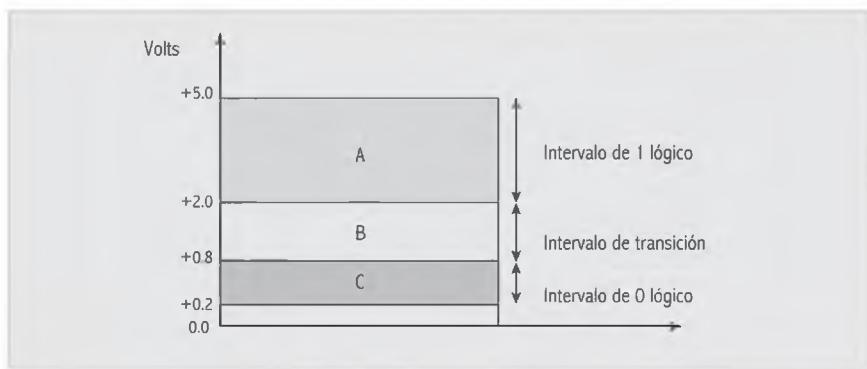


Fig. 5.1. Convenio de voltaje.

Este diagrama pertenece a las llamadas señales de "lógica positiva", donde la tensión lógica "1" es más positiva que la tensión lógica "0".

5.5.1 "Compuerta AND", "compuerta y" o "compuerta producto lógico"

La tabla de verdad de la función producto lógico representa el comportamiento de la salida de esta compuerta:

Compuerta AND

La salida de la compuerta (función AND) sólo toma el valor 1 lógico cuando todas las entradas son 1 lógico.

Si la compuerta tiene, por ejemplo, 2 entradas será (figura 5.2):



Fig. 5.2. Compuerta AND.

La salida de esta compuerta responde a la tabla de verdad del producto lógico, para cada par de valores correspondientes a las cuatro combinaciones posibles de dos entradas.

5.5.2 "Compuerta OR", "compuerta +" o "compuerta suma lógica"

La tabla de verdad de la función suma lógica representa el comportamiento de la salida de esta compuerta:

Compuerta OR

La salida de la compuerta (función OR) sólo toma el valor 0 lógico cuando todas las entradas son 0 lógico.

Si la compuerta tiene solo dos entradas, será (fig. 5.3):



Fig. 5.3. Compuerta OR.

La salida de esta compuerta responde a la tabla de verdad de la suma lógica, para cada par de valores correspondientes a las cuatro combinaciones posibles de dos entradas.

5.5.3 "Compuerta OR EXCLUSIVA" o "compuerta exclusiva"

En realidad, el comportamiento de esta compuerta no responde a ningún operador "booleano". Sin embargo, su comportamiento se puede comprender a partir de la función "disyunción exclusiva" del Álgebra proposicional. Recuerde que esa función sólo es falsa cuando el valor de verdad de todas las proposiciones es el mismo; en el lenguaje de lógica de compuertas esto se declara como:

Compuerta XOR

*La salida de la compuerta (función XOR) sólo toma el valor 0 lógico cuando **todas** las entradas son 0 lógico, o bien cuando una cantidad par de entradas son 1 lógico.*

Si la compuerta tiene, por ejemplo, dos entradas, será (fig. 5.4):

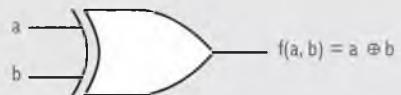


Fig. 5.4. Compuerta XOR.

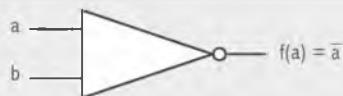
La salida de esta compuerta responde a la tabla de verdad de la suma "aritmética", para cada par de valores correspondientes a las cuatro combinaciones posibles de dos entradas.

5.5.4 "Compuerta NOT" o "inversión"

La tabla de verdad de la función complemento representa el comportamiento de la salida de esta compuerta (fig. 5.5):

Compuerta NOT

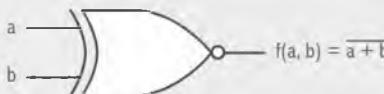
Siempre tiene una sola entrada y una sola salida, siendo el valor de la salida el complemento del valor de la entrada.

**Fig. 5.5.** Compuerta NOT.**5.5.5 “Compuertas con funciones negadas”**

Se pueden definir dos nuevas compuertas, que son las más utilizadas como elementos básicos para la realización de los circuitos actuales. Estas compuertas se denominan: NOR (NO-OR) y NAND (NO-AND).

Compuerta NAND (fig 5.6)**Fig. 5.6.** Compuerta NAND.

La salida de la compuerta (función NAND) sólo toma el valor 0 lógico cuando todas las entradas son 1 lógico.

Compuerta NOR (fig 5.7)**Fig. 5.7.** Compuerta NOR.

La salida de la compuerta (función NOR) sólo toma el valor 1 lógico cuando todas las entradas son 0 lógico.

Las tres funciones elementales, suma, producto e inversión lógica, pueden ser representadas con compuertas NOR y NAND. En efecto, aplicando el teorema de De Morgan se tiene:

La negación de la suma de dos variables puede expresarse como el producto de las variables negadas.

Negar la función suma dos veces. Por teorema 6 (involución), la doble negación de una función es igual a la función.

Aplicar el teorema enunciado a una de las negaciones.

La función queda expresada con un producto y tres negaciones.

Desde el punto de vista del circuito la compuerta OR fue reemplazada por compuertas NOT y NAND.

Veamos cómo aplicamos este procedimiento (fig. 5.8):

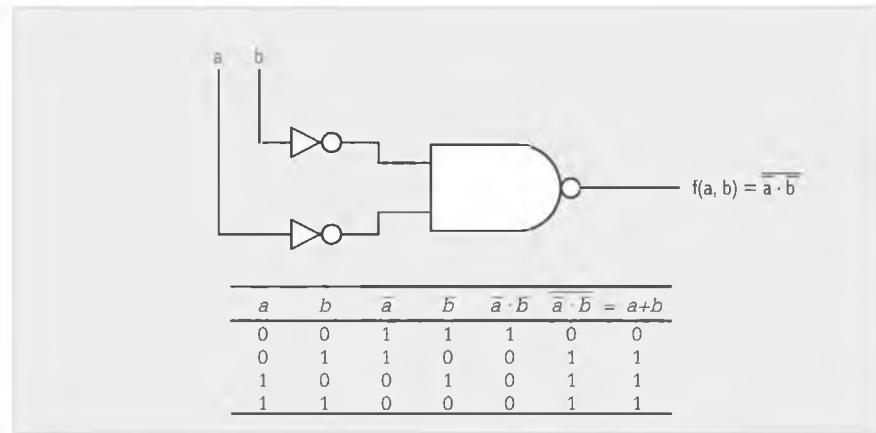


Fig. 5.8. Teorema de De Morgan aplicado a la suma lógica.

La negación del producto de dos variables puede expresarse como la suma de las variables negadas.

Negar la función producto dos veces. Por teorema 6 (involución) la doble negación de una función es igual a la función.

Aplicar el teorema enunciado a una de las negaciones.

La función queda expresada con una suma y tres negaciones.

Desde el punto de vista del circuito la compuerta AND fue reemplazada por compuertas NOT y NOR.

Veamos cómo aplicamos este procedimiento (fig. 5.9):

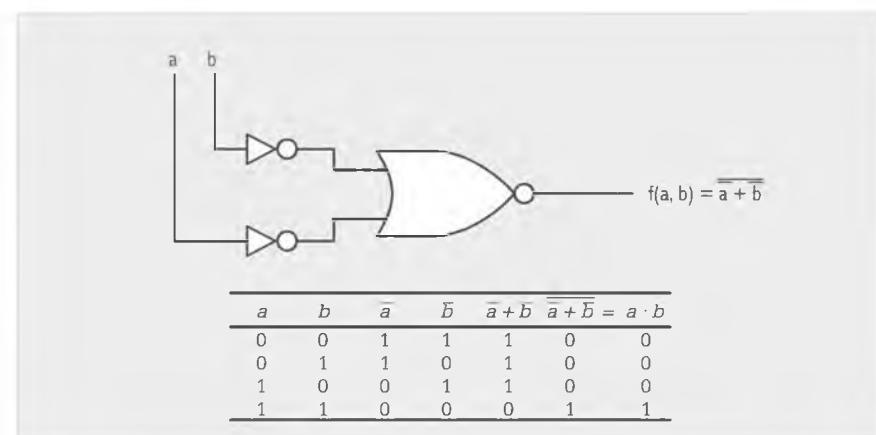


Fig. 5.9. Teorema de De Morgan aplicado al producto lógico.

La inversión se realiza con una función NOR o NAND con sus dos entradas iguales. Veamos cómo se representa gráficamente para con una compuerta NOR (fig. 5.10):

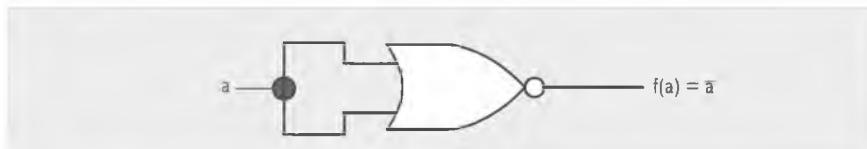


Fig. 5.10. Implementación de la negación con compuertas negadas.

Compuerta YES

Un símbolo triángulo por sí solo designa un circuito separador, que no produce ninguna función lógica particular, puesto que el valor binario de la salida es el mismo que el de la entrada. Este circuito se utiliza simplemente para amplificación de la señal. Por ejemplo, un separador que utiliza 5 volts para el binario 1 producirá una salida de 5 volts cuando la entrada es 5 volts. Sin embargo, la corriente producida a la salida es muy superior a la corriente suministrada a la entrada de la misma. De esta manera, un separador puede excitar muchas otras compuertas que requieren una cantidad mayor de corriente, que no se encontraría en la pequeña cantidad de corriente aplicada a la entrada del separador. Se podría pensar que una compuerta YES es inútil, si se considera que el mismo valor lógico de la entrada se establece en la salida. Sin embargo, los ceros y unos son corriente eléctrica a cierto voltaje. La corriente no es suficiente si el abanico de salida es grande. Se denomina "abanico de salida" a la cantidad de dispositivos a la que el terminal de salida de una compuerta lógica puede abastecer.

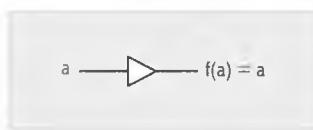


Fig. 5.11. Compuerta lógica YES.

5.6 Circuito lógico

Un circuito lógico es la representación de la estructura de una red de compuertas y su comportamiento se expresa algebraicamente por una o varias funciones booleanas; las variables constituyen las entradas y las funciones constituyen las salidas del circuito.

La expresión algebraica de la función relaciona las variables con los operadores suma lógica, producto lógico y complemento, que se representan en el circuito, también denominado **diagrama de lógica**, con las compuertas que representan a estos operadores.

Así, la función $f = (a \cdot b) + c$ está definiendo un circuito de tres entradas y una salida, cuyo valor representa el que la función asume en cada caso según la tabla de verdad adjunta (fig. 5.12):

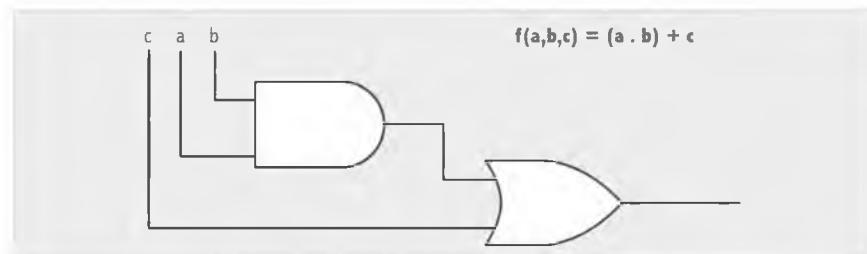


Fig. 5.12. Diagrama de lógica.

El dibujo del circuito se denomina **diagrama de lógica** (tabla 5-8).

Tabla 5-8. Diagrama de lógica

a	b	c	f (a,b,c)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

5.6.1 Transistor

Es un conmutador electrónico que controla el flujo de un nivel de tensión, con la particularidad de que puede conmutar (cambiar de estado) muy rápidamente.

Las compuertas se sustituyen por dispositivos electrónicos que, como explicaremos, se implementan con redes de transistores y permiten calcular una función de una, dos o más señales binarias. Estas funciones tienen relación directa con los operadores booleanos e incluso asumen nombres semejantes a ellos. Sin intención de incluir en esta unidad un tratado de técnicas digitales, observemos algunos conceptos que explican cómo se puede transferir, transformar y almacenar información binaria en una computadora.

Consideremos al transistor como el elemento básico en la construcción de una compuerta. Un transistor puede operar como si fuera un conmutador abierto (bit 0) o cerrado (bit 1), según el comportamiento de las señales binarias que actúen sobre él.

Obsérvese la figura 5.13:

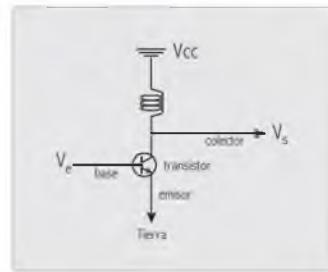


Fig. 5.13. Esquema de un transistor.

La tensión de alimentación se regula con la resistencia, cuando el voltaje de entrada está en un nivel alto, lo que permite que el transistor se encuentre en estado de conducción; la tensión en la salida cae, asume el valor de tierra, que sería 0 volts.

Cuando la entrada es igual a 1 lógico la salida es igual a 0 lógico en el caso inverso, cuando la entrada es igual a 0 lógico el transistor se corta y actúa como un interruptor abierto, la salida entonces asume el valor retenido de la tensión de alimentación equivalente a 1 lógico. Entonces, en el ejemplo mostrado, el transistor reproduce la función complemento del Álgebra de Boole.

Así, con distintas estructuras de transistores se pueden armar otras funciones booleanas; la más simple es la compuerta NAND, ejemplificada en la figura 5.14:

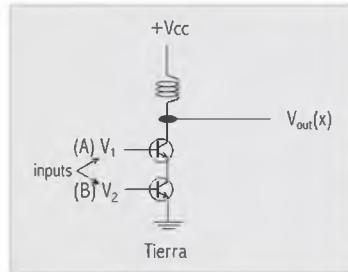


Fig. 5.14. Esquema de una compuerta NAND.

Con el mismo criterio, sólo se produce la caída de la salida cuando ambos transistores se encuentran en estado de "interruptor cerrado", esto es, cuando las entradas son iguales a 1.

En el caso de una compuerta NOR es similar, sólo que los transistores están montados en paralelo, por lo tanto, basta que uno de ellos, o ambos, estén cerrados para que la salida asuma el valor 0. La única posibilidad de que la salida mantenga un valor semejante a la tensión de alimentación es que ambos transistores estén abiertos; por lo tanto, las dos señales de entrada deben estar en 0 lógico.

Para lograr las compuertas AND y OR se debe acoplar a la tensión de la salida de un circuito inversor. Por lo tanto, las compuertas AND y OR son estructuras más complejas que las anteriores. Así, se demuestra que la implementación de circuitos con compuertas NAND, o bien NOR, son estructuras más simples que sus equivalentes con AND y OR, y éste es el motivo por el cual las compuertas NAND y NOR se utilizan como "compuertas básicas" en el diseño de circuitos.

De lo expuesto, surge la razón tecnológica por la que se utiliza el sistema binario como elemento representativo de la información en una computadora, dado que, en la práctica, establecer sólo dos niveles de tensión es más confiable que establecer varios niveles que respondan sin ambigüedad y con toda precisión a la información que se quiera representar.

5.6.2 Compuerta triestado

Aparece aquí un nuevo estado lógico que es el tercer estado posible en la salida de una compuerta. Éste se denomina de **alta impedancia** y significa que la salida se **desconecta** o asume un **estado flotante**. En algunos casos puede ser útil que una salida conectada a otros componentes aparezca como si no lo estuviese, el procedimiento resulta análogo a cuando en la vida cotidiana se "desenchufa" un dispositivo eléctrico, separándolo de la red. El estado de alta impedancia en un circuito con lógica triestado (de tres estados), se puede controlar con una señal adicional denominada **entrada de habilitación**. La lógica triestado es muy útil para transferir datos entre varios registros relacionados por un "bus" único.

5.6.2.1 Buffer triestado

La compuerta buffer triestado tiene una línea de entrada adicional **d** (desconectado), que opera como entrada de habilitación; con un 1 lógico habilita el estado de alta impedancia (fig. 5.15 y tabla 5-9).



Una compuerta de tres estados es un circuito digital que permite los estados 0 lógico, 1 lógico y, además, el estado de "alta impedancia".

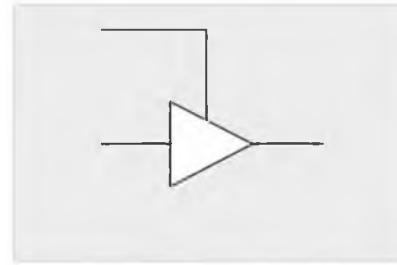


Fig. 5.15. Buffer triestado.

5.7 Circuito sumador-binario en paralelo

Ahora analizaremos cómo a partir de la expresión de una función booleana se puede confeccionar el circuito que lleve a cabo la operación que describe.

Supongamos que queremos sumar dos números de 8 dígitos cada uno:

$$\begin{array}{r}
 \text{Cy} \quad 0 \quad 0 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \\
 A \quad \quad 1 \quad 0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 1 \quad 0 \\
 + B \quad + \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 0 \quad 0 \quad 1 \\
 \hline
 S \quad \quad 1 \quad 1 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 1
 \end{array}$$

Se puede confeccionar un circuito sumador binario de 4 bits, cuya operación permita llevar a cabo la suma de los primeros cuatro bits de los operandos A y B . En la figura 5.16 el circuito se representa con cuatro módulos; el primero será un circuito semisumador (SS) y los restantes serán sumadores completos (SC).

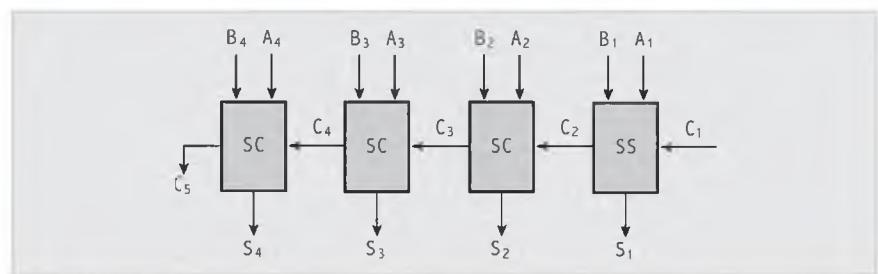
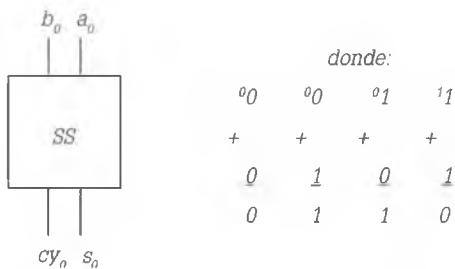


Fig. 5.16. Sumador binario paralelo con acarreo serie.

Los bits de ambos operandos constituyen la entradas del circuito sumador e ingresan al mismo tiempo (en paralelo); sin embargo, la suma de los dos primeros bits genera el acarreo después de cierto tiempo de operación del bloque semisumador, lo que produce un acarreo que afectará al bloque siguiente; esto ocurre así en todos los bloques. Las salidas del circuito son funciones que dependen de las entradas y representan los 8 bits del resultado y los sucesivos acarreos. Veamos cómo cumple su función cada bloque.

5.7.1 Circuito semisumador (SS) o Half Adder (HA)

Es un circuito cuya función es sumar 2 bits sin tener en cuenta el acarreo anterior. Tiene dos salidas, una representa la suma y la otra, el valor del acarreo (*carry*)



Estas igualdades pertenecientes al resultado de la suma (s_0) y al acarreo (cy_0), las podemos expresar con una tabla de verdad (tabla 5-9):

Tabla 5-9. Tabla para la suma de 2 bits.

a_0	b_0	s_0	cy_0
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Si se observan los valores representados en la tabla de verdad, se puede deducir que la función s_0 responde a la compuerta XOR (suma aritmética), mientras que los valores de la función cy_0 corresponden a una compuerta AND (fig. 5.17):

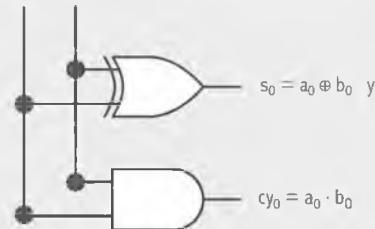
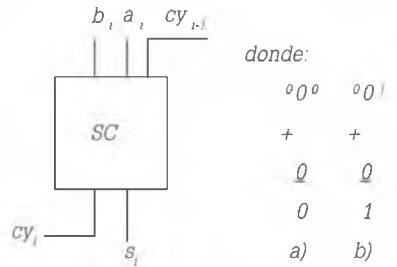


Fig. 5-17. Circuito lógico del semisumador.

5.7.2 Circuito sumador completo (SC) o Full Adder (FA)

Este circuito tiene como finalidad sumar 2 bits y el acarreo anterior, generando como salidas el resultado de la suma y el nuevo acarreo.

Genéricamente:



En el esquema anterior se observa un ejemplo de suma considerando acarreo anterior. En a) la suma de las variables con acarreo anterior en 0 da un resultado 0 y un nuevo acarreo 0; en b) la suma de las variables con acarreo anterior en 1 da un resultado 1 y un nuevo acarreo 0. En la tabla de verdad se deben representar las ocho combinaciones posibles de las tres variables de entrada al circuito, y las funciones suma (S) y nuevo acarreo (cy) con los valores que correspondan en cada caso.

Para no tener problemas de subíndices trabajaremos con:

$$b_i = b \text{ (variable que se ha de sumar)}$$

$$a_i = a \text{ (variable que se ha de sumar)}$$

$$cy_{i-1} = c \text{ (acarreo anterior)}$$

$$S_i = S \text{ (resultado de la suma)}$$

$$cy_i = C \text{ (nuevo acarreo)}$$

La tabla de verdad del circuito semisumador representa la función correspondiente a la suma S_0 y al acarreo cy_0 . Si observamos la tabla 5-10, notamos que al utilizar el mismo procedimiento para las funciones s_i y cy_i del circuito sumador completo se hace complejo determinar la estructura de compuertas del circuito. Para salvar esta dificultad recurriremos a la utilización de las formas normales o canónicas de una función.

Tabla 5-10

a_i	b_i	cy_{i-1}	S_i	Cy_i
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

5.8 Formas normales o canónicas de una función

Una función se puede representar por una serie de términos canónicos

Se llama **término canónico** de una función lógica a "todo producto o toda suma en los que aparecen todas las variables que componen una función, en su forma directa o inversa". O sea que hay **productos canónicos** y **sumas canónicas**.

Un **minitérmino** o producto canónico es: "el producto de las variables en juego, o sus negaciones individuales que hacen que el producto valga 1 (uno)". o, expresado de otro modo, "es la representación de una de las combinaciones de la tabla de verdad por medio del producto lógico".

En la tabla 5-11 se observa su representación.

Tabla 5-11

<i>a</i>	<i>b</i>	Minitérminos
0	0	$\bar{a} \cdot \bar{b}$
0	1	$\bar{a} \cdot b$
1	0	$a \cdot \bar{b}$
1	1	$a \cdot b$

En una función se pueden definir 2^n minitérminos, siendo n la cantidad de variables en juego.

5.8.1 Forma normal disyuntiva

"Si se expresa una función como la suma lógica de aquellos minitérminos que en su tabla de verdad tengan valor 1, se obtiene la expresión de su forma normal disyuntiva (FND)".

Un **maxitérmino** o suma canónica es: "la suma de las variables en juego, o sus negaciones individuales que hacen que la suma valga 0 (cero)", o, expresado de otro modo, "es la representación de una de las combinaciones de la tabla de verdad por medio de la suma lógica, pero teniendo en cuenta que los 0 en la combinación se expresan con la variable correspondiente sin negar y los 1, con la variable correspondiente negada"

En la tabla 5-12 se observa su representación.

Tabla 5-12

<i>a</i>	<i>b</i>	Maxitérminos
0	0	$a + b$
0	1	$a + \bar{b}$
1	0	$\bar{a} + b$
1	1	$\bar{a} + \bar{b}$

En una función se pueden definir 2^n maxitérminos, siendo n la cantidad de variables en juego.

5.8.2 Forma normal conjuntiva

"Si se expresa una función como el producto lógico de aquellos maxitérminos que en su tabla de verdad tengan valor 0, se obtiene la expresión de su forma normal conjuntiva (FNC)".

La FND y la FNC se denominan **formas normales o canónicas** de una función.

Ejemplos:

Sea un circuito semisumador, a partir de su tabla de verdad podremos obtener en primer lugar los minitérminos y los maxitérminos de la función y luego expresar la FND y la FNC de ésta.

Anteriormente dedujimos que:

$$S_0 = a_0 \oplus b_0 \quad y \quad Cy_0 = a_0 \cdot b_0$$

Si recuerda la tabla de verdad de la tabla 5-9, podrá hallar las formas canónicas de las funciones.

Tabla 5-13

<i>a₀</i>	<i>b₀</i>	<i>S₀</i>	<i>Cy₀</i>
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Para s_0

$$FND(s_0) = (\bar{a}_0 \cdot b_0) + (a_0 \cdot \bar{b}_0)$$

$$FNC(s_0) = (a_0 + b_0) \cdot (\bar{a}_0 + \bar{b}_0)$$

y para cy_0

$$FND(cy_0) = a_0 \cdot b_0$$

$$FNC(cy_0) = (a_0 + b_0) \cdot (a_0 + \bar{b}_0) \cdot (\bar{a}_0 + b_0)$$

En este momento podremos diseñar el gráfico del circuito a partir de las FND correspondientes a cada una de las funciones (fig. 5.18):

Luego, observando de la tabla de verdad 5-14 el circuito sumador-completo, podemos deducir las funciones de suma y acarreo correspondientes, ya sea por intermedio de la forma normal disyuntiva (FND) o por la forma normal conjuntiva (FNC). Elegiremos siempre la que más nos convenga, teniendo en cuenta la menor cantidad de términos que se han de representar, para favorecer, así, la implementación del circuito.

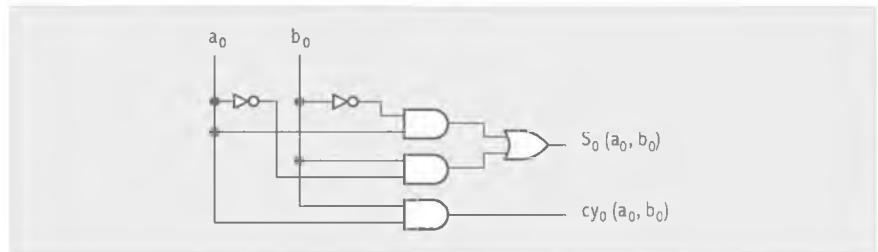


Fig. 5.18. Circuito lógico con las FND del semisumador.

Ahora hallamos las formas normales para el sumador completo.

Recordemos que:

$$b_i = b \text{ (variable que se ha de sumar)} \quad s_i = S \text{ (resultado de la suma)}$$

$$a_i = a \text{ (variable que se ha de sumar)} \quad cy_i = C \text{ (nuevo acarreo)}$$

$$cy_{i-1} = c \text{ (acarreo anterior)}$$

Tabla 5-14

a	b	c	S	C
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$FND_{(S)} = (\bar{a} \cdot \bar{b} \cdot c) + (\bar{a} \cdot b \cdot \bar{c}) + (a \cdot \bar{b} \cdot \bar{c}) + (a \cdot b \cdot c)$$

$$FNC_{(S)} = (a + b + c) \cdot (a + \bar{b} + \bar{c}) \cdot (\bar{a} + b + \bar{c}) \cdot (\bar{a} + \bar{b} + c)$$

$$FND_{(C)} = (\bar{a} \cdot b \cdot c) + (a \cdot \bar{b} \cdot c) + (a \cdot b \cdot \bar{c}) + (a \cdot b \cdot c)$$

$$FNC_{(C)} = (a + b + c) \cdot (a + b + \bar{c}) \cdot (a + \bar{b} + c) \cdot (\bar{a} + b + c)$$

Si graficamos el circuito con las formas normales disyuntivas de S y C , obtendremos el diagrama de lógica del circuito sumador-completo o *full adder* (fig. 5.19).

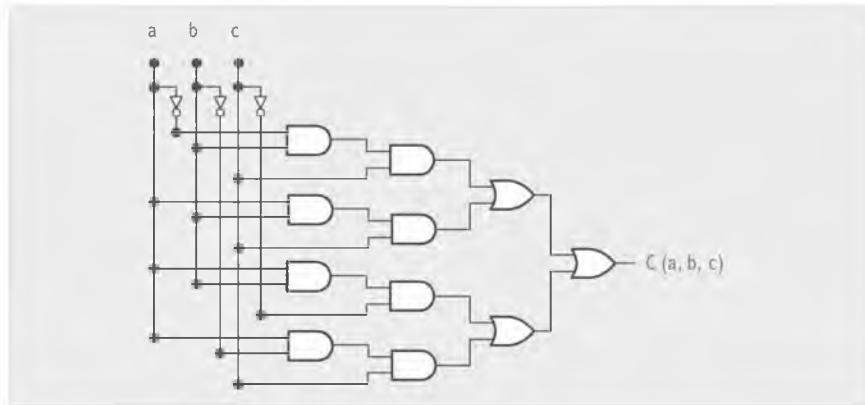


Fig. 5.19. Circuito lógico con las FND del sumador completo.

Estos diagramas están representados con compuertas de dos entradas. Recuérdese que el producto lógico (así como la suma lógica) es una operación asociativa, por lo tanto, $(a \cdot b) \cdot c$ es igual a $a \cdot (b \cdot c)$, o sea que el último producto expresado entre paréntesis se puede representar con una compuerta AND de tres entradas.

5.9 Circuitos equivalentes

Dos circuitos son equivalentes cuando son representados con distintas formas algebraicas pero responden a la misma tabla de verdad

En las figuras 5.20 y 5.21 se observan ejemplos:

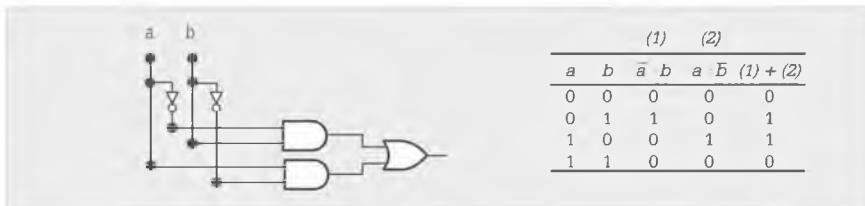


Fig. 5.20. Circuito A.

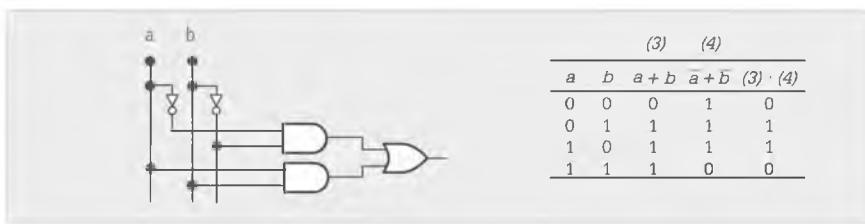


Fig. 5.21. Circuito B.



Karnaugh: Ingeniero de Telecomunicaciones estadounidense y director emérito del ICCC (*International Council for Computer Communication*). Hizo grandes aportes sobre la utilización de métodos numéricos en las Telecomunicaciones. Es el creador del método tabular o mapa de Karnaugh.

Los circuitos A y B representan las formas normales de la función $a \oplus b$. Como ambas son expresiones diferentes obtenidas de la misma tabla, generan circuitos equivalentes.

Al circuito A le corresponde la función $FND = (a \cdot b + a \cdot \bar{b})$, y al circuito B le corresponde la función $FNC = (a + b) \cdot (\bar{a} + \bar{b})$.

5.10 Minimización de circuitos

La minimización de circuitos se realiza para que el diseño de un circuito sea lo más simple posible, que cumpla la misma función con la menor cantidad de compuertas posibles, la menor cantidad de entradas a ellas y en dos niveles. Para eso se debe minimizar la función. Uno de los métodos más utilizados para la representación de funciones y su posterior reducción es la minimización por medio de un mapa de "Veitch-Karnaugh". Éste consiste en representar la matriz que contemple todas las combinaciones posibles del juego de variables de las funciones. Por lo tanto, para 2 variables que implican 4 combinaciones, tendremos que utilizar un mapa o matriz de $2 \cdot 2$ (4 casillas); para 3 variables, una matriz de $4 \cdot 2$ (8 casillas) y para 4 variables, una matriz de $4 \cdot 4$ (16 casillas). La minimización por este método para funciones de más de 4 variables de entrada se torna complicada.

Así, las líneas interceptadas por las columnas forman cubos que se completarán con un "1" o un "0" según la tabla de verdad (los 0 pueden no ponerse). Luego de este paso, se deberá agrupar la mayor cantidad de 1 adyacentes posibles, siempre y cuando esta cantidad sea una potencia exacta de 2. O sea que, en primer lugar debemos intentar hacer grupos de 8 "1" adyacentes, luego de 4 y después de 2, teniendo en cuenta que las adyacencias se producen con las aristas de los cubos y no con los vértices y que el borde superior de la matriz es adyacente con el inferior, de forma que si los unimos obtenemos un cilindro y, de idéntica manera, el borde derecho de la matriz se une con el izquierdo para possibilitar nuevas adyacencias. Agrupar la mayor cantidad de 1 posibles evita la redundancia de variables en la función final, o sea que permite hallar variables que no tienen sentido de estar porque no alterarían el funcionamiento del circuito.

Cada grupo queda definido, entonces, como la intersección de las adyacencias definidas (o sea, el producto de las zonas adyacentes). Luego, la función mínima será la unión (suma lógica) de todos los grupos definidos antes.

Ahora trataremos de minimizar con el método de Veitch-Karnaugh la función suma del semisumador, a partir de su tabla de verdad 5-13 (que repetimos abajo). Para ello confeccionaremos el mapa correspondiente.

Tabla 5-13

a_o	b_o	S_o	C_o
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Función suma del semisumador Si observamos el mapa correspondiente a la función suma (S_o), vemos que por tratarse de "1" aislados no es posible agruparlos, resultando la suma de minitérminos, o sea, la suma mínima (S_m) (tabla 5-14).

Tabla 5-15

	\bar{b}	b
\bar{a}	0	1
a	1	1

$$S_m = a\bar{b} + a\bar{b}$$

5.10.1 Ejemplos de minimización a partir de distintos mapas de Karnaugh de 2, 3 y 4 variables

En este ejemplo los dos únicos "1" son adyacentes y se agrupan, ambos cubren toda la zona de \bar{b} ; esto significa que el valor que asume la variable a no tiene significado en el valor de la función y , por lo tanto, no se incluye en el circuito.

	b	b
	0	1
\bar{a}	0	(1)
a	1	(1)

Esta situación se puede demostrar fácilmente si se aplica a la expresión de la función una minimización algebraica:

$$f = \bar{a} \cdot \bar{b} + a \cdot b$$

$$f = \bar{b} \cdot (\bar{a} + a) \text{ de acuerdo con las leyes de complemento}$$

$$f_m = \bar{b} \cdot 1 = \bar{b}$$

	b	b
	0	1
\bar{a}	0	1
a	1	(1)

En este ejemplo no hay "1" adyacentes, por lo tanto, no es posible la minimización. La función se expresa como:

$$f_m = \bar{a} \cdot \bar{b} + a \cdot b$$

	c	\bar{c}	c
a	b	0	1
0	0		
0	1		(1)
1	1		
1	0		

En este ejemplo hay un solo "1", por lo tanto, no es posible la minimización y la función se representa

$$f_m = \bar{a} \cdot b \cdot c$$

	c	\bar{c}	c
a	b	0	1
0	0	(1)	(1)
0	1		
1	1	(1)	(1)
1	0		

En este ejemplo se agrupan los 2 únicos pares de "1" adyacentes. El grupo superior indica que las variables a y b deben estar negadas para que la función valga 1, sin importar el valor que asume la variable c . El grupo inferior indica que la función vale 1 si a y b no están negadas, sin importar el valor de la variable c . La función se expresa como la suma de los productos que definen ambos grupos: $f_m = \bar{a} \cdot \bar{b} + a \cdot b$

	c	\bar{c}	c
a	b	0	1
0	0	(1)	(1)
0	1	(1)	
1	1		
1	0		

En este ejemplo se deben agrupar los "1" adyacentes horizontales y luego los dos "1" adyacentes verticales. El "1" definido por $a \cdot b \cdot \bar{c}$ se agrupa nuevamente, por lo que este procedimiento resulta óptimo para la minimización, dado que el grupo vertical permite la eliminación de la variable b .

$$\text{La función mínima es } f_m = \bar{a} \cdot \bar{c} + \bar{a} \cdot \bar{b}$$

	c	\bar{c}	c
a	b	0	1
0	0	(1)	(1)
0	1	(1)	(1)
1	1		
1	0		

En este caso se observan cuatro "1" adyacentes que se deben tomar formando un solo grupo. Esta agrupación indica que sólo tiene significado para el valor de la función la variable \bar{a} , eliminando las variables b y c . La función mínima resulta $f_m = \bar{a}$

	c	\bar{c}	c
a	b	0	1
0	0	1	1
0	1		
1	1		
1	0	1	1

Esta matriz aparece contener dos grupos de "1" adyacentes (esta vez identificados por la zona marcada). Sin embargo, esto no es así; si se observa con detenimiento, ambos grupos pertenecen a la zona \bar{b} , por lo tanto, los cuatro "1" deben considerarse adyacentes, aun cuando la limitación de la figura plana (los "1" de los extremos no se tocan) pueda confundir. De este modo la función queda definida como $f_m = \bar{b}$, eliminando las variables a y c de la función.

	c	\bar{c}	c
a	b	0	1
0	0	1	1
0	1		
1	1		
1	0	1	1

Este es un ejemplo aún más complejo; el primero es un grupo horizontal de dos "1" adyacentes. Luego el segundo grupo es vertical, constituido por dos "1" en apariencia no adyacentes, y, además, reagrupa un "1" tomado anteriormente. Por último, el "1" definido por $a \cdot b \cdot c$ queda solo sin posibilidad de agruparse. La función queda expresada como: $f_m = \bar{a} \cdot \bar{b} + \bar{b} \cdot \bar{c} + a \cdot b \cdot c$

5.11 Resumen

Podemos considerar de interés para la comprensión de la representación de datos, por lo menos dos formas de representación de cantidades: la digital, que es la que nos compete, y la analógica, mencionada en el capítulo Evolución del procesamiento de datos. En el caso de las representaciones analógicas, podemos mencionar que una cantidad se representa por su relación proporcional con otra magnitud, por ejemplo, el voltaje de una tensión o el movimiento a escala de una sustancia (el caso típico y el que primero abordamos cuando somos pequeños es el termómetro, que cuantifica la temperatura del cuerpo en relación con el movimiento del mercurio en el tubo de vidrio). Lo más importante para indicar de las representaciones analógicas es que varían en forma continua dentro de un rango predeterminado.

Al "marcar" el tubo de vidrio del termómetro con pequeñas líneas estamos haciendo que la información sea discreta, porque como la temperatura del cuerpo humano es un valor que fluctúa en forma "continua" y el mercurio es un elemento que también lo hace, preferimos saber si nuestra temperatura es mayor que 37,5°C y, además, "cuantas líneas", para decidir si sólo nos damos un baño o es preciso tomar un medicamento y llamar al médico. Las mediciones "38" o "39" se denominan "digitales", si consideramos el término desde el punto de vista de los dígitos 3, 8 y 9.

O sea que los sistemas como el decimal, el octal, el hexadecimal y el binario, explicados en el capítulo Sistemas numéricos se utilizan para representar datos "discretos". En un avión habrá sistemas analógicos para "medir" altitud, presión y temperatura, y sistemas digitales para realizar cálculos con ellos, que pueden contemplar un margen de error predecible cuando una cantidad analógica se procesa de manera digital.

De todos los sistemas, el que mejor se adapta a la operación de componentes electrónicos es el binario. Por esta razón, nuestro mundo informático tiene millones de bits. Éstos se representan en el interior de las computadoras con dos rangos de voltaje, uno para el 0 y otro para el 1. Por ejemplo, se puede representar un "1" con un voltaje fluctuante en el entorno de 3,5 voltios y el "0" con un voltaje en el entorno de 0 voltios, o se puede representar un "1" con un voltaje fluctuante en el entorno 4,5 voltios y otro para el "0". En realidad, en el estudio de los sistemas digitales no interesa el valor de voltaje que representan los niveles lógicos "1" o "0", sino que lo que importa es si se interpreta como "0" o como "1". Es como marcar en el termó-

metro tres zonas por debajo de $3,4^{\circ}$ ("no hay fiebre") y por encima de $3,5^{\circ}$ ("hay fiebre"); la zona continua entre $3,4^{\circ}$ y $3,5^{\circ}$ es de transición y no se utiliza para la determinación de "no hay" y "sí hay", que por analogía serían el 0 o el 1 , respectivamente.

En un circuito digital o circuito lógico, así lo llamaremos en adelante, las entradas se interpretan como niveles lógicos, esto es, como 0 y 1 . Asimismo, las salidas también son niveles lógicos " 0 " o " 1 ", o sea que "entran" bits y "salen" bits. La relación que hay entre las entradas y las salidas constituye la "lógica" del circuito, de ahí los nombres de lógica digital, diagrama de lógica, lógica de circuitos y tantos otros. La lógica digital halla una forma de representación matemática en el Álgebra de Boole, donde las variables booleanas son las "entradas" del circuito y las funciones booleanas sus "salidas"; estas últimas no son más que "variables booleanas que dependen del valor de otras variables booleanas". Las formas normales de una función permiten la expresión algebraica que "manifiesta" la forma en que las variables se relacionan entre sí, utilizando los operadores booleanos, de manera que se pueda "dibujar el comportamiento del circuito" con compuertas lógicas y armar así el "diagrama de lógica" o "circuito lógico".

Por último, el método de mapas de Karnaugh permite simplificar las funciones utilizando un método visual, que es equivalente a la simplificación que se puede lograr al aplicar los teoremas y los postulados del Álgebra de Boole a la expresión algebraica. Esta simplificación permite reducir el número de compuertas lógicas y, eventualmente, la cantidad de entradas a ellas o al circuito. Esto se traduce en un menor número de componentes electrónicos, que "mejoran" la eficacia de su comportamiento al minimizar su complejidad.

5.12 Ejercicios propuestos

1) Represente la tabla de verdad de la siguiente función:

$$f = a \cdot b + a \cdot \bar{b} + \bar{b} \cdot c$$

2) Represente el diagrama lógico de la función f del enunciado anterior.

3) Dada la función $f = a \cdot b + \bar{a} \cdot \bar{b}$:

a) Representar el diagrama de lógica con compuertas AND, OR y NOT.

b) Representar el diagrama de lógica sólo con compuertas OR y NOT.

c) Representar el diagrama de lógica sólo con compuertas AND y NOT.

4) Dada la siguiente tabla de verdad represente la forma normal más conveniente para cada función:

a	b	c	f_1	f_2
0	0	0	1	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	0	1
1	0	1	0	1
1	1	0	0	1
1	1	1	0	1

5) Considerando $n = 3$ verifique que la suma de los minitérminos de una función de Boole para n variables es = 0.

6) Considerando $n = 3$ verifique que el producto de los maxitérminos de una función de Boole para n variables es = 1.

7) Infiera un procedimiento que generalice los enunciados de los dos últimos ejercicios.

8) Dada la tabla de verdad de las funciones f_1 y f_2

a) Represente la forma normal disyuntiva y la forma normal conjuntiva para cada una de ellas.

b) Represente los cuatro diagramas de lógica con compuertas NAND o NOR, según corresponda.

a	b	c	f_1	f_2
0	0	0	0	0
0	0	1	1	0
0	1	0	0	1
0	1	1	1	0
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



5.13 Contenido de la página Web de apoyo

El material marcado con asterisco (*) sólo está disponible para docentes.

Resumen gráfico del capítulo

Simulación

Herramienta interactiva que permite crear el diagrama lógico de una expresión booleana.

Animación

Cómo trabajan los interruptores no mecánicos.

Autoevaluación

Lecturas adicionales:

Álgebra booleana de José A. Jiménez Murillo, es parte del libro "Matemáticas para la Computación" de Alfaomega Grupo Editor (42 páginas). Agradecemos a su autor por permitir que su escrito sea parte de las lecturas complementarias de esta obra.

Video explicativo (01:44 minutos aprox.)

Audio explicativo (01:44 minutos aprox.)

Evaluaciones Propuestas*

Presentaciones*

6

Lógica digital

Contenido

6.1 Introducción	114
6.2 Circuitos lógicos de sistemas digitales	114
6.3 Circuitos combinacionales	114
6.4 Circuitos secuenciales	131
6.5 Resumen.....	141
6.6 Ejercicios propuestos.....	142
6.7 Contenido de la página Web de apoyo.....	143

Objetivos

- Comprender la lógica de circuitos combinacionales en el nivel de compuerta y bloque.
- Comprender la lógica de circuitos secuenciales en el nivel de compuerta y bloque.



En la página Web de apoyo encontrará un breve comentario de la autora sobre este capítulo.

6.1 Introducción

En este capítulo se desarrollará la metodología de diseño lógico de circuitos electrónicos, digitales, combinacionales y secuenciales, así como de dispositivos lógicos programables. Se muestran los componentes más utilizados y su descripción se ajusta a la norma ANSI. Es de destacar que el diseño y la implementación de estos circuitos corresponden a materias que los alumnos cursan normalmente en un semestre o en su modalidad anual, para alcanzar la profundidad que el tema requiere. Desde el punto de vista de la arquitectura de las computadoras, el tema sólo se incluyó como una referencia para los alumnos que no tuvieron las materias Técnicas digitales o Lógica digital en la currícula de su plan de estudio.



Recuerde que las compuertas son bloques que generan salidas con señales equivalentes a 1 o 0, cuando se satisfacen los requisitos de la tabla de verdad para cada combinación de entradas. Las diversas compuertas lógicas suelen encontrarse en sistemas de computadoras digitales. Cada compuerta tiene un símbolo gráfico diferente y su operación puede describirse por medio de la expresión algebraica de la función que las representa, o por las relaciones entre las entradas y las salidas representadas en una tabla de verdad.

6.2 Circuitos lógicos de sistemas digitales

Según lo visto, una expresión booleana permite la representación algebraica del valor de una función que representa la salida de un circuito. En los circuitos lógicos de sistemas digitales, los valores 0 o 1 se representan con dos niveles de tensión. Las variables de la función booleana se estudian como las entradas en el circuito para luego estudiar su comportamiento. Estos componentes de hardware se denominan circuitos lógicos de sistemas digitales y la representación de sus compuertas lógicas es otra forma de analizar la organización de los elementos electrónicos reales o sistemas electrónicos digitales.

6.3 Circuitos combinacionales

Cada compuerta es un circuito lógico combinacional en sí. De esta manera, se puede "armar" el esquema del comportamiento total del circuito sin necesidad de conocer los elementos electrónicos que lo constituyen (así como lo son la organización de varias de ellas). Los circuitos lógicos se clasifican en combinacionales o secuenciales. Un circuito combinacional o combinatorio permite que en las salidas se obtengan valores binarios "transformados" por la operación de las compuertas vinculadas en él y cuyo valor depende únicamente de los valores establecidos en las entradas. En un circuito secuencial se considera la idea de estado del circuito y, dado un estado actual y las entradas, se puede determinar un próximo estado del circuito y las salidas. Expresado de otra manera, los valores binarios de las salidas dependen no sólo de los valores binarios de las entradas sino también de, por lo menos, una de sus salidas.

Un circuito se representa a partir de un diagrama de bloque que es "una caja negra", donde se especifican las "n" entradas y las "m" salidas, así como, eventualmente, el nombre de la función que realiza, sin definir el esquema de compuertas internas. Se aclara que en todos los circuitos se eliminaron entradas adicionales dispuestas para la alimentación y el control del integrado (Vcc, GND...) y, que en este caso en particular, "n" es igual a "m".

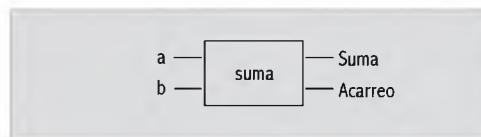


Fig. 6.1. Diagrama de bloque del circuito semisumador.

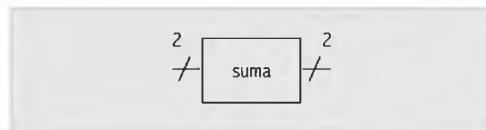


Fig. 6.2. Diagrama de bloque del circuito semisumador (otra opción).

Comenzaremos a analizar circuitos combinacionales. Para realizar el análisis se consideran todos los posibles valores que pueden darse en las entradas. La cantidad de combinaciones se calcula con la fórmula: para n entradas 2^n combinaciones posibles. Las n entradas representan los requerimientos de un problema, cada una puede asumir un valor verdadero (igual a 1) si el requerimiento se cumple, o falso (igual a 0) en caso contrario; por ejemplo, en el circuito semisumador, aquí identificado simplemente como "suma" en el capítulo anterior, el problema era resolver la suma de 2 bits representados ahora por a y b , que constituyen las entradas del circuito. Para cada combinación de entradas, se establece el valor de las salidas esperadas, por lo tanto, una de las formas de llegar a la expresión booleana y , así, a la organización de compuertas, es representar el problema en una tabla de verdad.

Tabla 6-1. Tabla de verdad			
<i>a</i>	<i>b</i>	Suma	Acarreo
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

A partir de la tabla para cada función de salida (suma y acarreo), se puede obtener la expresión como suma de productos (forma normal disyuntiva) o como producto de sumas (forma normal conjuntiva), o bien, si se aplica un método de simplificación, se obtiene una expresión mínima de la función o salida del circuito, que para ese caso son dos (suma y acarreo).

Recuerde que para el análisis y la simplificación de sistemas digitales binarios se utiliza como herramienta el Álgebra de Boole.

Niveles de descripción de un circuito combinacional

De menor a mayor nivel de abstracción encontramos, en primer lugar, la descripción del circuito electrónico; en segundo lugar, los diagramas de lógica; luego, la representación algebraica; y, por último, el nivel algorítmico de diseño de componentes. Éste describe su comportamiento utilizando como herramienta un lenguaje, como el VHDL (*Verilog HDL*): es un lenguaje de modelado de hardware similar al C que se presentó como de dominio público y se estandarizó con los estandares 1364-1995, 1364-2001, 1364-2005, <http://ieeexplore.ieee.org/xls/standardstoc.jsp?isnumber=33945>

6.3.1 Circuito generador de paridad

Un circuito generador de paridad permite la detección de error en la transmisión de información binaria entre un emisor y un receptor, cuando se asume que la probabilidad de error en la transferencia es lo suficientemente baja como para esperar que se produzca un error y no más. Ambos, emisor y receptor, se asocian a un circuito generador de paridad, de manera que el emisor envíe los bits de paridad. Si el circuito controla paridad, par quiere decir que contabiliza que la cantidad total de 1 en la transmisión es un número par. Si el circuito controla paridad impar significa que contabiliza la paridad impar de 1. Si se analiza el presente diagrama de lógica, se observa que el circuito no sólo permite generar la función "paridad par" de una cadena de 5 bits y la función "paridad impar" para la misma cadena, sino que también permite seleccionar qué tipo de paridad se utilizará.



Descripción de un circuito digital:

1. Algorítmico o de descripción de la función.
2. Álgebra de Boole, tabla de verdad y diagrama lógico.
3. Comutador de componentes electrónicos.

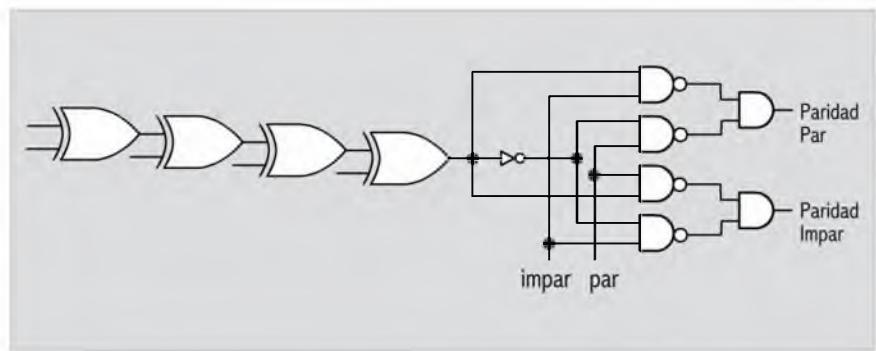


Fig. 6.3. Diagrama de lógica de un circuito generador de paridad.

Ejercicios

Siga la evolución para la combinación de entradas 00100011 en cada nivel de compuertas, primero para el caso de seleccionar paridad "ímpar", e indique qué ocurre con P_I , y luego para el caso de paridad par, e indique que sucede con P_P .

Investigue cómo sería un circuito verificador de paridad para una mensaje original de 3 bits. Represente su tabla de verdad y su diagrama de lógica.

6.3.2 Circuito comparador de magnitud

Es común que se necesite comparar dos números binarios de n bits para saber si éstos son iguales, o si uno es mayor o menor que otro. Podemos simplificar el problema analizando la comparación de 2 bits.

En este caso, es factible pensar en un circuito con dos entradas a y b , que representan los bits que se han de comparar, y tres funciones de salida que determinan: X si a es menor que b , Y si a es igual a b , y Z si a es mayor que b . El circuito se representa a partir de los elementos utilizados hasta el momento: diagrama de bloque, tabla de verdad y gráfico o diagrama de lógica.

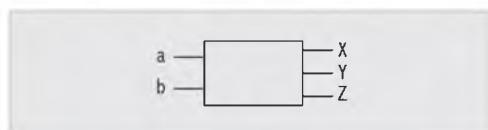


Fig. 6.4. Diagrama de bloque de un circuito comparador de magnitud.

Tabla 6-2. Tabla de verdad de un circuito comparador de magnitud

a	b	$a > b$	$a < b$	$a = b$
0	0	0	0	0
0	1	0	1	1
1	0	1	0	1
1	1	0	0	0

Diagrama de lógica

Las expresiones algebraicas que representan las funciones de salida son:

$$X = \bar{a} \cdot b \quad Y = a \cdot \bar{b} + \bar{a} \cdot b \quad Z = a \cdot \bar{b}$$

Por lo tanto, el diagrama de lógica o gráfico de compuertas que le corresponde es:

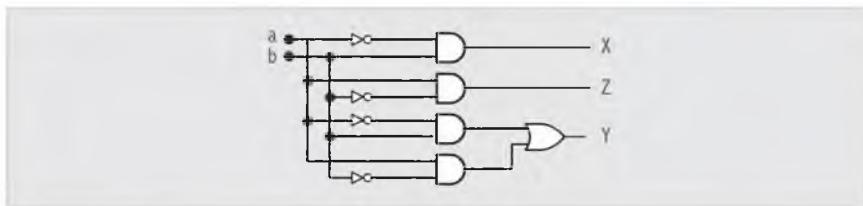


Fig. 6.5. Diagrama de lógica de un circuito comparador de magnitud.

6.3.3 Circuitos convertidores de código

Son circuitos cuya función digital es obtener en las salidas combinaciones binarias pertenecientes a alguna convención de representación de caracteres (numéricos o alfanuméricicos). Las entradas pueden ser señales que provienen de algún dispositivo de entrada de información (p. ej., señales de teclado) o pueden ser combinaciones binarias de otro código para convertir.

Ejemplo:

Un circuito que convierte un dígito BCD puro en su correspondiente valor BCD 2421.

Entradas BCD				Salidas BCD			
8	4	2	1	2	4	2	1
A0	A1	A2	A3	F0	F1	F2	F3
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	1
0	1	0	0	0	1	0	0
0	1	0	1	1	0	1	1
0	1	1	0	1	1	0	0
0	1	1	1	1	1	0	1
1	0	0	0	1	1	1	0
1	0	0	1	1	1	1	1
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X



Fig. 6.6. Diagrama de bloque.

Estas entradas faltantes, desde la combinación 1010 hasta la combinación 1111, no se van a dar nunca, porque no corresponden a combinaciones válidas de un dígito BCD y, por lo tanto, las salidas no están determinadas. Por eso las representamos con X y pueden ser consideradas individualmente como 0 o 1, según convenga al momento de simplificación.

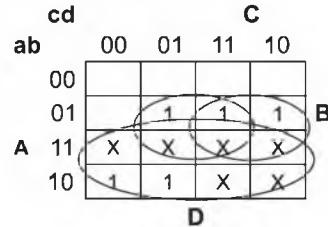


Fig. 6.7. Diagramas de simplificación (a, b, c, d) para la función F0.

En el diagrama se observan tres grupos de 1. En este caso las "X" fueron consideradas como 1 para optimizar la simplificación:

$$F0 = a + b \cdot c + b \cdot d$$

Una vez halladas las expresiones finales del circuito $F0, F1, F2$ y $F3$ podemos confeccionar el diagrama lógico que lo representa. Se muestra a continuación el correspondiente a $F0$.

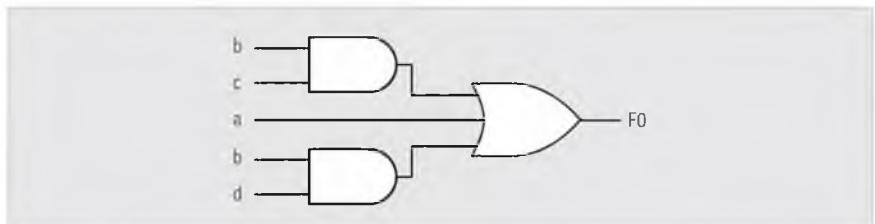


Fig. 6.8. Diagrama de lógica (parcial) de un circuito convertidor de código.

Todo circuito derivado de una expresión minimizada con un mapa tiene dos niveles de compuertas AND y OR (o sea que el nivel lógico de la entrada debe atravesar dos compuertas hasta alcanzar la salida).

Ejercicio:

Para completar el estudio del comportamiento del circuito, represente las funciones mínimas $F1, F2$ y $F3$, y sus respectivos diagramas de lógica, y compruebe que para las entradas 1001 corresponden las salidas 1111.

6.3.4 Circuitos codificadores

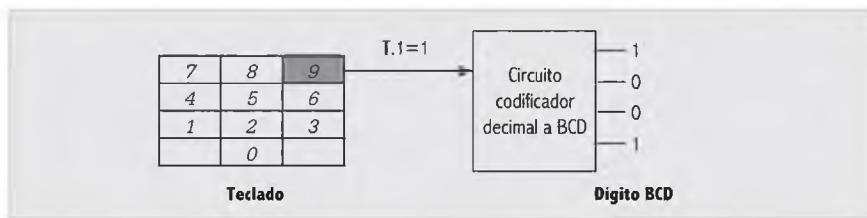
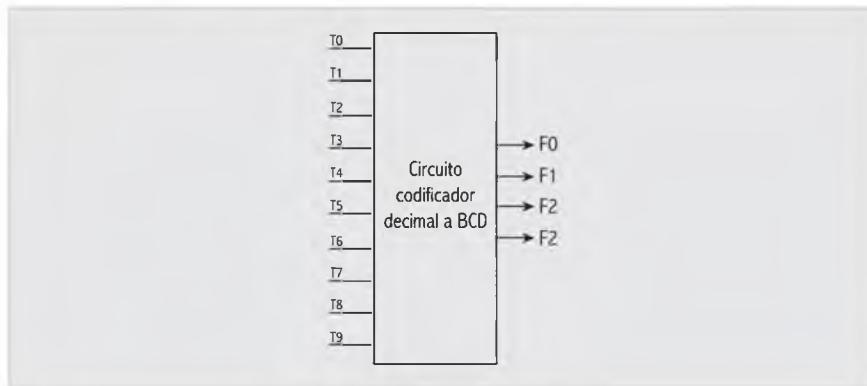
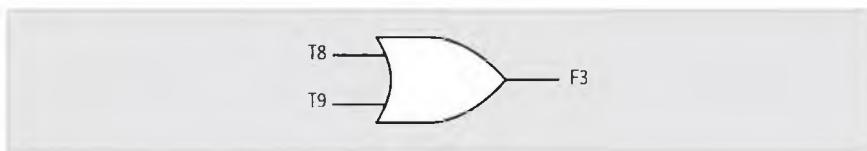
Veremos cómo las señales binarias provenientes de un teclado numérico pueden transformarse en salidas codificadas, por ejemplo, en BCD. Consideraremos teclas numéricas sin tener en cuenta las teclas de operación (+, -, *, etc.) comunes en todo teclado. Uno de los códigos más usados en la representación de números es el BCD 8421 o BCD puro. Si consideramos solo 10 teclas identificadas como $T9, T8, \dots, T0$ para los dígitos decimales de 9 a 0, debemos asumir que constituyen las entradas del circuito y, por lo tanto, la tabla de verdad deberá representarse con 2^{10} combinaciones posibles; sin embargo, la tabla de verdad muestra sólo 10 de las $2^{10} = 1024$ combinaciones posibles para 10 entradas, esto se debe a que se asume por simplicidad del circuito que no se presionan dos teclas en forma simultánea. En la tabla sólo se representan las combinaciones canónicas que indican que si se presionó una tecla, no se presionó ninguna otra. Esto permite transformar cada minitérmico, que en principio es el producto de todas las variables en juego, en una sola variable, ya que si esa variable es verdadera las demás son falsas.

Tabla 6-4. Tabla de verdad

Entradas que identifican las teclas											Salidas BCD			
T9	T8	T7	T6	T5	T4	T3	T2	T1	T0	F ₃	8	4	2	1
											F ₃	F ₂	F ₁	F ₀
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	0	1	0
0	0	0	0	0	0	1	0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0	0	0	0	0	1	0	1
0	0	0	1	0	0	0	0	0	0	0	0	1	1	0
0	0	1	0	0	0	0	0	0	0	0	0	1	1	1
0	1	0	0	0	0	0	0	0	0	0	1	0	0	0
1	0	0	0	0	0	0	0	0	0	1	0	0	0	1



Los terminales de entrada de un circuito digital aceptan señales binarias, dentro de las tolerancias permitidas, y los circuitos responden en los terminales de salida con señales binarias, que caen dentro de las tolerancias permitidas.

**Fig. 6.9.** Esquema de codificación de tecla a dígito BCD.**Fig. 6.10.** Diagrama de bloque.**Fig. 6.11.** Diagrama de lógica correspondiente a F3.

que se interpreta de la siguiente manera: F_3 "vale 1" cuando se presiona indistintamente la tecla T_8 o la tecla T_9 .

Las funciones de salida F_0, F_1, F_2, F_3 no se pueden minimizar por Karnaugh, porque cada función está definida en términos de más de 6 variables; sin embargo, vemos claramente que en cada caso podemos representarlas en forma algebraica de la siguiente manera:

$$F_0 = T_1 + T_3 + T_5 + T_7 + T_9$$

$$F_1 = T_2 + T_3 + T_6 + T_7$$

$$F_2 = T_4 + T_5 + T_6 + T_7$$

$$F_3 = T_8 + T_9$$

que se interpreta verbalmente como: F_0 se activa cuando se presiona la tecla T_1, T_3, T_5, T_7 o T_9 .

Ejercicio:

Observe que cuando se presiona la tecla 0 el valor de las salidas también es $0\ 0\ 0\ 0$, y cuando no se presiona ninguna de ellas se observa lo mismo. Como no se ha contemplado esta situación, piense qué entrada adicional agregaría para establecer la diferencia y grafique el circuito.



Aplicaciones de los decodificadores:

1. En chips de memoria, posibilitan el acceso *random*, convirtiendo el número que identifica la dirección de la celda de memoria en la fila y la columna.
2. En bancos de memoria, permiten la decodificación que identifica el banco.
3. Permiten la selección de dispositivos de E/S de microprocesadores.
4. Facilitan la decodificación de instrucciones en la CPU para activar señales de control.
5. Decodificador BCD-7 segmentos mostrado y codificadores de teclados (asignando valores binarios a códigos alfanuméricos o numéricos); por ejemplo, asignar la combinación 1010 para desplegar la condición E de "error".

6.3.5 Circuito decodificador de código

Opera en forma inversa al codificador en el siguiente sentido: la información que ingresa al circuito son combinaciones binarias pertenecientes a alguna convención de representación de caracteres (numéricos o alfanuméricos) y las convierte a señales binarias para representar en algún dispositivo de salida de información (por ejemplo: señales para video o para impresora). El circuito más simple que podemos analizar realizando esta función es el decodificador de BCD a un *display* de 7 segmentos muy utilizado en calculadoras. Se asume que para cada segmento, un "1" activa un diodo emisor de luz (LED o *Light-Emitting Diode*) y un "0" lo vuelve inactivo. El circuito recibe un dígito BCD y genera en su salida los 0 y 1 que "dibujen" el valor decimal en el *display*. Representamos en una tabla de verdad, todos los números tal como queremos que aparezcan:

Tabla 6-5. Tabla de verdad

A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	0	0	1	1
1	0	1	0	X	X	X	X	X	X	X
1	0	1	1	X	X	X	X	X	X	X
1	1	0	0	X	X	X	X	X	X	X
1	1	0	1	X	X	X	X	X	X	X
1	1	1	0	X	X	X	X	X	X	X
1	1	1	1	X	X	X	X	X	X	X

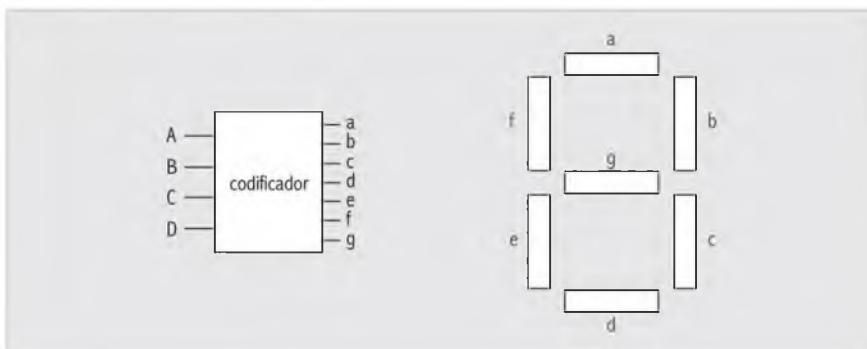


Fig. 6.12. Esquema de un display de 7 segmentos.

Ejercicio:

Para completar este circuito decodificador de código, es necesario hallar las funciones mínimas a, b, c, d, e, f, g y confeccionar el diagrama de lógica que incluye todas las funciones. Ejercítelo usted, considerando que el análisis e implementación de cualquier otro circuito codificador o decodificador se asemeja a los ejemplos dados.



Encuentre un simulador de un *display BCD de siete segmentos* en la Web de apoyo.

6.3.6 Circuito decodificador $n \cdot 2^n$

Existe otro tipo de decodificadores que permiten señalar en una de sus salidas qué combinación binaria se dio en la entrada. Los decodificadores de este tipo se denominan " $n \cdot 2^n$ " (se lee " n a la n " o " n a 2 a la n ") o "decodificador binario" y contemplan una salida activa para cada posible combinación de entradas. Esto quiere decir que una sola línea de salida obtendrá el valor 1, mientras que las demás permanecen en 0. Se puede decir que la combinación de las entradas "elige" una salida entre las demás y, por esta razón, las entradas también reciben el nombre de entradas de selección.



Encuentre un simulador de un decodificador de dos entradas en la Web de apoyo.

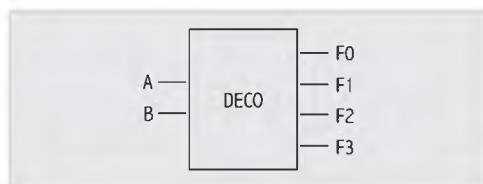


Fig. 6.13. Diagrama de bloque del decodificador $n \cdot 2^n = 2 \cdot 2^2 = 2 \cdot 4$ sin entrada de habilitación

La siguiente tabla representa las cuatro combinaciones posibles para las dos entradas. La salida F_0 señala con un 1 la combinación de entradas 00 , la salida F_1 señala con un 1 la combinación de entradas 01 , y así sucesivamente. El nivel lógico que identifica la combinación puede variar; lo importante es que ese nivel sea el opuesto de las otras tres combinaciones. Si se analiza cada función, se observa que en este caso tienen un solo "1" y, por lo tanto, no es posible la agrupación de 1 en un diagrama de simplificación.

Tabla 6-6. Tabla de verdad para un decodificador de 2 entradas y 4 salidas

A	B	F0	F1	F2	F3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

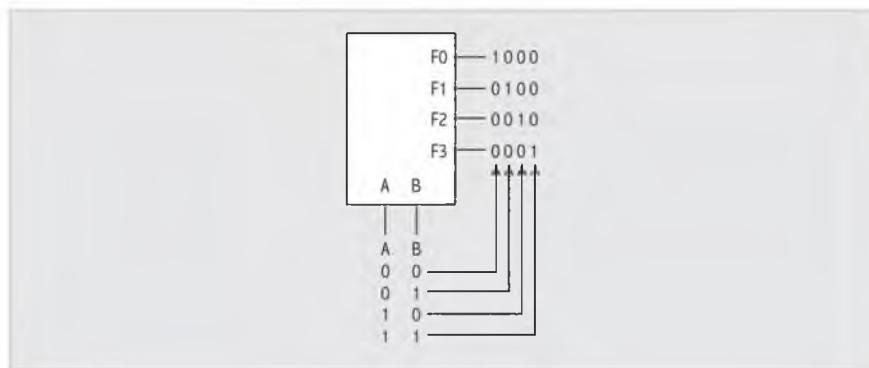


Fig. 6.14. Diagrama de bloque.

Un decodificador $n \times 2^n$ activa la salida correspondiente al número binario establecido en las entradas.

Luego, las funciones se representan algebraicamente con el minitérmino que corresponde al valor de entrada.

$$F0 = \bar{A} \cdot \bar{B}$$

$$F1 = \bar{A} \cdot B$$

$$F2 = A \cdot \bar{B}$$

$$F3 = A \cdot B$$

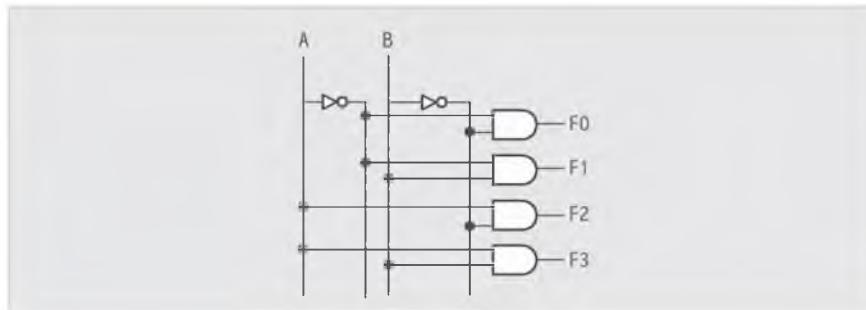


Fig. 6.15. Diagrama de lógica.

En el circuito anterior cada compuerta AND depende de dos entradas, pero puede tener una entrada adicional, asociada a cada AND, que permita controlar su operación (llamada entrada de habilitación). Si esta entrada, llamémosla *EN* (*enable*), es igual a 0, el circuito genera 0 en todas sus salidas, o sea, “no decodifica”, pues no señala la combinación dispuesta en la entrada; una habilitación de este tipo se denomina *strobe* o estrobo. Si *EN* es 1, entonces el circuito opera según lo que hemos visto.

En términos algebraicos, si *EN* = 1, entonces los minitérminos adquieren una nueva variable y las funciones se representan así:

$$F_0 = EN \cdot \bar{A} \cdot \bar{B} = 1 \cdot \bar{0} \cdot \bar{0} = 1 \cdot 1 \cdot 1 = 1$$

$$F_1 = EN \cdot \bar{A} \cdot B = 1 \cdot \bar{0} \cdot 0 = 1 \cdot 1 \cdot 0 = 0$$

$$F_2 = EN \cdot A \cdot \bar{B} = 1 \cdot 0 \cdot \bar{0} = 1 \cdot 0 \cdot 1 = 0$$

$$F_3 = EN \cdot A \cdot B = 1 \cdot 0 \cdot 0 = 1 \cdot 0 \cdot 0 = 0$$

con entrada de habilitación.

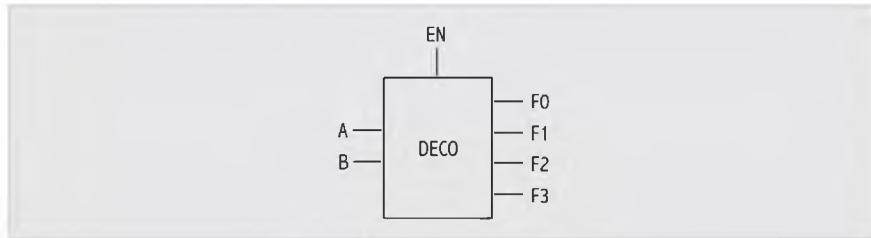


Fig. 6.16. Diagrama de bloque.

En las cuatro primeras combinaciones no decodifica.

Tabla 6-7. Tabla de verdad						
<i>EN</i>	<i>a</i>	<i>b</i>	<i>F₀</i>	<i>F₁</i>	<i>F₂</i>	<i>F₃</i>
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

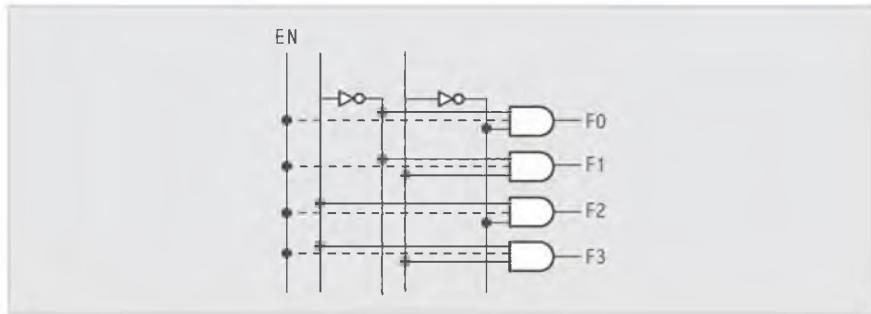


Fig. 6.17. Diagrama de lógica.

Genéricamente, un decodificador de n entradas se implementa con 2^n compuertas AND, cada una con n entradas. Un ejemplo de aplicación del circuito es el decodificador que se puede utilizar para el acceso, al seleccionar una dirección al azar en una memoria (acceso random).

La dirección coincide con el valor de una combinación en la entrada y permite indicar qué palabra va a ser leída o escrita. El decodificador de dirección habilita la palabra seleccionada e inhabilita las demás, de acuerdo con la dirección que reciba en sus entradas. Por ejemplo, si consideramos una memoria de cuatro palabras (ejemplo teórico) tendremos cuatro direcciones, empezando de 0 a 3. Las direcciones binarias son entonces, 00, 01, 10 y 11; la salida del decodificador habilitará con un 1 la palabra seleccionada e inhabilitará con un 0 las tres restantes.

Ejercicio:

Pruebe el circuito colocando una combinación a la entrada y siga su evolución para cada nivel de compuertas, por ejemplo, para la dirección de entrada 01, que indica que la palabra seleccionada es la segunda.

Los decodificadores pueden usar compuertas NAND o NOR, en vez de AND, y la salida representativa de la combinación se distingue con 0, es decir, la habilitación de la palabra se activa con 0 y las restantes con 1 (la deshabilitación de la palabra se activa con 1), ya que la salida es el complemento de los valores de las variables de salida de la tabla de verdad.

Ejercicio:

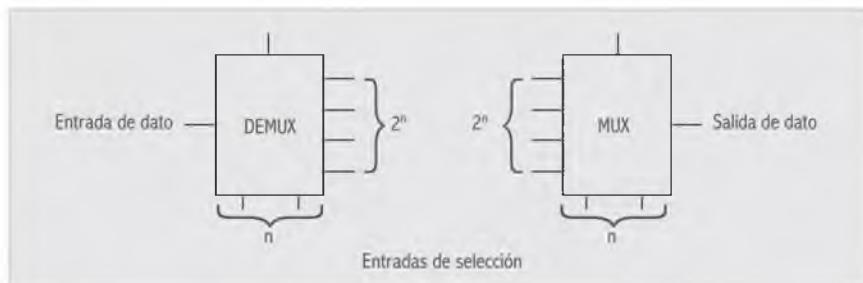
Para este caso represente la tabla de verdad y el diagrama de lógica.

6.3.7 Circuitos multiplexores y demultiplexores

Un demultiplexor es un circuito cuya función digital es "encaminar" la información que viene en una sola línea de entrada, en 1 de 2^n . Se puede pensar en un decodificador en el que la entrada de habilitación es la portadora del bit que se ha de "encaminar" y la selección de la línea de salida (habilitada por líneas de selección) determina qué salida es la encargada de transmitirlo. Por este motivo se lo conoce como decodificador/demultiplexor. Algunas de las aplicaciones de un demultiplexor son, por ejemplo, distribuir una entrada de datos en serie a una salida en paralelo o transferir bits de una línea de bus a un registro determinado; por eso también se lo denomina distribuidor.

Un multiplexor es un circuito combinacional cuya función digital es "encaminar" las señales binarias de 1 de 2^n líneas posibles de entrada en una única línea de salida. La línea de entrada de dato se "elige" a partir de los valores que puedan tomar las n líneas de selección. Ésta también se parece a un circuito decodificador $n \times 2^n$, donde las compuertas AND decodifican las líneas de selección de entrada, que se unifican utilizando una compuerta OR, cuya salida es el dato seleccionado. En sentido inverso, el multiplexor convierte una entrada paralela a una cadena en serie.

Por lo tanto, ambos se utilizan para transmitir datos a través de una sola línea en forma de serie, el multiplexor envía y el demultiplexor recibe. El número de entradas a un multiplexor o de salidas de un demultiplexor puede aumentarse usando más compuertas AND; para ese fin, se requieren más entradas o líneas de selección. Por ejemplo, con 2 entradas (A y B) se controlan 4 compuertas AND; con 3 se controlan 8, etcétera.



El multiplexor conecta una de las 2^n entradas a la salida. Esta entrada se selecciona con las n entradas de selección, o sea que permite realizar una función de commutación de n variables.

Fig. 6.18. Diagrama de bloque.

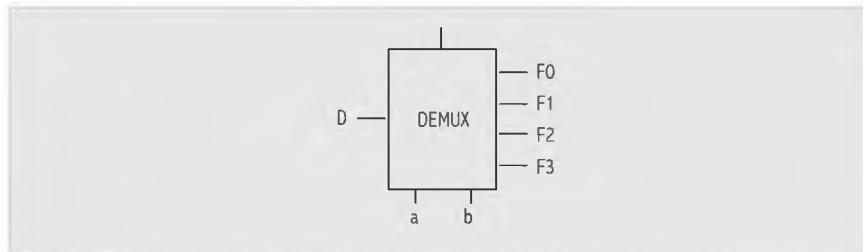


Fig. 6.19. Diagrama de bloque de un demultiplexor para una entrada de dato D, dos entradas de selección a y b y cuatro posibles salidas F0, F1, F2, F3.

Tabla 6-8. Tabla de verdad

a	b	F0	F1	F2	F3
0	0	D	0	0	0
0	1	0	D	0	0
1	0	0	0	D	0
1	1	0	0	0	D

El demultiplexor conecta la entrada con una de las $2n$ salidas, ésta se selecciona con las n entradas de selección.

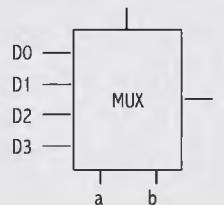


Fig. 6.20. Diagrama de bloque de un multiplexor para cuatro entradas de dato D0, D1, D2 y D3, dos entradas de selección a y b, y la única salida de dato F.

Tabla 6-9. Tabla de verdad

a	b	F
0	0	D0
0	1	D1
1	0	D2
1	1	D3

Aplicaciones de los multiplexores

1. Un MUX se puede implementar con cierto número de compuertas lógicas, porque básicamente es un decodificador con sus salidas asociadas a una única compuerta.
2. Los MUX de pocas entradas se utilizan en los dispositivos de lógica programables descriptos más abajo.
3. Asimismo, el MUX se utiliza para la conversión de bits transferidos en paralelo a serie, esto es, de a uno por vez, utilizando una sola línea.
4. La multiplexación de bits de dirección en memorias se emplea, por ejemplo, cuando un bloque de caché supera la capacidad del bus, si el bloque es de 128 bits y el bus de 64, entonces, primero se transfiere un grupo de 64 bits y luego el otro.
5. Se puede armar un módulo desplazador que permite mover bits a derecha e izquierda en registros de desplazamiento bidireccionales, bajo microoperaciones de control, por ejemplo, las que genera la unidad de control cuando se ejecuta una instrucción de desplazamiento (ver sección "Registros con facilidad de desplazamiento").

Ejemplo de una unidad aritmético-lógica (ALU) para 4 operaciones de tipo lógico utilizando un multiplexor. Supongamos las operaciones NOT (negación o complemento), AND (conjunción o producto lógico), OR (disyunción inclusiva o suma lógica), XOR (disyunción excluyente o suma exclusiva), que identificamos con los números 00, 01, 10, 11, respectivamente. Las señales de control indican cuál de las "instrucciones" pasan por la línea de resultado, al operar los bits *Op1* y *Op2*.

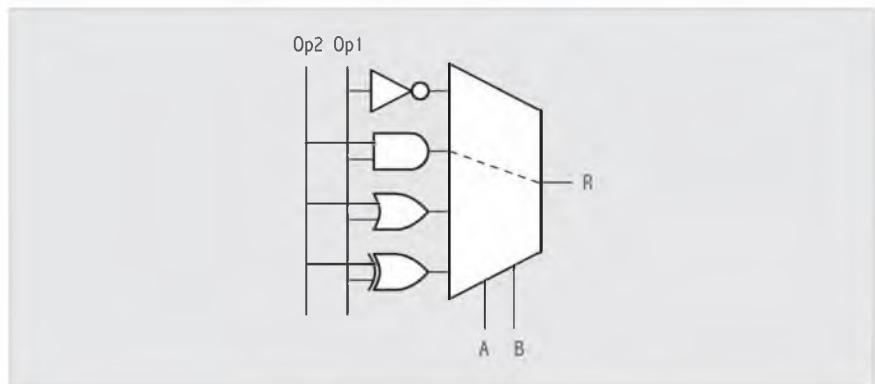


Fig. 6.21. Diagrama de lógica.

Para este ejemplo, $A = 0$ y $B = 1$, entonces, el resultado R es el producto lógico entre ambos operadores: $Op1 \text{ AND } Op2$.

6.3.7.1 Bus asociado a un multiplexor-demultiplexor

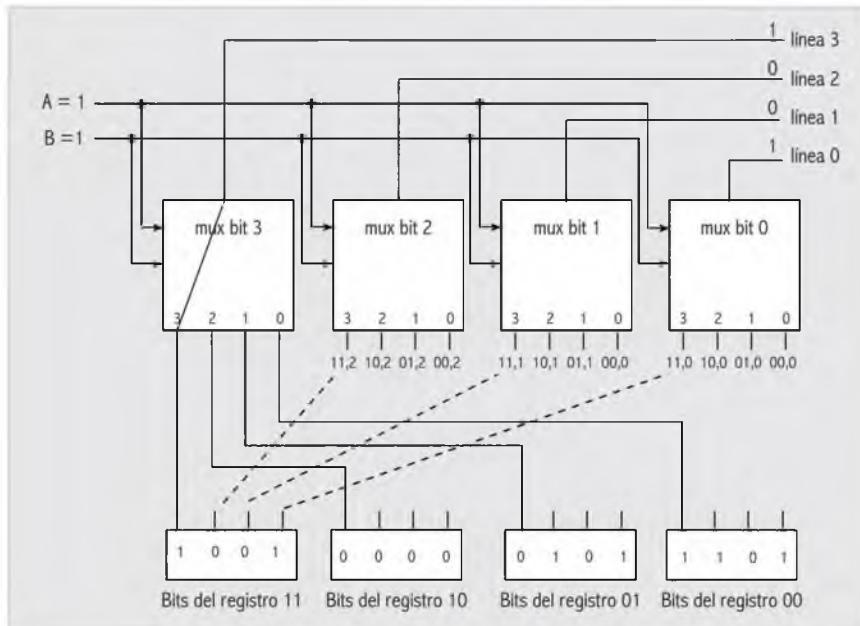
Un bus es un conjunto de conductores que permiten relacionar registros que actúan de emisores o de receptores de bits. En este caso se indica que los registros están asociados "a un bus común o bus único".

El vuelco de la información de los registros sobre el bus se realiza con circuitos multiplexores. Cada registro está asociado a un número de multiplexor y éste lo puede reconocer en sus entradas de selección. En el ejemplo de la siguiente figura los cuatro registros emisores posibles envían sus bits de mensaje sobre las n líneas del bus.

Cada multiplexor habilita el paso de una señal sobre el bus comandado por la dirección en sus entradas de selección. Así:

$$A = 1 \text{ y } B = 1$$

indican que D_3 es la línea de entrada seleccionada para salir por F .



Este esquema muestra cómo 4 registros de 4 bits cada uno, pueden transferir su información a través de un bus de 4 líneas. Éstas se identifican como **línea 3**, **línea 2**, **línea 1** y **línea 0**. Sobre las líneas de selección A y B , cada multiplexor recibe la identificación del registro que va a transferir, en este caso el registro identificado como 11. Así, el Mux de bit 3 se relaciona con los "bit 3" de cada registro, pero sólo el bit 3 del registro 11 es el "seleccionado" para ser transferido por la **línea 3**. Se debe pensar que los siguientes multiplexores operan de la misma manera, sólo que las relaciones están marcadas en el esquema en línea punteada para facilitar su comprensión. Cada señal de entrada al multiplexor se identificó con el número de registro y número de bit; o sea que 11,2 se debe interpretar como el bit 2 del registro 11.

6.3.8 Circuitos "programables" para múltiples funciones

Si se considera que las salidas de un decodificador $n \cdot 2^i$ son entradas a una compuerta OR, se puede diseñar una estructura de compuertas AND-OR sobre la cual "programar" una función digital. Para que el circuito resulte programable las salidas de las compuertas AND se asocian a un enlace electrónico, y éste, a su vez, a la entrada de la compuerta OR. Entonces, la lógica AND-OR no es fija, sino que puede programarse.

El proceso de programación de estos circuitos es físico y, por lo tanto, la mayoría de las veces es inalterable; el ejemplo presentado pertenece al tipo de circuitos conocido como PLD (*Programmable Logic Device*), que se utilizan para implementar funciones lógicas sin mayor análisis que el de la tabla de verdad y, también, como soporte de secuencias binarias fijas. O

Algunos tipos de PLD: ROM (*Read Only Memory*), PROM (*Programmable Read Only Memory*), EPROM (*Erasable PROM*), EEPROM (*Electrically Erasable PROM*), FLASH (subtipo de memoria EEPROM)

sea, "instrucciones" de programas en los que la secuencia de 0 y 1 esperada a la salida sea siempre la misma para cada combinación. Los dispositivos de lógica programable (PLD) son circuitos integrados con cientos de compuertas AND-OR que se relacionan por medio de enlaces electrónicos y conexiones fijas. Para la representación de sus diagramas de lógica en PLD se utiliza una simbología particular, por ejemplo, para representar una compuerta AND de tres entradas:



Supongamos que queremos programar la función digital que responde al circuito semisumador a partir de su tabla de verdad y que las variables de este circuito son las entradas a un decodificador 2 a 4.

Tabla 6-10. Tabla de verdad

a	b	Suma	Acarreo
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

En total hay 4 enlaces intactos y 4 destruidos; para programar el circuito se debe destruir el enlace, según indique la tabla de verdad, de la siguiente manera:

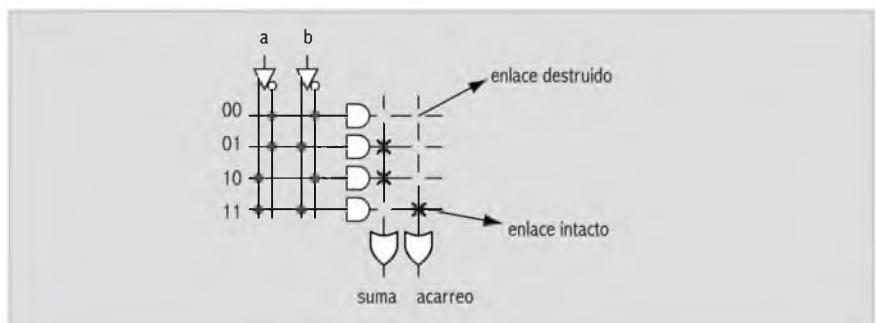


Fig. 6.22. Diagrama de lógica.

En la red OR las cruces representan la programación de 1 o enlace intacto. Los espacios vacíos representan la programación de 0 o enlace destruido. En la red AND los círculos rellenos representan uniones fijas o no programadas.

6.3.8.1 Memorias sólo de lectura

El arreglo de compuertas AND-OR, programado de esta manera, es un ejemplo simple de memoria ROM (*Read Only Memory*). Desde el punto de vista de los circuitos lógicos, forma parte de los dispositivos de lógica programable y se caracterizan por tener conexiones fijas en el arreglo de compuertas AND y conexiones programables en el arreglo de compuertas OR. Una vez establecida la combinación binaria de las salidas, éstas se mantienen inalterables.

Cuando una ROM se utiliza para implementar una función lógica, las entradas al decodificador AND constituyen las variables de la expresión booleana; cuando se utiliza como programa fijo,

constituyen la dirección de memoria para acceder. Cada línea de salida del decodificador se considera una palabra de memoria y cada enlace con un enlace (intacto o no) se considera una celda de "almacenamiento" de un bit. La cantidad de compuertas OR determina la cantidad de bits por palabra, y así se forma una matriz de $m \times n$ (m palabras de n bits cada una). Para identificar una de m palabras existe un valor p , tal que $2^p = m$, entonces p indica el número de entradas al decodificador. Este circuito puede tener entrada de habilitación y sus salidas de dato (salidas OR) pueden tener tres estados: "1" lógico, "0" lógico, o bien "alta impedancia" (este último estado se debe considerar "fuera de servicio"). Al inicio, las salidas de una ROM están todas en "1" (enlace intacto). La programación de los 0 es un procedimiento físico que genera la destrucción de los enlaces. Según las características de este procedimiento, se fabrican distintos tipos de dispositivos. En las ROM propiamente dichas, se confecciona la tabla de verdad con las combinaciones que se necesita en cada palabra; el proceso de fabricación incluye una máscara que representa la tabla. La grabación de esta máscara es el último paso en la fabricación del dispositivo. Según la aplicación, se pueden utilizar dispositivos tipo PROM (ROM-programable); en este caso el circuito viene "virgen" y puede ser programado individualmente. Hay ROM reprogramables llamadas EPROM (PROM borrible) o EEPROM (PROM borrible eléctricamente), que pueden utilizarse no sólo como soporte de un programa fijo, sino también como complejos codificadores. Una vez programada, la EPROM permite la restauración de los enlaces destruidos, razón por la que posibilita un nuevo ciclo de grabado, si bien los ciclos de borrado de escritura están limitados a la tecnología que se emplee.

Distribución de líneas en una memoria de sólo lectura

- La distribución de una memoria forma una matriz de $M \times N$ bits (M direcciones de N bits cada una).
- El bus de direcciones tiene p líneas, tal que $2^p = M$
- El bus de datos tiene N líneas para transferir los N bits leídos.
- El bus de control tiene una línea que habilita el chip y normalmente se denomina CS (*Chip Select*).
- En cualquiera de los ROM, la red AND es inalterable y la red OR es programable.

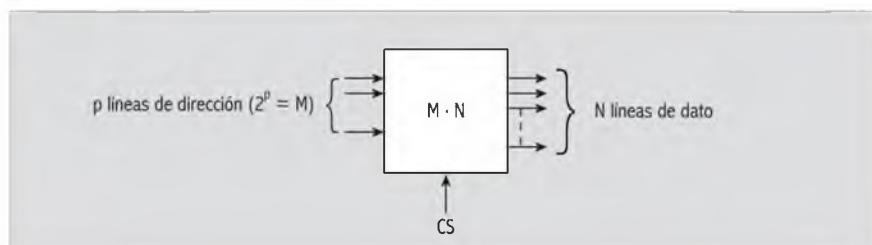


Fig. 6.23. Diagrama de bloque de cualquiera de las memorias de sólo lectura.

Diagrama de lógica de una ROM sin programar

Una memoria ROM no es más que otra forma de representar los minitérminos para cada función de p variables; así, en el diagrama siguiente se representan las ocho combinaciones posibles de tres variables (a, b y c) como enlaces fijos en una red AND que, acoplada a una red OR, podrá "programarse" para cada una de las dos funciones de salida F_0 y F_1 .



En las ROM, los bits (fijos) se almacenan en posiciones de memoria. Cada posición de memoria está asociada a una dirección.

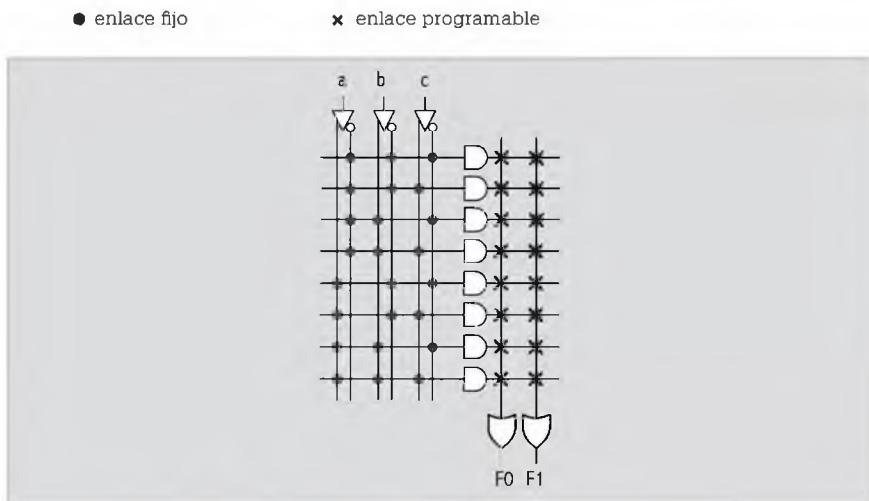


Fig. 6.24. Diagrama de lógica de una ROM sin programar.



En una ROM, las direcciones se indican por medio del bus de direcciones y los datos se leen desde el bus de datos.

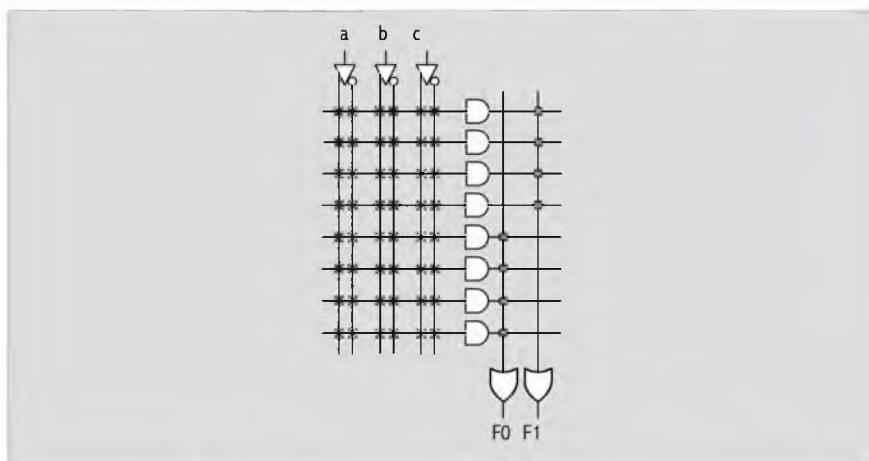


Fig. 6.25. Diagrama de lógica de un dispositivo tipo PAL.

6.3.8.3 Dispositivos tipo PLA o F-PLA (field-PLA)

Los dispositivos tipo F-PLA (*field* = campo) constituyen otra categoría de los dispositivos de lógica programable, y tienen aún mayor flexibilidad, dado que ambas redes son programables.

Diagrama de lógica para un PLA no programado, esta vez de tres entradas y tres salidas: F_0, F_1 y F_2

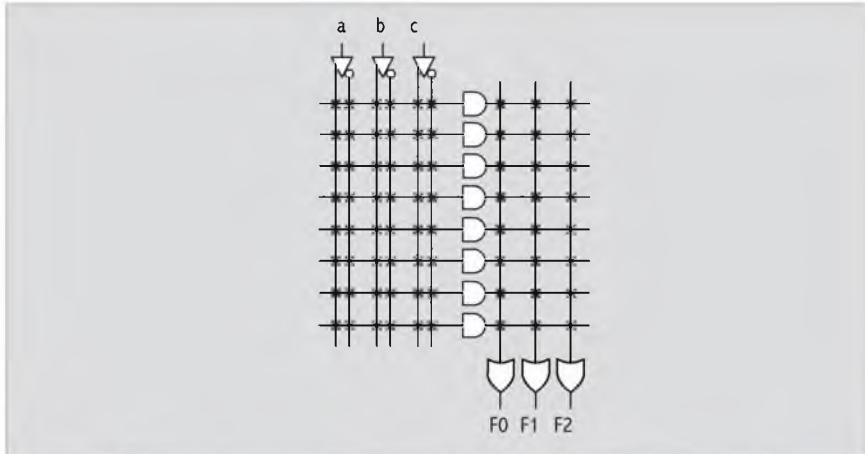


Fig. 6.26. Diagrama de lógica de un dispositivo tipo PLA o F-PLA (field-PLA).

6.4 Circuitos secuenciales

Un circuito es secuencial cuando sus salidas dependen del valor de las entradas y del estado anterior del circuito representado, que puede tener una o más salidas. Por lo tanto, la representación de estos circuitos debe tener en cuenta que las funciones de salida deben contemplarse como entradas; por ello se afirma que el circuito tiene uno o más lazos de realimentación.

En los circuitos secuenciales asincrónicos los cambios se producen únicamente por activación de alguna de las entradas.

6.4.1 Biestables o flip-flops

Un biestable es una celda binaria capaz de almacenar un bit. Tiene dos salidas, una para el valor del bit almacenado y otra que representa su complemento. Su denominación sugiere que el biestable tiene sólo dos estados posibles de funcionamiento, permaneciendo en cualquiera de ellos si los niveles lógicos de las entradas no fuerzan su cambio. El biestable es un arreglo de compuertas, caracterizado por tener lazos de realimentación que permitan mantener o memorizar el efecto de combinaciones anteriores en las entradas.

Analicemos el siguiente diagrama de lógica:



Si consideramos que F' es el valor de F después de aplicada la señal sobre S ($F' = F + S$), vemos que $S = 0$, y si $F = 0$ entonces F' permanece en 0 hasta que no se modifique el valor en S . Por lo tanto, podemos indicar que el circuito retiene o memoriza un 0.

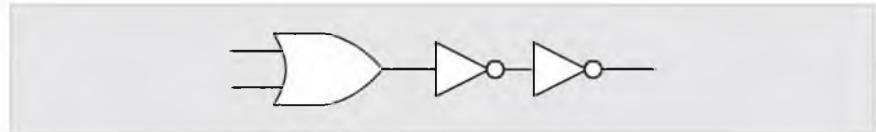
Si consideramos ahora $S = 1$ y $F = 0$, luego del tiempo en que la señal de entrada se propaga a través de la compuerta, F' cambia a 1 y podemos afirmar que, a partir de allí, el estado de F' permanecerá en 1 de manera indefinida, en forma independiente del valor de S . En consecuen-

Asincrónicos: los cambios sólo se producen cuando están presentes las entradas sin necesidad de una señal de reloj.

Sincrónicos: los cambios se producen cuando se establecen las entradas y, además, se genera una transición de señal de reloj.

cia, aunque este tipo de circuito retiene o almacena un bit, no podrá volver a 0 nunca y, por lo tanto, no sirve como "para ser escrito con 0" luego de haberse escrito con 1.

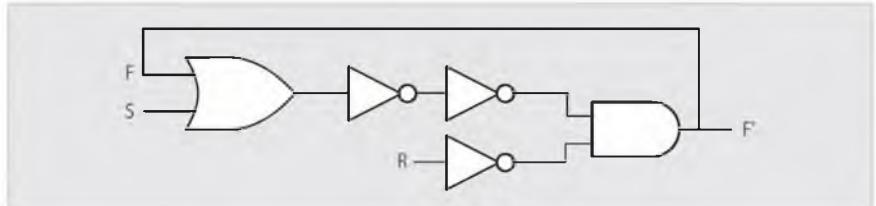
Primero observemos que si a una compuerta OR le acoplamos dos compuertas NOT en serie, la salida final no cambia el funcionamiento de la compuerta, como lo demuestra la tabla de verdad



A	B	$A + B$	$\text{no}(A + B)$	$\text{no}(\text{no}(A + B))$
0	0	0	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	0	1

Para que un circuito sea un elemento de memoria tiene que poder cumplir con dos condiciones: "retener un bit", aún después de la desactivación de las entradas, y poder ser "puesto a 0 o puesto a 1", independientemente del valor del bit almacenado con anterioridad. Una posibilidad de lograr el cambio del valor almacenado sería modificar el circuito original, agregando una puerta AND que permita el ingreso de una "línea de borrado" (en el diagrama, la línea R).

Analicemos el siguiente gráfico. Si ahora agregamos a esta salida una compuerta AND, que realimenta una de las entradas de la compuerta OR, si a la otra entrada de la compuerta la llamamos S de SET o puesta a 1 y si, a su vez, se agrega una compuerta NOT a la entrada –la otra entrada de la AND a la que denominamos R de RESET o puesta a 0–, obtenemos el circuito siguiente que cumple con las condiciones enunciadas:



La entrada F es el estado del bit antes de que se produzca un cambio.

La entrada S es la "acción de puesta a uno" o set.

La entrada R es la "acción de puesta a cero" o reset.

La salida F' es el estado del bit luego de activar las entradas S o R con un 1 en cualquiera de ellas;

Al adicionar la compuerta AND se ha logrado que el circuito "resetee" la salida o, lo que es lo mismo, se ha logrado su "puesta a cero". De este modo, si se analiza su funcionamiento, ahora con cada una de las combinaciones en las entradas y la correspondiente realimentación de la salida, se podría comprobar que su funcionamiento coincide con la siguiente tabla de verdad.

Tabla 6-11. Tabla de verdad

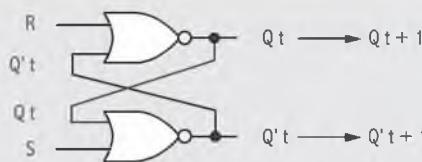
R	S	F	F'
0	0	0	0
0	0	1	1
0	1	1	1
0	1	1	1
1	0	0	0
1	0	0	0
1	1	0	X
1	1	1	X

Este circuito, tal como está graficado, se parece a un biestable llamado "flip-flop" o "biestable R-S". Para transformarlo como tal, podemos operar algebraicamente a partir de la doble negación y las leyes de De Morgan, de manera que se permita lograr ese circuito implementado con compuertas NOR:

6.4.1.1 Biestable R-S asincrónico

Tabla 6-12. Tabla de verdad

R	S	Q _t	Q _{t+1}
0	0	0	0
0	0	1	1
0	1	1	1
0	1	1	1
1	0	0	0
1	0	0	0
1	1	0	X
1	1	1	X

**Fig. 6.27.** Diagrama de lógica con compuertas NOR.

El gráfico del biestable R-S asincrónico reemplaza al expuesto antes, ya que resultan más visibles las interconexiones entre las dos compuertas NOR.

Siendo:

- R la entrada de reseteo o puesta a 0
- S la entrada de seteo o puesta a 1
- Q_t el almacenamiento del valor del biestable antes de ser activado
- Q_{t+1} el almacenamiento del estado posterior del biestable
- Q'_t el almacenamiento del complemento del valor del biestable antes de ser activado
- Q'_{t+1} el almacenamiento del estado posterior del complemento del biestable antes de ser activado

Debido a las distintas nomenclaturas aceptadas en diversos manuales, se debe considerar que la salida Q puede denominarse Q_t y que el estado posterior de Q puede denominarse indistintamente Q_{t+1} o Q' . De la misma manera, no Q es equivalente a \bar{Q} y, por lo tanto, \bar{Q}' es el estado posterior de \bar{Q} .

6.4.1.2 Biestable R-S sincrónico (temporizado)

En los sincrónicos o temporizados, la presencia de un pulso en una entrada especial determina el instante a partir del cual las entradas modifican su estado. Los pulsos de sincronismo son generados por un sistema de reloj y, por eso, llamaremos a la señal que lo representa Ck (*clock*).

La señal Ck oscila entre 0 y 1 a intervalos regulares, de manera que un biestable sincrónico es accionado cuando $Ck = 1$. Así, si varios biestables están relacionados entre sí por la señal de sincronismo, se modifican en el mismo instante.

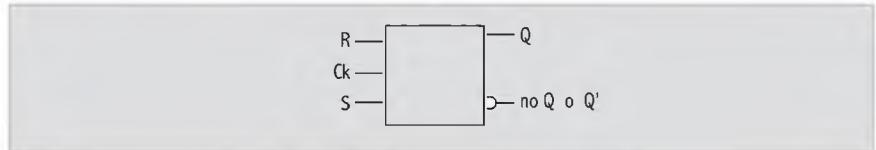


Fig. 6.28. Diagrama de bloque.

6.4.1.3 Biestable J-K sincrónico

El comportamiento del biestable JK es similar al del biestable RS , sólo que para las entradas J y K iguales a "1", las salidas se complementan según la siguiente tabla de verdad:

Tabla 6-13. Tabla de verdad de un biestable J-K sincrónico

J	K	Q_t	Q_{t+1}
0	0	0	0
0	0	1	1
0	1	1	0
0	1	1	0
1	0	0	1
1	0	0	1
1	1	0	1
1	1	1	0

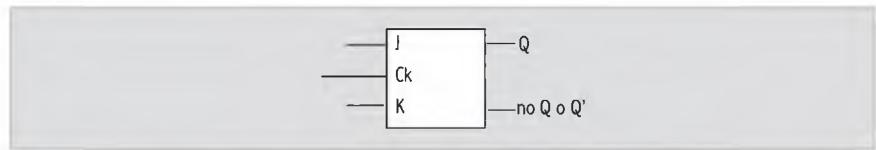


Fig. 6.29. Diagrama de bloque de un biestable J-K sincrónico.

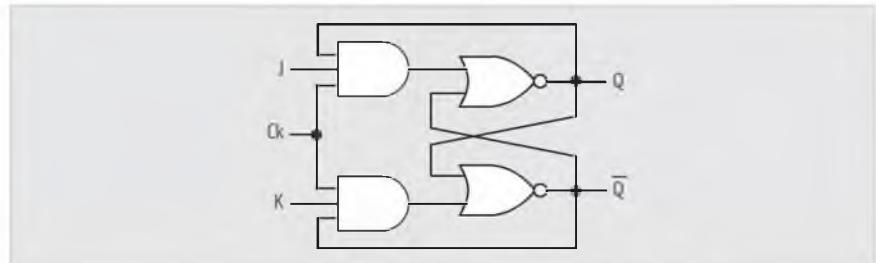


Fig. 6.30. Diagrama de lógica de un biestable J-K sincrónico.

Recuérdese que en la tabla la salida Q del diagrama es $Q(t)$ y que el estado posterior de la misma señal Q es $Q(t+1)$.

6.4.1.4 Biestable T sincrónico

El biestable T se puede analizar como un derivado del $J-K$, que se logra considerando que las entradas J y K son iguales, de manera que cuando éstas valen 00 o 11, se logra que se comporten según la tabla de funcionamiento siguiente:

Tabla 6-14. Tabla de verdad			
<i>J</i>	<i>K</i>	<i>Q t</i>	<i>Q t+1</i>
0	0	0	0
0	0	1	1
0	1	1	0
0	1	1	0
1	0	0	1
1	0	0	1
1	1	0	1
1	1	1	0

Tabla 6-15. Tabla de verdad		
T	Q_t	Q_{t+1}
0	0	0
0	1	1
1	0	1
1	1	0

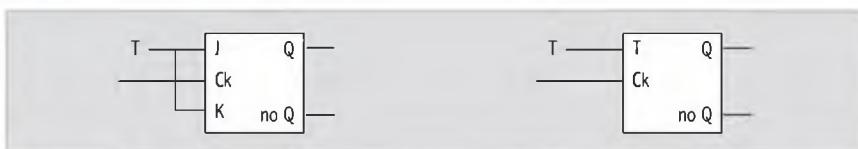


Fig. 6.31. Diagrama de bloques.

Fig. 6.32. Diagrama de bloques.

Al igualar las entradas $J-K$, el biestable sólo recuerda y complementa.

6.4.1.5 Biestable D sincrónico

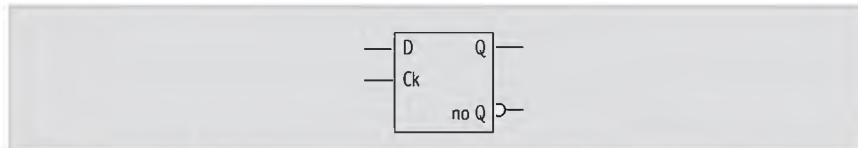


Fig. 6.33. Diagrama de bloque de un biestable D sincrónico

Aquí se utilizó un *R-S* sincrónico con entradas unidas para representar un biestable *D* sincrónico.

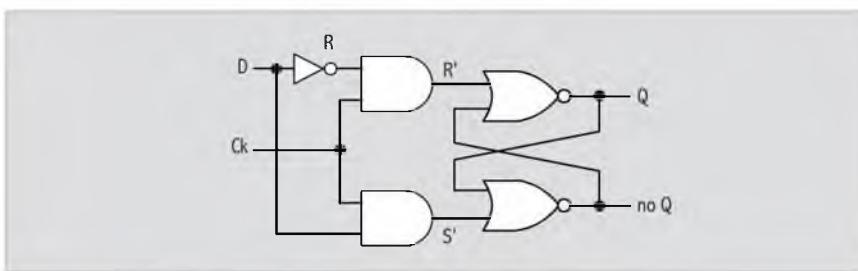


Fig. 6.34. Diagrama de lógica de un biestable D sincrónico

El biestable D también se puede analizar como un derivado del $J-K$, unificando las entradas J y K negada, de manera que cuando éstas son distintas (valen 01 o 10), se logra que se comporten según la siguiente tabla de verdad:

Tabla 6-16. Tabla de verdad		
D	Q_t	Q_{t+1}
0	0	0
0	1	0
1	0	1
1	1	1

6.4.2 Registros

Un registro es una función digital, que suele utilizarse para retener información binaria temporalmente. Puede estar formado por uno o más biestables. La CPU tiene una pequeña memoria de acceso rápido, formada por varios registros que almacenan información intermedia, parte de los cuales ya mencionamos como registros auxiliares (PC, IR, AC, SR, etc.).

6.4.2.1 Registros paralelo-paralelo

Algunos registros cumplen la función de transferir la información que viene por las líneas de entrada en paralelo a salidas en paralelo. Estos registros reciben el nombre de paralelo-paralelo. En este ejemplo se utilizaron biestables D .

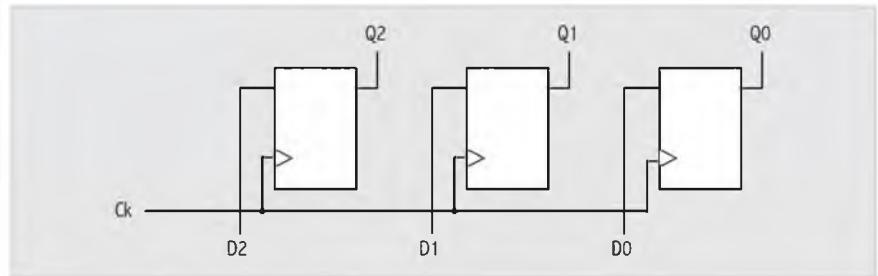


Fig. 6.35. Diagrama de bloques de un registro paralelo-paralelo.

6.4.2.2 Registros contadores

6.4.2.2.1 Contador progresivo de 8 eventos con biestables T

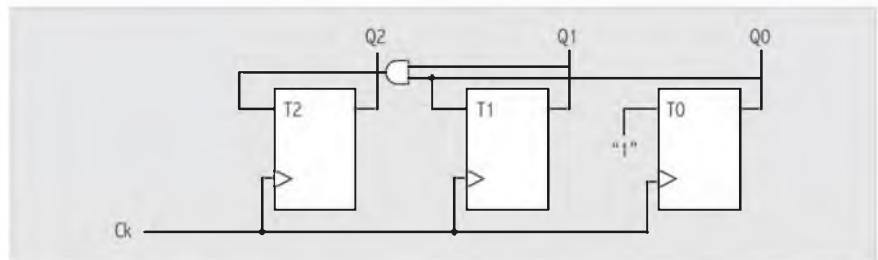


Fig. 6.36. Diagrama de bloques de un contador progresivo de 8 eventos con biestables T.

Como su nombre lo indica, un registro contador progresivo tiene la característica de contar eventos en orden ascendente, dependiendo de la cantidad de biestables que lo compongan. Si el registro está compuesto por n biestables, contará de 0 a $2^n - 1$, después de lo cual volverá a su estado inicial.

Para verificar su funcionamiento, veremos un ejemplo que se basa en un registro conformado por tres biestables "T" (T_0 , T_1 y T_2), donde la salida "Q" de cada uno de ellos ingresa en los biestables siguientes, relacionándose con compuertas AND, para posibilitar el conteo correcto. Cada uno de los biestables se activa en el mismo instante de tiempo C_k proveniente del reloj o temporizador. Este registro sólo puede contar hasta 8 (0-7) eventos, ya que tiene tres biestables.

Para seguir la tabla de funcionamiento de este registro contador progresivo de 8 eventos, confeccionados con biestables T , debemos partir de un estado inicial, donde tanto las salidas "Q" de los biestables como las entradas a los biestables T (T_1 y T_2) se encontrarán inicializadas a 0 (estas últimas por recibir la información de los biestables que las preceden). La entrada T_0 siempre recibirá un valor 1 (1). Por cada pulso de reloj, todos los biestables serán activados en forma simultánea, de acuerdo con el valor que posean en su entrada, teniendo en cuenta la tabla de funcionamiento del biestable T . Entonces, en el primer instante de tiempo, en el primer biestable, Q_0 está en 0 y recibe un 1 por T_0 ; por lo tanto, debe complementar el valor que tenía y dejar un 1 en **Q_0 antes del siguiente C_k** . En el segundo biestable, Q_1 está en 0 y recibe un 0 por T_1 **relacionado con Q_0 anterior**; por lo tanto, debe permanecer con el valor que tenía, 0. Asimismo, en el último biestable, Q_2 está en 0 y recibe un 0 de Q_1 y un 0 de Q_0 , que ingresan en una compuerta AND, cuya salida por el producto lógico es 0; por lo tanto, debe permanecer con el valor que tenía, o sea, 0.

Si completamos la tabla de funcionamiento por cada pulso de reloj, veremos, hasta ahora, que los valores obtenidos reflejan el cambio de 0, que era el estado inicial, a 1. Antes de que observemos cómo el registro se activa por segunda vez, debemos recordar que las salidas del pulso anterior ingresan en los biestables subsecuentes durante el nuevo pulso, como lo habíamos indicado antes.

Con el segundo pulso de reloj, en el primer biestable Q_0 hay un 1, y recibe un 1 por T_0 ; por lo tanto, debe complementar el valor que tenía y deja un 0 **antes del siguiente C_k** . En el segundo biestable, Q_1 está en 0 y recibe un 1 por T_1 **relacionado con el Q_0 anterior**; por lo tanto, debe complementar el valor que tenía y dejar un 1. Asimismo, en el último biestable, Q_2 está en 0 y recibe un 0 por Q_1 y un 1 por Q_0 , que ingresan en la compuerta AND, cuya salida por el producto lógico es 0, y este valor es el que ingresa por T_2 , provocando que el valor de Q_2 siga siendo 0.

Al completar la tabla de funcionamiento, observamos que los valores de salida de los biestables (Q_2 , Q_1 y Q_0) muestran un incremento de una unidad; en este último caso, 2 en binario. Si completamos el análisis de la tabla de funcionamiento, podremos observar que en el octavo pulso de reloj las salidas de los biestables vuelven a su valor original, o sea, 0 en binario.

Tabla 6-17. Tabla de funcionamiento del registro

C_k	Q_2	Q_1	Q_0	T_2	T_1	T_0
	0	0	0			
1	0	0	1	0	0	1
2	0	1	0	0	1	1

Ejercicio:

Complete la tabla hasta lograr que las salidas del registro vuelvan a 0.

6.4.2.2 Contador regresivo de 8 eventos (con biestables T)

Un registro contador regresivo, en forma inversa al anterior, tiene la característica de contar eventos en orden descendente, según la cantidad de biestables que lo compongan. Si el registro está compuesto por n biestables, contará de $2^n - 1$ a 0, después de lo cual volverá a su estado inicial ($2^n - 1$). Para verificar su funcionamiento, también veremos un ejemplo que está basado en un registro conformado por tres biestables "T" (T_2, T_1 y T_0), donde, en este caso y por ser regresivo, la salida " $no\ Q$ " de cada uno de ellos ingresa en los biestables siguientes, relacionándose con compuertas AND para posibilitar, así, el funcionamiento correcto.

Cada uno de los biestables se activa en el mismo instante de tiempo Ck proveniente del reloj o temporizador. Como tiene tres biestables, este registro sólo puede contar, en forma regresiva, de siete a cero (7-0). En forma análoga, para seguir la tabla de funcionamiento de este registro contador regresivo de 8 eventos confeccionados con biestables T, debemos partir de un estado inicial, donde las salidas " Q " de los biestables se encontrarán inicializadas con 1 y las salidas " $no\ Q$ ", por ser el complemento de las salidas Q , se encontrarán inicializadas con 0. Los valores de las salidas complementadas de Q serán los que ingresan en los biestables subsecuentes; por lo tanto, la tabla de funcionamiento se verá afectada de acuerdo con la evaluación que se encuentra más adelante.

La entrada T_0 siempre recibirá un valor 1 y, por cada pulso de reloj, todos los biestables se activarán en forma simultánea de acuerdo con el valor que posean en su entrada, teniendo en cuenta la tabla de funcionamiento del biestable T. A partir de estas premisas, el mecanismo de funcionamiento de este registro es similar al estipulado para el contador progresivo de 8 eventos, y no vale la pena desarrollarlo de nuevo. Sólo mencionaremos que es conveniente incorporar a la tabla de funcionamiento los valores complementados de las salidas Q de cada biestable para realizar un mejor seguimiento.

Ejercicio:

Dibuje un contador regresivo de 3 bits con biestables T, represente su tabla de funcionamiento y explique cómo funciona el registro para los dos primeros cambios. Utilice como referencia la siguiente tabla:

Ck	Q_2	Q_1	Q_0	$no\ Q_1$	$no\ Q_0$	T_2	T_1	T_0
	1	1	1	0	0			
1	1	1	0	0	1	0	0	1
2	1	0	1	1	0	0	1	1

6.4.2.3 Registros con facilidad de desplazamiento

En el set de instrucciones de una computadora hay instrucciones de desplazamiento, ya que la multiplicación de dos números se puede realizar sobre la base de sumas sucesivas y desplazamientos, al igual que las divisiones que, también, se pueden realizar sobre la base de restas sucesivas y desplazamientos.

Recuerde el mecanismo de desplazamiento de la coma al multiplicar o dividir por la base en el sistema decimal. En el sistema binario los desplazamientos se realizan corriendo los bits de un registro hacia la derecha o hacia la izquierda. Aunque los procesadores sencillos sólo disponían de operaciones de desplazamiento de una posición a derecha o izquierda, casi todos los procesadores actuales permiten el desplazamiento de varios bits en ambos sentidos.

Para que esto se produzca, las celdas binarias que forman parte del registro tienen que estar relacionadas de manera que, por cada orden que reciban, se produzca un desplazamiento de un bit, o sea que la salida de una celda alimente la entrada de la otra.

6.4.2.3.1 Desplazamientos lógicos

Cuando el bit del extremo queda libre y se rellena con un 0, se indica que el registro produce un desplazamiento lógico.

6.4.2.3.2 Desplazamientos circulares

Cuando en un registro de desplazamiento los n bits que se vacian en un extremo se completan con los que salen por el otro, se indica que el registro admite un desplazamiento circular. Con estos desplazamientos no hay pérdida de bits, sino que éstos circulan a través del registro.

6.4.2.3.3 Desplazamientos aritméticos

Los desplazamientos aritméticos afectan a números que pueden ser o no signados. Son parecidos a los desplazamientos lógicos, pero mantienen el signo del número. Estos desplazamientos producen una multiplicación o una división por una potencia de 2.

6.4.2.3.4 Desplazamientos concatenados

Son desplazamientos que afectan a un conjunto concatenado de dos o más elementos, que pueden ser:

- a) Dos registros.
- b) Un registro con el biestable de acarreo.
- c) Un registro con el biestable de signo.
- d) Combinaciones de algunos de los anteriores.

A continuación, se tratará el tema de la relación entre los desplazamientos y la rotación de bits en un registro, y las instrucciones que activan su operación. En cualquiera de los casos, los bits del registro pueden desplazarse o rotar su información dentro del registro o fuera de él.

Un desplazamiento mueve los bits del registro en forma lineal, mientras que una rotación lo hace de manera circular.

Las instrucciones que utilizaremos a modo de ilustración son:

SHL (*Shift Left*: desplazamiento lógico a la izquierda).

SHR (*Shift Right*: desplazamiento lógico a la derecha).

SAL (*Shift Arithmetic Left*: desplazamiento aritmético a la izquierda, para valores signados).

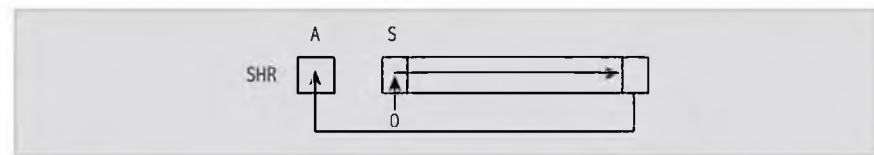
SAR (*Shift Arithmetic Right*: desplazamiento aritmético a la derecha, para valores signados).

Los desplazamientos lógicos no permiten conservar el signo del operando, razón por la cual para valores signados se deberán realizar desplazamientos aritméticos.

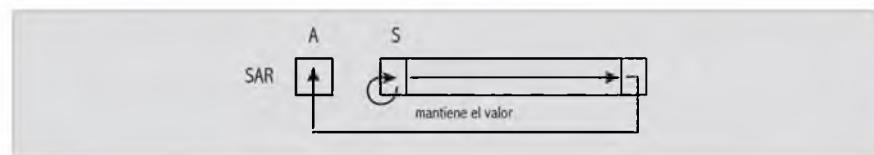
SHL y SAL desplazan todos los bits hacia la izquierda, menos el más significativo (bit 15 en el caso de una palabra y 7 en el caso de un byte), que se inserta en el bit de acarreo del registro de banderas (*status register*). A su vez, se coloca un 0 en el bit menos significativo (bit 0 en ambos casos), ya sea una palabra o un byte.



SHL desplaza todos los bits hacia la derecha, excepto el menos significativo (bit 0), que lo inserta en el bit de acarreo y coloca un 0 en el bit más significativo.



SAR desplaza todos los bits a la derecha, excepto el menos significativo (bit 0), que lo inserta en el bit de acarreo y no modifica el bit más significativo (bit de signo), o sea, mantiene el valor del signo.



El desplazamiento aritmético a izquierda multiplica el contenido del registro por la base, en este caso por 2^1 ; mientras que el desplazamiento aritmético a derecha divide el contenido del registro por la base. En el caso de la instrucción SAR, la división puede producir un resto, lo que provoca un redondeo "por defecto" del resultado.

Las instrucciones que permiten rotar un bit a la vez en un registro son:

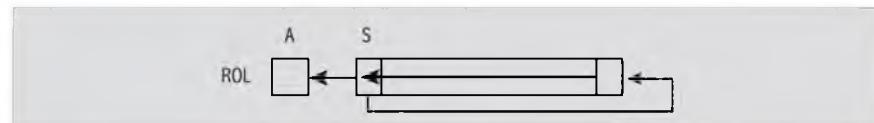
ROL (*Rotate Left*: rotación a la izquierda).

ROR (*Rotate Right*: rotación a la derecha).

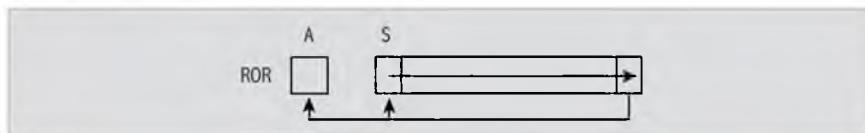
RCL (*Rotate Left Through Carry*: rotación con acarreo a la izquierda).

RCR (*Rotate Right Through Carry*: rotación con acarreo a la derecha).

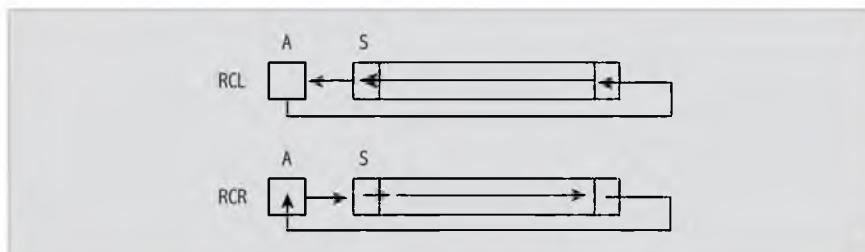
ROL rota los bits del registro, ya sea de 16 u 8 bits, a izquierda, colocando el bit más significativo en el bit de acarreo del registro de banderas y, también, en el bit menos significativo del registro.



ROR rota los bits del registro a la derecha, insertando el bit menos significativo en el de acarreo del registro de banderas y en el bit más significativo del byte o palabra.

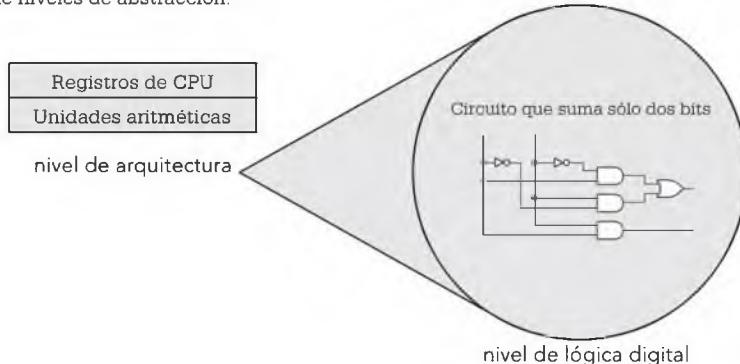


RCL y RCR utilizan el bit de acarreo del registro de banderas como una extensión del propio registro durante la rotación. Los bits de estas instrucciones rotan a izquierda o derecha, respectivamente, arrastrando con ellos al bit de acarreo.



6.5 Resumen

El estudio de la lógica digital permite la representación y el análisis de un componente definido como un bloque funcional en la arquitectura. En la figura siguiente se observa la diferencia de niveles de abstracción.



Los circuitos combinacionales son aquellos cuyas salidas dependen sólo de la combinación binaria establecida en las entradas; en el capítulo se incluyeron las funciones digitales más importantes para la comprensión del funcionamiento del hardware al nivel de abstracción de la arquitectura de una computadora. Así, se puede entender, por ejemplo, cómo se decodifica una dirección en un acceso *random*, si se considera la combinación de las entradas de un decodificador como la dirección de una posición en la matriz de memoria, y si se considera que cada una de las salidas del decodificador es la entrada de habilitación de cada una de las celdas que constituyen la posición direccionada.

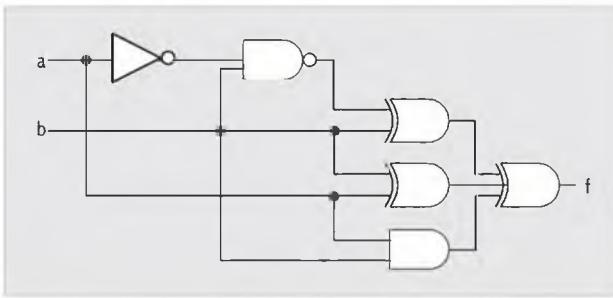
Los circuitos secuenciales son los sistemas digitales cuyas salidas no sólo dependen de sus entradas en un momento dado, sino también de su estado anterior. Se los puede pensar como un sistema combinacional con entradas independientes y la realimentación de una o más de sus funciones de salida. Las salidas adoptarán un “valor nuevo”, que dependerá de las

entradas y de "su valor anterior". En un sistema secuencial sincrónico, el instante en el que se "habilitan" las entradas se produce durante un pulso de reloj.

Los biestables son circuitos secuenciales básicos y permiten la construcción de circuitos secuenciales más complejos. Se utilizan para implementar matrices de memoria estática y registros de desplazamiento, contadores, de banderas, punteros, etcétera. Los registros son elementos de memoria de poca capacidad y alta velocidad, cuyo tamaño se expresa en bits. En una microarquitectura se estudian los registros asociados a la CPU no sólo para conocer su funcionamiento, sino también para utilizarlos con instrucciones del set de instrucciones que la CPU puede ejecutar. Estos registros se conocen como banco de registros de la CPU o registros del entorno de la programación de aplicaciones. Los registros pueden relacionarse entre sí por medio de un bus. La relación de cada registro con el bus se puede realizar utilizando un multiplexor para la transferencia de-registro-a-bus y con un demultiplexor para la transferencia de-bus-a-registro. Otra forma de establecer la transferencia es utilizar compuertas *buffer triestado* de manera que, en estado de alta impedancia, el registro pueda "desconectarse del bus".

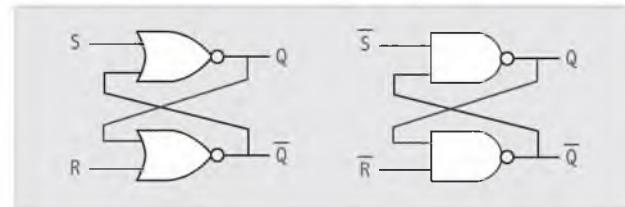
6.6 Ejercicios propuestos

- 1) Para el siguiente circuito:



- a) Halle la tabla de verdad.
 - b) Exprese la forma normal conjuntiva.
 - c) Exprese la forma normal disyuntiva.
 - d) Rediseñe el diagrama de lógica utilizando la forma normal disyuntiva.
 - e) Rediseñe el diagrama de lógica utilizando la forma normal conjuntiva.
 - f) Demuestre si los tres circuitos son equivalentes para la combinación de las entradas $a=0$ $b=1$ [recorrer cada diagrama con el valor binario que resulta a la salida de cada compuerta lógica hasta llegar a la función].
- 2) Diseñe un circuito que tome un número de 4 bits (a, b, c, d) y produzca una salida f que sea verdadera si la entrada presenta un número impar.
- 3) Diseñe un circuito que tome un número BCD de 4 bits (a, b, c, d) y produzca cuatro salidas f_0, f_1, f_2, f_3 en código Aiken.

- 4) Un circuito digital activa las cuatro luces de un semáforo. La combinación 00 activa la luz R (roja), la 01 activa la luz A (amarilla), la 10 activa la luz V (verde) y la 11, la luz G (flecha de giro). Represente la tabla de verdad con entradas a y b y salidas R, A, V, G y demuestre que este circuito en realidad es un decodificador.
- 5) Diseñe el diagrama de lógica del circuito anterior y agregue una entrada de habilitación para que, cuando ésta se encuentre en 0, apague todas las luces.
- 6) Analice el comportamiento de los siguientes circuitos y realice la tabla de verdad para cada uno de ellos



- 7) a. Implemente un flip-flop tipo T sobre la base de un J-K.
b. Realice la tabla de verdad del flip-flop tipo T.
- 8) a. Implemente un biestable tipo J-K sobre la base de un R-S y compuertas NOR.
b. Realice la tabla de verdad del flip-flop tipo J-K.
- 9) Diseñe el diagrama de un contador sincrónico progresivo que realice la secuencia de cuenta binaria de 0000 a 1111 y muestre su comportamiento en una tabla de verdad.

6.7 Contenido de la página Web de apoyo

El material marcado con asterisco (*) sólo está disponible para docentes.

Resumen gráfico del capítulo**Simulación**

Decodificador de dos entradas

Display BCD siete segmentos

Autoevaluación

Video explicativo (01:34 minutos aprox.)

Audio explicativo (01:34 minutos aprox.)

Evaluaciones Propuestas*

Presentaciones*

7

Diseño de una computadora digital

Contenido

7.1 Introducción	146
7.2 Módulo de cálculo en una computadora digital.....	146
7.3 Relación entre el diseño del hardware y la ejecución de instrucciones	147
7.4 Presentación del modelo de estudio	150
7.5 Resumen.....	176
7.6 Ejercicios propuestos.....	177
7.7 Contenido de la página Web de apoyo.....	177

Objetivos

- Incorporar el lenguaje técnico.
- Rescatar la importancia del lenguaje ensamblador; para conocer y programar a nivel hardware una computadora. Su inclusión obedece a la consideración de que la programación de máquina permite comprender de manera más eficaz el módulo más importante de la computadora.
- Conocer una máquina Von Neumann para comprender el funcionamiento de una computadora en relación con su diseño interno.
- Comprender la capacidad del hardware para interpretar una secuencia de instrucciones ordenadas lógicamente: el programa.
- Reconocer los componentes de una computadora y el set de instrucciones básicos de la máquina del ejemplo.



En la página Web de apoyo encontrará un breve comentario de la autora sobre este capítulo.

7.1 Introducción

El capítulo siguiente apunta a que se pueda comprender el funcionamiento de una computadora básica en relación con su diseño interno y demostrar la importancia del estudio del Álgebra de Boole y de la teoría de circuitos lógicos como elementos necesarios e imprescindibles para alcanzar ese fin. El modelo de computadora presentado es de una estructura muy sencilla, de manera que permite que se lo estudie en su totalidad. Para lograr entender la estructura interna de esta computadora se hará referencia a algunas instrucciones básicas del lenguaje de programación *Assembler*, que permitirán que se imagine el movimiento de las instrucciones o los datos de un programa a través de los registros internos que componen cada unidad de esta computadora. La verdadera magnitud de los sistemas actuales se desarrolla en capítulos posteriores a éste.

7.2 Módulo de cálculo en una computadora digital

El diseño de una computadora digital es la organización de módulos de hardware relacionados por rutas de control y datos. Su función es permitir el flujo de señales binarias para transformar datos de entrada en información útil al usuario de la computadora.

Un módulo está constituido por una configuración determinada de compuertas. En el diseño de lógica se hace abstracción de los elementos electrónicos que constituyen los circuitos; simplemente se asume que un circuito es una "caja negra" que cumple determinada función lógica. Lo que interesa es relacionar las cajas negras para lograr un producto que, ante determinadas entradas, presente las salidas esperadas.

Cada módulo realiza una o varias operaciones sobre datos codificados en sistema binario, que se almacenan en registros asociados al módulo mientras dura la operación. Una operación aplicada a un registro se denomina **microoperación** (**μop**) y se activa en un instante de tiempo sincronizado por los pulsos del reloj. El diseño de una computadora es más abstracto que el diseño de la lógica de un módulo y se ocupa de ensamblarlos.

Considérese como ejemplo de módulo de cálculo el sumador binario paralelo de la figura 7.1.

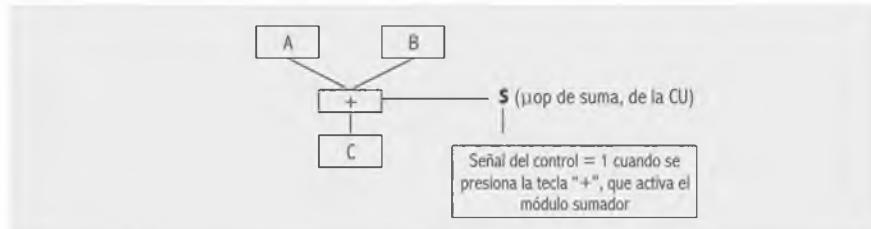


Fig. 7.1. Diagrama de bloque de un dispositivo que suma los bits de los registros A y B.

Su función es operar datos binarios para obtener en la salida el resultado de su suma. Los datos de entrada se almacenan en forma temporal en los registros A y B y, tras la orden de comando S, el resultado se obtiene sobre el registro C. La orden de comando es la microoperación de suma que habilita al registro C para que actúe de receptor del resultado. La orden puede ser una señal "1" generada por otro módulo, cuya función es "dar órdenes" en el caso de que este sumador pertenezca a una computadora.

7.3 Relación entre el diseño del hardware y la ejecución de instrucciones

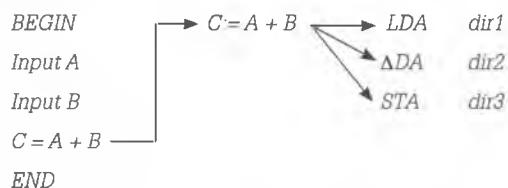
La programación en lenguaje de máquina implica el conocimiento del tipo de instrucciones en **código de máquina** (o **código nativo**) a las que obedece la computadora. Para simplificar la tarea de recordar secuencias binarias tediosas a la hora de programar, se ha desarrollado un lenguaje simbólico de bajo nivel conocido extensamente como **Assembler**. Cada computadora (y sus compatibles) tiene su propio Assembler, que tiene una íntima relación con su diseño. Ciertos autores consideran que su estudio no es muy necesario para abarcar el conocimiento de la computadora en su ámbito físico y lo declaran parte de las materias de programación. Su inclusión obedece a la consideración de que la programación de máquina permite comprender de manera más eficaz el módulo más importante de la computadora, el que entiende instrucciones y genera microoperaciones para su ejecución, llamado unidad de control (CU). Además, según se mencionó, permite el esclarecimiento de un concepto muy importante, que es la interacción entre lo físico y lo lógico, la capacidad del hardware como elemento que soporta e interpreta una secuencia de instrucciones ordenadas de manera lógica: el programa.

En síntesis, si en una computadora se ingresa un programa en lenguaje simbólico de alto nivel que indique:

```
BEGIN
Input A
Input B
C = A + B
Output C
END
```

este programa será convertido en programa ejecutable con la ayuda de un programa "traductor", donde para cada sentencia del programa fuente (o *source*) corresponderán n instrucciones en lenguaje de máquina. En este caso, las sentencias "**entrada A**" y "**entrada B**" solicitan el ingreso de las variables que se han de sumar (que aún no se describirán); la sentencia $C = A + B$ equivale a ordenarle a la computadora que sume la variable *A* a la variable *B* y guarde el resultado en la locación de memoria asignada para la variable *C*. La sentencia "**salida C**" permite mostrar el resultado en un dispositivo de salida.

Si, por el contrario, en la computadora se ingresara sólo la sentencia aritmética para el mismo programa pero en lenguaje simbólico de bajo nivel –como lo es el Assembler–, esto demandaría tres instrucciones para obtener el mismo resultado, en una relación 1 a 1 con las instrucciones en código de máquina:



Tanto el **lenguaje simbólico de alto nivel**, que es aquel que está estructurado de una manera más cercana al hombre, como el **lenguaje simbólico de bajo nivel**, que se generó de una forma más cercana a la computadora, permiten obtener las mismas instrucciones en **código de máquina**, o sea, las que el procesador "entiende" y puede ejecutar.

En la figura 7.2 se pueden apreciar estas relaciones.

Para un experto en Informática, dos buenas herramientas son el conocimiento de las características de diseño de su computadora y el aprovechamiento de las facilidades del sistema operativo, o sea, maximizar la eficiencia de la computadora.

La programación en lenguaje de máquina permite aumentar las prestaciones del sistema operativo, por ejemplo, al crear rutinas no definidas, que no podrían programarse en los lenguajes de alto nivel. Por lo tanto, debe considerarse satisfactorio el estudio de las instrucciones de máquina que entiende la CPU de su computadora. En este capítulo se creará un "set" o conjunto de instrucciones para el modelo propuesto, pero con otro objetivo: conocer, de la manera más simple, la forma en que estas secuencias binarias generan órdenes a los distintos módulos del hardware en un modelo de computadora básico.

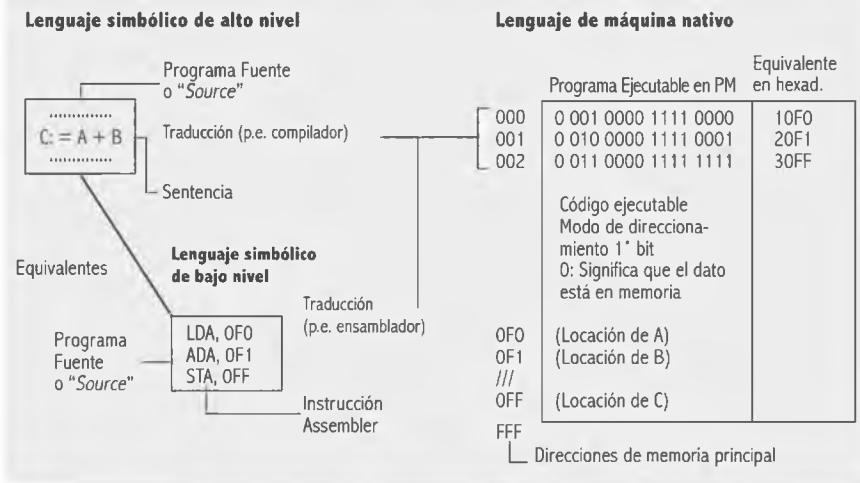


Fig. 7.2. Sentencia de ejemplo.

7.3.1 Instrucciones

Cuando la computadora realiza una tarea compleja, a pedido del usuario, ejecuta una serie de pasos simples representados por su propio juego de instrucciones. Estas instrucciones constituyen su lenguaje de máquina o lenguaje nativo. Como ya se indicó, no es usual que el programador plante la tarea en términos de secuencias binarias, sino que se utiliza un lenguaje simbólico más orientado a su modalidad de expresión que a la de la computadora. Sin embargo, todo programa que utiliza un lenguaje simbólico debe traducirse a código de máquina antes de su ejecución.

Por el momento no se entrará en detalle respecto de esta herramienta "que traduce" instrucciones simbólicas a instrucciones de máquina. Considérese la notación simbólica como una forma alternativa para representar instrucciones binarias, teniendo siempre presente que la computadora sólo ejecuta códigos de instrucción en lenguaje de máquina.

Así como se establece un código de representación de caracteres, unidades elementales que constituyen los datos (que ingresan, por ejemplo, por teclado), también hay un código de representación de instrucciones, unidades elementales que constituyen los programas.

El **código de una instrucción** es la combinación de bits que la unidad de control de la CPU interpreta para generar las microoperaciones que permitan su ejecución.

La forma de agrupar estos bits en entidades diferenciadas determina la estructura de la instrucción y se define como **formato de la instrucción**. Una misma unidad de control puede "comprender" distintos formatos de instrucción.

El formato de instrucción más simple es el que asigna un grupo de bits para representar una "acción" y otro grupo para representar el "dato" al que afecta esta acción. Como se ve en la figura 7.3, el primer grupo de bits se denomina código de operación (**OPCODE**). La cantidad de bits del COP determina el número de acciones distintas que se podrían definir, según la fórmula siguiente:

"n bits" determinan " 2^n códigos de operaciones distintos".



Fig. 7.3. Formato de instrucción.

Por lo general, el segundo grupo de bits hace referencia a un dato en memoria, por lo tanto, determina la dirección de la posición de memoria (locación) donde se aloja el dato. La cantidad de bits debe permitir hacer referencia a cualquier posición de memoria.

Sin embargo, no siempre los datos se encuentran en memoria. Un dato puede estar almacenado en un registro de CPU y, en este caso, los bits de dato deben poder hacer referencia a ese registro. Incluso hay códigos de operación que no afectan dato alguno y, si es así, el segundo grupo de bits puede aparecer como redundante o tomarse como una extensión del código de operación.

Según se indicaba antes, una computadora de propósito general tiene definida la tarea que se ha de realizar según las instrucciones de un programa almacenado en memoria, que es intercambiable. La memoria de lectura/escritura está dividida lógicamente en memoria asignada a programa y memoria asignada a datos y constituye el módulo de almacenamiento de la computadora. La unidad de control lee una instrucción de la memoria, la aloja en un registro interno (que llamaremos en este capítulo registro de instrucción) e interpreta si el código de operación afecta a un dato almacenado en memoria, en cuyo caso provoca su lectura. Cuando una instrucción está alojada en la unidad de control se afirma que está en **estado de ejecución** y su código binario indica dónde está el dato, cuál es la acción que lo afecta y, por lo tanto, qué módulo del hardware la llevará a cabo.

Conocer la naturaleza del juego de instrucciones de máquina es una de las mejores formas de aprovechar (con programas más simples de ejecutar) la potencia de la computadora y comprender la relación entre los módulos que la constituyen.

En ocasiones se hace difícil encontrar un límite entre las capacidades del hardware y las del software, ya que algunas funciones no definidas por uno pueden ser provistas por el otro. Los diseños actuales muestran la tendencia para desarrollar la mayor cantidad de funciones sobre hardware, porque esto incrementa la velocidad de procesamiento, acompañada de un declive constante de su costo. Hasta principios de la década del 80, la tendencia era hacer el hardware más complejo (más funciones en hardware); desde aquellos años hasta hoy, se usa un criterio cuantitativo, que indica que se implementan en hardware las funciones que se utilizan en forma más frecuente y el resto se implementan en software.

Cada computadora está diseñada para abastecer las necesidades de un grupo determinado de usuarios en el mercado total, lo que implica que las bondades y las limitaciones de un diseño siempre son características relativas a las aplicaciones para las que sirven mejor. Computadoras muy buenas para aplicaciones comerciales pueden ser bastante inútiles para aplicaciones científicas. Organizar el diseño del hardware de la computadora y de su software de base es una empresa cuyo objetivo es lograr la mayor eficiencia a menor costo en el mercado al que apunta.



Von Neuman (1903-1957). Matemático húngaro-estadounidense que realizó contribuciones importantes en Física cuántica, análisis funcional, teoría de conjuntos, Informática, Economía, análisis numérico, Hidrodinámica, Estadística y muchos otros campos de la Matemática. Fue pionero de la computadora digital moderna y publicó un artículo acerca del almacenamiento de programas. El concepto de programa almacenado permitió la lectura de un programa dentro de la memoria de la computadora y, después, la ejecución de las instrucciones del mismo, sin tener que volverlas a escribir. EDVAC (*Electronic Discrete Variable Automatic Computer*) fue la primera computadora que usó este concepto desarrollado por Von Neumann, Eckert y Mauchly. Los programas almacenados dieron a las computadoras flexibilidad y confiabilidad, haciéndolas más rápidas y menos sujetas a errores que los programas mecánicos.

7.4 Presentación del modelo de estudio

La presentación del material aquí expuesto es de lectura necesaria si se desea conocer un diseño completo, aunque elemental, de una computadora. La computadora presentada no es un modelo concreto del mercado actual, sino que fue pensada para que usted se acerque lo suficiente, y de la forma más simple, a las características de diseño de la mayoría de las computadoras. La disposición esquemática de los componentes no tiene relación con su emplazamiento físico real; el tamaño de los bloques en cada figura no sigue una escala fija, incluso el tamaño mayor de alguno de ellos respecto de los demás, sólo pretende resaltar sus características.

Se decidió llamar "X" a esta computadora de propósito general, cuyo diseño se observa en la figura 7.4 y encuadra en los parámetros definidos para una arquitectura Von Neumann.

"X" tiene un módulo de almacenamiento de lectura/escritura de 4 Kpalabras de 16 bits cada una, con acceso aleatorio a cada palabra identificada por una dirección de 12 bits que se notará siempre en hexadecimal, de modo que el rango de direcciones varía entre 000H y FFFH (0-4095). La memoria está dividida "lógicamente" en memoria asignada a programa y memoria asignada a datos. "X" puede ejecutar un programa por vez, o sea que su modalidad de procesamiento es la monoprogramación.

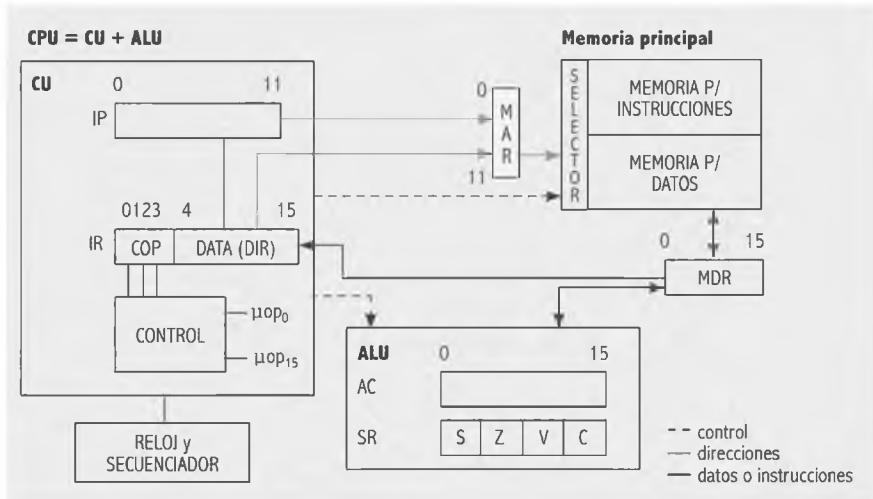


Fig. 7.4. Arquitectura básica de "X".

El modelo plantea cómo evoluciona la ejecución de las instrucciones sin mencionar cómo llegaron a la memoria desde el exterior.

Formato de datos:

Los datos son del tipo enteros signados de 16 bits (1 para el signo y 15 para la magnitud), según se puede observar en la figura 7.5.

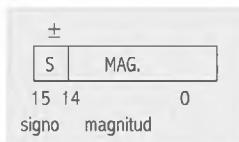


Fig. 7.5. Formato de datos.

Formato de instrucción:

Las instrucciones se almacenan en palabras consecutivas a partir de la palabra $000H$. El código de instrucción es de 16 bits, con un formato único de instrucción, el más simple, representado en la figura 7.6, donde los primeros 4 bits definen el código de operación, y los 12 restantes definen la dirección de un dato (en memoria asignada a datos), de lo contrario, son ignorados.

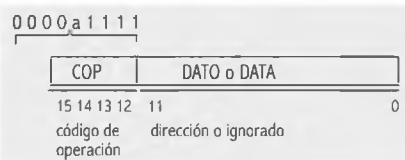


Fig. 7.6. Formato de instrucción de la CPU de "X".

La CPU de "X" (unidad central de procesamiento) se encarga de buscar las instrucciones de la memoria, interpretarlas y gestionar su ejecución, mientras que los datos son operados aritméticamente o lógicamente en la ALU.

La función de la CPU se puede separar en dos partes:

1. Tratamiento de instrucciones
2. Operación de los datos

Del tratamiento de instrucciones se encarga un conjunto de componentes denominado **unidad de control** (CU o *Control Unit*), sincronizado por el generador de pulsos de reloj, y de la operación de los datos se encarga, como se indicó, la **unidad aritmético-lógica** (ALU o *Arithmetic Logic Unit*). De modo que en la figura 7.7 se muestra un esquema simplificado de "X".



Fig. 7.7. Esquema simplificado de "X".

La memoria de "X" no necesita más detalle del que se aportó. El medio de comunicación entre la memoria y la CPU es un bus que transmite órdenes y datos, interpretando como bus de datos el camino que permite la transferencia de grupos de bits que identifican el contenido de una posición de memoria.

La CPU merece una dedicación especial, ya que su función es gestionar, precisamente, el procesamiento de información. Consideremos primero cómo toma la CU las instrucciones del programa almacenado en memoria para interpretarlas y ejecutarlas. Este proceso se puede dividir en las etapas siguientes:

1. Búsqueda de la instrucción en memoria.
2. Interpretación del código de instrucción.

3. Búsqueda del dato afectado (si afecta a dato) por la instrucción

4. Generación de órdenes al módulo que opera sobre ese dato.

La etapa 1 también se denomina **fase de búsqueda** o **fase fetch**, mientras que las otras tres se agrupan en la llamada **fase de ejecución** o **execute**. Se puede afirmar que una vez que la computadora comienza a funcionar, su CPU se encontrará siempre en una de estas dos fases. Para organizar el estudio de estas etapas, la atención se debe centrar primero en qué elementos intervienen en la gestión y luego en cómo operan para llevarla a cabo.

7.4.1 Fase fetch: búsqueda de una instrucción en memoria

Cuando la CU ejecuta cada instrucción de un programa debe alternar sus etapas *fetch* (*f*) y *execute* (*e*) desde la primera instrucción hasta la última, esto es, $f_{(0)} e_{(0)}; f_{(1)} e_{(1)}; \dots; f_{(n)} e_{(n)}$.

La secuencia del ciclo $f_{(0)} e_{(0)}$ se denomina **ciclo de instrucción**

Para cada fase $f_{(0)}$ la CU debe enviar a la memoria la dirección de la palabra donde se encuentra la instrucción, una orden de lectura y una orden de transferencia de la instrucción a la CU. La CU retiene la dirección de la instrucción en un registro especial denominado "puntero de instrucción" (IP o *Instruction Pointer*) o "contador de programa" (PC o *Program Counter*). Considerese este registro como el señalador de páginas del libro que usted está leyendo; el señalador le permite cerrar el libro en forma distraída y luego retomar en la página correcta. El IP cumple la misma función al permitir que la CU "se distraiga" de la secuencia de la próxima instrucción del programa que se ha de ejecutar. La longitud del IP depende de la cantidad de bits que se necesiten para direccionar cualquier instrucción en la memoria asignada al programa.

Considerese que para "X" una instrucción I_n podría, "supuestamente," alojarse en la palabra $FFFH$ que es la última, por lo tanto, el IP mide 12 bits como máximo, que permiten 2^{12} direcciones numeradas de 0 a $4095_{(10)}$. O sea que este puntero puede hacer referencia a las 4K direcciones de memoria, la última de ellas se representa con doce 1 binarios ($FFFH$).

Se indicó que I_0 siempre se carga en la palabra $000H$ (esta condición es válida para "X", pero no para todas las computadoras), de modo que éste es el valor inicial del IP cuando se arranca el funcionamiento de la computadora. Por otra parte, las instrucciones se almacenan en palabras sucesivas y ocupan en este caso una sola palabra; por esta razón, el IP debe poder incrementarse en una unidad para señalar siempre la próxima instrucción para buscar, luego de una ejecución. Más adelante veremos que el IP también puede aceptar un valor cualquiera impuesto por una instrucción que provoca una ruptura de secuencia en el programa. En la figura 7.8 se muestran las posibilidades de actualización y función de este registro.

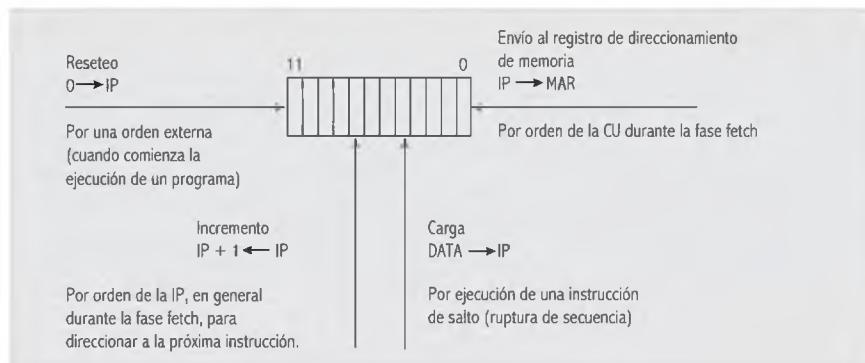


Fig. 7.8. Registro IP.

Una vez que la CU envía a memoria el contenido del IP, da la orden de lectura para que la palabra implicada, que llamaremos W , se almacene en el registro de palabra de memoria (MDR o *Memory Data Register*). Usualmente, la unidad de acceso a memoria es el bit; aquí la computadora "X" tiene una memoria organizada por palabras de 2 bytes y la unidad de direccionamiento es entonces la palabra (W).

Por último, la etapa *fetch* termina con la transferencia de la instrucción leída a un registro interno de la CU, donde la instrucción permanece almacenada mientras dure su ejecución. Este registro se denomina registro de instrucción (IR o *Instruction Register*) y su capacidad soporta el conjunto de bits del código de instrucción exemplificado, en este caso, 16 bits.

Como se observa en la figura 7.9, IR se relaciona directamente con el hardware que genera microoperaciones de ejecución y que denominaremos CU.

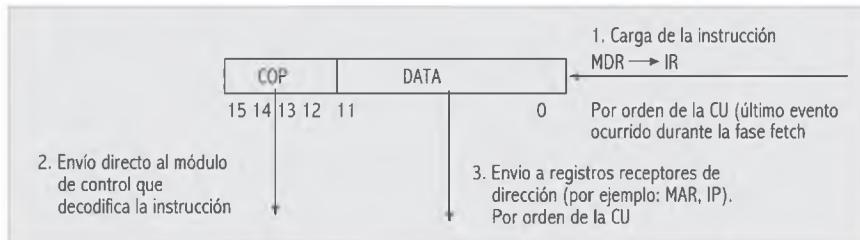


Fig. 7.9. Código de instrucción.

En este caso, la interpretación del COP la realiza un decodificador $4 \cdot 2^4 = 4 \cdot 16$, lo que significa que con 4 dígitos binarios se generan 16 instrucciones referenciadas por una de las salidas del circuito COP.

La zona denominada DATA es de 12 bits; se relaciona con el MAR cuando se debe tomar un dato de memoria y con el IP cuando se debe romper la secuencia normal del programa. Esta zona se ignora cuando la instrucción no afecta dato alguno. La CU actualiza el IP y entra en la fase *execute* de la instrucción buscada. En una *flag* llamada F (fase) se retiene un "1" durante la fase de búsqueda que cambia a 0 cuando comienza la fase de ejecución. Su función es importantísima para la CU de "X", porque le permite alternar de una fase a la otra, inhibiendo órdenes de búsqueda durante la fase de ejecución, y al revés.

Ahora que se conocen los dos registros de la CU de "X" implicados en la búsqueda de la instrucción, veremos quién y cómo se realiza su operación. En el caso de "X", el que genera la secuencia de microoperaciones de las fases *fetch* y *execute* es el circuito denominado control de instrucción. Aunque su nombre lo asocia a la segunda fase, gestiona la fase *fetch* comandado por el valor "1" de la *flag* F y por señales de tiempo que le permiten sincronizar las microoperaciones de búsqueda.

Por el momento no se entrará en detalle respecto de la estructura interna del control ni del sistema de reloj, pero se debe considerar que las microoperaciones obviamente no pueden enviarse simultáneamente. Por lo tanto, los componentes de hardware implicados en la fase *fetch* y la secuencia de sus eventos, que se muestran en la figura 7.10, son los siguientes:

1. $IP \longrightarrow MAR$
2. $Word \longrightarrow MDR$ (por orden de lectura)
3. $MDR \longrightarrow IR$
4. $IP + 1 \longrightarrow IP$
5. $0 \longrightarrow F$

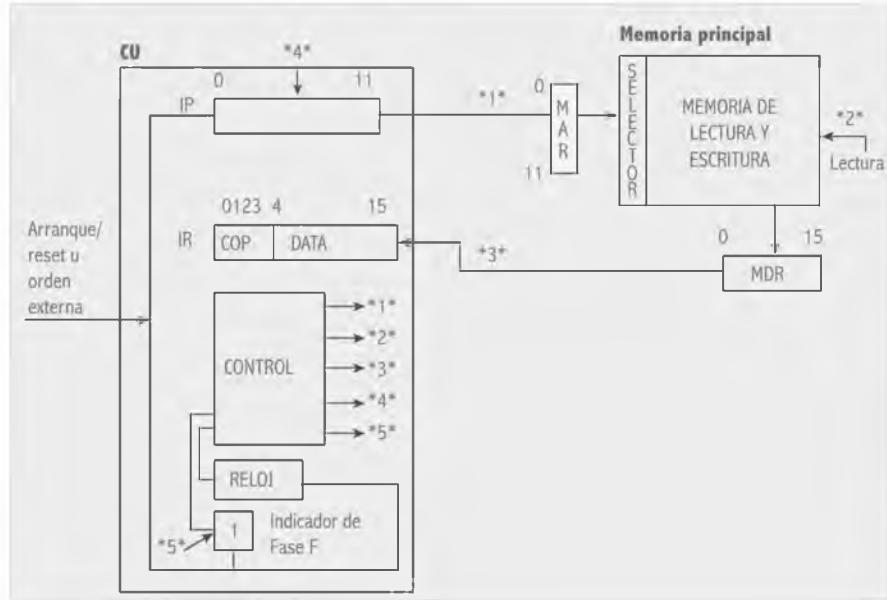


Fig. 7.10. Fase fetch.

F es una *flag* cuyo valor realmente como información de entrada al control, de modo de inhibir la búsqueda de una instrucción nueva a partir del evento 5 de la fase *fetch* y, de esta manera, delimitar esta fase con la de ejecución.

Las microoperaciones enumeradas antes definen la fase *fetch* y se producen cuando el biestable *F* está en 1 en los pulsos de reloj $t_p, t_{i+1}, \dots, t_{i+n}$.

La primera es una transferencia del contenido del puntero IP al MAR para seleccionar la palabra que se ha de leer.

La segunda es una orden de lectura a la memoria para hacerla efectiva.

La tercera es una transferencia de la instrucción al IR.

La cuarta corresponde a la actualización del IP para que señale la próxima instrucción para ejecutar.

La quinta corresponde a la actualización del indicador de fase *F* para entrar en fase *execute* (nótese que la cuarta microoperación puede producirse al mismo tiempo con la segunda o la tercera, porque no afecta los mismos registros; no así con la primera, porque se corre el riesgo de incrementar el IP antes de acceder a memoria). Si se quiere tener una idea de la estructura de hardware del control que gestiona este nivel de microoperaciones, la forma más simple es armar un circuito cuyas funciones de salida dependan de las variables de tiempo t_i y de la variable de fase *F*.

7.4.1.1. Diseño parcial del módulo del control asociado a la fase fetch

Las microoperaciones para completar esta fase se encuentran definidas en el control, como se verifica en la figura 7.11. El método que utilizaremos para el diseño final del control es el de asociar, mediante compuertas lógicas variables de tiempo, variables de estado de la CU y variables generadas por la instrucción para ejecutar.

Para este ejemplo se debe asumir que el tiempo de respuesta de memoria es igual a cuatro pulsos del reloj ($4 t_p$) y que ninguna ejecución de instrucción supera los ocho tiempos del ciclo de la computadora.

Como ya se mencionó, en la fase de búsqueda el conjunto de compuertas depende sólo de dos variables: la de tiempo y la de control de fase; esta última en su estado 1, por lo tanto:

$$\begin{aligned}
 f_1 &= F \cdot t_0 & IP &\longrightarrow MAR \\
 f_2 &= F \cdot t_1 & W &\longrightarrow MDR \\
 f_3 &= F \cdot t_5 & MDR &\longrightarrow IR \\
 f_4 &= F \cdot t_7 & IP + 1 &\rightarrow IP & f_4 \text{ se solapa con } f_2 \text{ ya que es} \\
 f_5 &= F \cdot t_7 & 0 &\longrightarrow F & \text{una microoperación independiente de la anterior}
 \end{aligned}$$

Nótese que el tiempo que transcurre entre la orden de lectura (f_2) y la transferencia del contenido de la palabra (f_3) coincide con el tiempo de respuesta de la memoria. En el tiempo t_7 se resetea la *flag* de estado de la CU para forzar la etapa de ejecución de la instrucción.

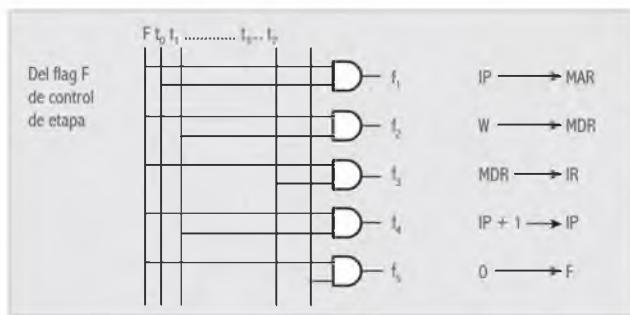


Fig. 7.11. Estructura de una parte del módulo de control para la fase *fetch*.

Ahora es oportuno que se analice cómo se comporta el control cuando decodifica las instrucciones del programa y genera las microoperaciones de ejecución.

Supóngase que el programa almacenado es muy elemental: sumar dos datos almacenados en memoria en las palabras *A0A* y *A0B* y calcular su promedio, almacenando el resultado en la palabra *A0C*.

Veamos cómo se resolvería si se propone como ejercicio práctico el mismo enunciado para realizar bajo las condiciones siguientes: *A* y *B* son binarios enteros signados que, representados en hexadecimal, asumen los valores *0004H* y *0002H*, respectivamente. La única herramienta de trabajo es una calculadora binaria, cuya capacidad máxima es de 16 bits, con un solo visor de 16 bits y un registro interno (que no se ve) también de 16 bits (en el caso de las calculadoras, el contenido del visor pasa al registro interno y se visualiza el segundo operando, mientras que en el caso de la computadora, el registro interno retiene el segundo operando). El teclado tiene dos teclas de entrada de dato (0 y 1), suma, desplazamiento a derecha (para dividir por la base sucesivamente) y cuatro luces que se prenden según el estado final de la operación.

En la figura 7.12 se observa un ejemplo de operación con una calculadora elemental.

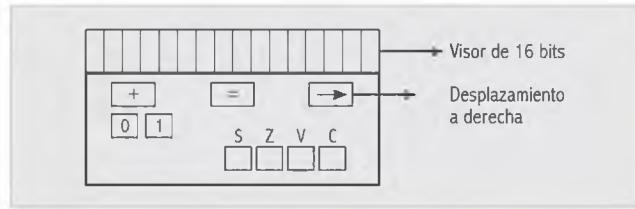
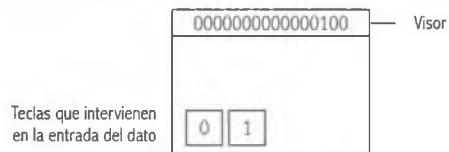
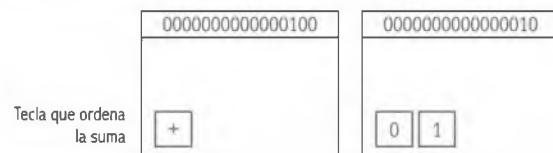


Fig. 7.12. Calculadora elemental.

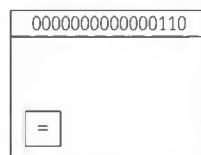
Su primera acción es ingresar el primer valor ($A = 0004$).



Su segunda acción es presionar la tecla "suma" e ingresar el segundo valor ($B = 0002$).



Al presionar la tecla "+" se obtiene:



El estado de las luces le indicará ciertas condiciones de la operación que podrá tomar en cuenta o no:

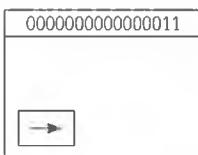
- La luz S se prende si el resultado de la operación fue negativo.
- La luz Z se prende si el resultado de la operación fue cero.
- La luz V se prende si el resultado excedió la capacidad del visor.
- La luz C se prende si el último acarreo, "el que me llevo", fue igual a 1.

Por lo tanto, para el juego de valores operados todas las luces permanecerán apagadas.

$$\begin{array}{r}
 \mathbf{C} \quad 00 \\
 0000\ 0000\ 0000\ 0100 \qquad \qquad \qquad V = 0 \\
 + \underline{0000\ 0000\ 0000\ 0010} \qquad \qquad \qquad Z = 0 \\
 \hline
 0000\ 0000\ 0000\ 0110 \qquad \qquad \qquad C = 0
 \end{array}$$

S Resultado ≠ 0 **S = 0**

Su última acción será presionar la tecla de desplazamiento a derecha y verá el resultado final



Esta calculadora elemental permite realizar operaciones muy simples. Esto se debe a su escasa cantidad de teclas "de control" (suma, igual y desplazamiento), por lo que se deduce que cuanto más deficiente sea la capacidad de "hardware", tanto más acciones serán necesarias para realizar lógicamente tareas complejas.

Esta calculadora pretende dar una noción del dispositivo de cálculo de nuestra computadora "X", su unidad aritmético-lógica. Llamemos acumulador a un registro de 16 bits que cumple la función del visor. Las teclas 0 y 1 no tienen sentido práctico, ya que los datos son traídos de memoria; las teclas +, -, → son órdenes que recibe la ALU por parte del control.

Las luces podrán, eventualmente, ser consultadas y permitirán que el programa lleve un control de qué ocurrió con la operación. Las llamaremos *flags* o banderas de estado; si el bit almacenado es 0, la condición no se satisface; si el bit almacenado es 1, sí. Ahora cada acción se transforma en una instrucción del programa P. A cada instrucción se le asigna un nombre simplificado y un código binario que se mostrará en hexadecimal.

	Nombre	Código (representa al código de máquina)
- Cargar el acumulador con la palabra A0A	LDA A0A <i>Load accumulator</i>	1A0A
- Sumar al acumulador la palabra A0B	ADA A0B <i>Add accumulator</i>	2A0B
- Desplazar el acumulador	SHR <i>Shift right</i>	AXXX
- Almacenar el acumulador en la palabra A0C	STA A0C <i>Store accumulator</i>	3A0C
- Fin del programa	HLT	0XXX

En la primera columna se detalla verbalmente cada instrucción; en la segunda se le asigna un mnemónico de tres letras, (se respetaron los mnemónicos más empleados en la descripción simbólica de instrucciones); la tercera columna muestra el código binario, que es, en definitiva, el código que entiende el control de "X", o código ejecutable

7.4.2 Fase execute

Una vez que se hace efectiva la búsqueda de alguna de las cuatro instrucciones de *P*, la CU entra en fase de ejecución (*O—F*). Cada código de instrucción tiene asignada una secuencia de microoperaciones definida en el control. Veamos qué sucede entonces en el estado de ejecución de cada una de ellas.

LDA AOA

La interpretación de su COP (0001) permite la transferencia del dato almacenado en la palabra *A0A* al acumulador de la ALU.

Las dos primeras microoperaciones hacen efectiva la lectura del dato de memoria y la tercera procede a la carga en sí.

Convengamos en representar los valores binarios en hexadecimal para mejorar la claridad de los ejemplos (fig. 7.13).

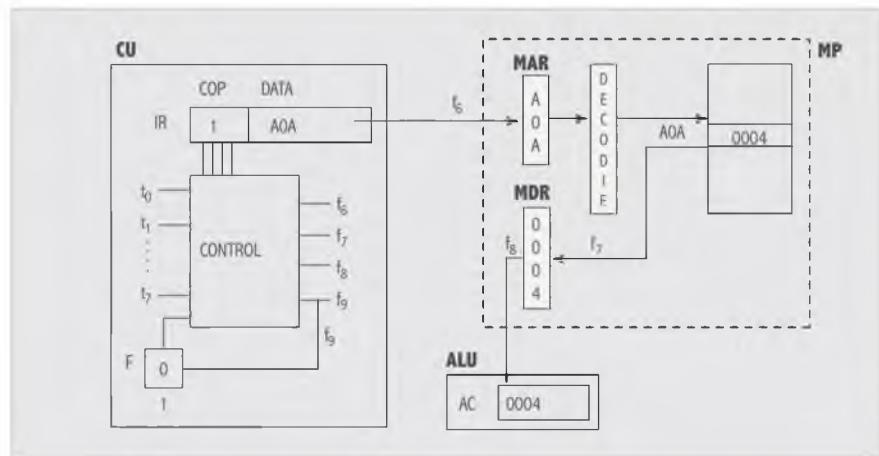


Fig. 7.13. Fase execute LDA.

$$f_6 = \bar{F} \cdot t_0 \cdot COP_1 \quad DATA \rightarrow MAR$$

COP₁ es la variable que representa al código de operación 0001

$$f_7 = \bar{F} \cdot t_1 \cdot COP_1 \quad WORD \rightarrow MDR$$

F es la variable que representa no fetch, esto es, execute

$$f_8 = \bar{F} \cdot t_5 \cdot COP_1 \quad MDR \rightarrow AC$$

F es la variable que representa no fetch, esto es, execute

$$f_9 = \bar{F} \cdot t_7 \cdot COP_1 \quad 1 \rightarrow F$$

f₆ Microoperación que habilita la transferencia de la dirección del dato indicada por la instrucción al registro de direccionamiento de memoria.

f₇ Microoperación que habilita la lectura de la unidad de memoria.

f₈ Microoperación que habilita la transferencia del dato leído al acumulador.

f₉ Microoperación que actualiza la *flag* controladora de fase y que provoca que el control se "entere" que debe producir microoperaciones de búsqueda para la instrucción siguiente.

ADA AOB

En el esquema para la ejecución de esta instrucción se incluye como dispositivo de cálculo un sumador binario paralelo, que se analizó con anterioridad en el capítulo Álgebra de Boole. Los bits *A_i* corresponden al dato almacenado en el acumulador (primer operando). Los bits *B_j* corresponden al dato leído de la palabra *A0B*, que permanece en el MDR (segundo operando).

Las salidas S están listas cuando la microoperación final habilita al acumulador como receptor de la suma, la misma microoperación habilita al *status register* para que se actualice.

Como en la instrucción anterior, las dos primeras microoperaciones permiten la lectura del dato almacenado en la palabra AOB (fig. 7.14).

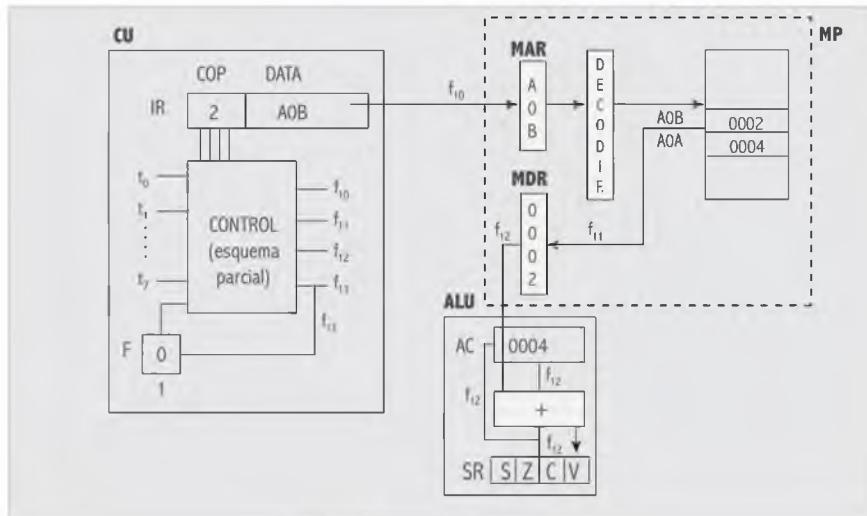


Fig. 7.14. Fase execute ADA.

$$f_{10} = \bar{F} \cdot t_0 \cdot COP_2 \quad DATA \longrightarrow MAR$$

$$f_{11} = \bar{F} \cdot t_1 \cdot COP_2 \quad WORD \longrightarrow MDR$$

$$f_{12} = \bar{F} \cdot t_5 \cdot COP_2 \quad MDR + AC \Rightarrow AC$$

$$f_{13} = \bar{F} \cdot t_7 \cdot COP_2 \quad 1 \longrightarrow F$$

f_{10} Similar a la microoperación f_6

f_{11} Similar a la microoperación f_7

f_{12} Microoperación que habilita al acumulador como receptor del resultado ($0006H$) y al SR, o *status register*, como receptor de la información del estado final de la operación (signo, cero, carry y overflow).

f_{13} Similar a la microoperación f_9

Nótese que en este ejemplo el MDR actúa como registro interno o segundo registro de la calculadora.

SHR

El control genera una microoperación que ordena al acumulador desplazar su información un bit a derecha (*shift right*). Incluir esta instrucción en este punto del programa implica lograr dividir el resultado almacenado en AC por dos, para calcular el promedio de los datos sumados.

Esta única microoperación afecta una entrada especial del registro acumulador que hace posible habilitar la función “desplazar”. Más adelante se verá que ésta es una de las varias funciones propias que tiene este registro (fig. 7.15).

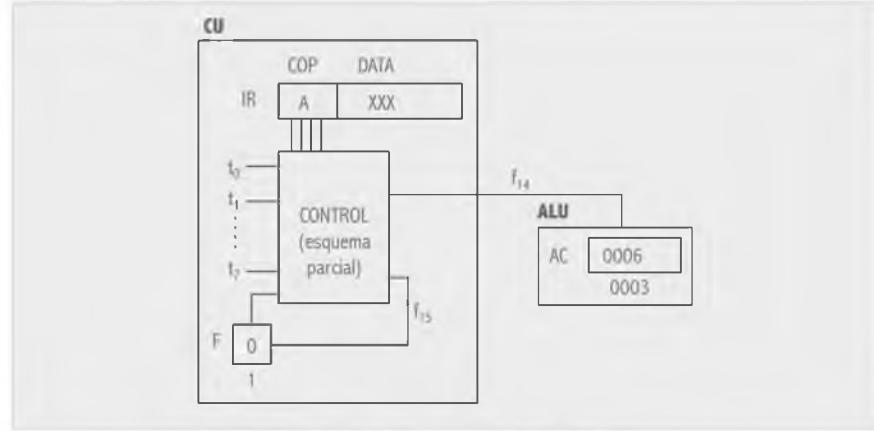


Fig. 7.15. Fase execute SHR.

$$f_{14} = \bar{F} \cdot t_0 \cdot COP_{16} \quad \xrightarrow{\text{AC}} \text{AC}$$

$$f_{15} = \bar{F} \cdot t_7 \cdot COP_{16} \quad 1 \xrightarrow{\text{ }} F$$

f_{14} Microoperación que permite desplazar la información del AC una posición a la derecha, con lo que divide por la base

f_{15} Similar a la microoperación f_9

STA AOC

Esta instrucción se incluyó en el programa para almacenar el resultado final en memoria.

El control genera microoperaciones que permiten la escritura del promedio calculado en la palabra de memoria AOC (fig. 7.16).

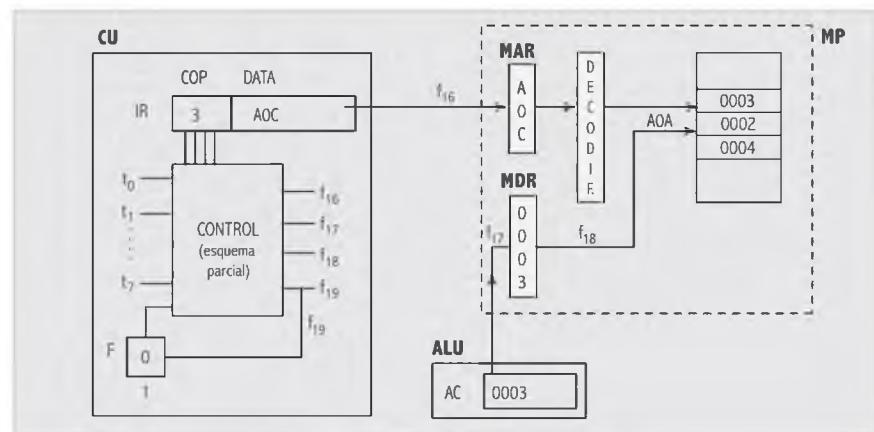


Fig. 7.16. Fase execute STA.

$$\begin{aligned}
 f_{16} &= \bar{F} \cdot t_6 \cdot COP_3 & DATA \rightarrow MAR \\
 f_{17} &= \bar{F} \cdot t_1 \cdot COP_3 & AC \rightarrow MDR \\
 f_{18} &= \bar{F} \cdot t_2 \cdot COP_3 & MDR \rightarrow WORD & \text{Tiempo necesario de escritura (el} \\
 f_{19} &= \bar{F} \cdot t_7 \cdot COP_3 & 1 \rightarrow F & \text{mismo asignado a la lectura)}
 \end{aligned}$$

f_{16} Microoperación que habilita la transferencia de la dirección de la palabra donde se va a escribir el dato, indicada por la instrucción, al registro de direccionamiento de memoria.

f_{17} Microoperación que habilita la transferencia del dato almacenado en el acumulador al registro de palabra de memoria.

f_{18} Microoperación que habilita la escritura en la unidad de memoria.

f_{19} Actualización de la *flag* F para entrar en una nueva fase de búsqueda.

(1— F).

HLT

La instrucción HLT (*halt* = parada) inhibe la generación de microoperaciones dejando a "X" suspendida. Esta instrucción es muy importante, puesto que de no existir se produce una etapa *fetch*, considerando la siguiente posición de memoria como una instrucción. Esto implica que el programa continúe de manera imprevista y provoque el fracaso de su ejecución. Para el modelo presentado, la ejecución de HLT genera una sola microoperación f_{20} que inactiva la generación de secuencias de tiempo t_i , con lo que se logra el efecto buscado. Todas las microoperaciones dependen de algún t_i , por lo tanto, si $t_i = 0$, todas las funciones de salida quedan en cero (fig. 7.17).

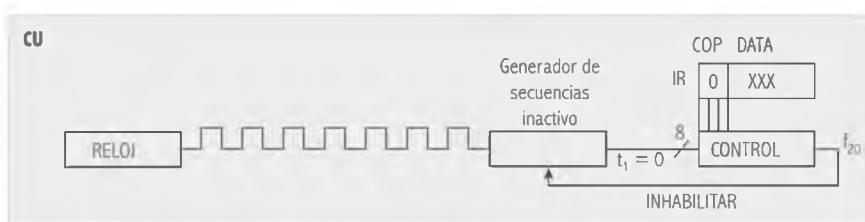


Fig. 7.17. Generador de unidades de tiempo t_i .

7.4.3 Flujo de datos entre los registros de la computadora básica

Por el momento, observe en las siguientes tablas el flujo de datos entre los registros de la computadora "X". Cada etapa de ejecución es precedida por la correspondiente etapa de búsqueda de la instrucción. En la tabla 7-1, que contiene el esquema de la evolución de ejecución de las instrucciones en forma detallada, y en la tabla 7-2, con un esquema simplificado por etapas, se indica cuál fue el efecto sobre los registros implicados.

Aclaraciones:

X: indica estado desconocido.

F: contenido de la *flag* indicadora de fase.

PC: contenido del *Program Counter* (PC) o contador de programa

MAR (o *Memory Address Register*): contenido del registro de direccionamiento de memoria.

MDR: contenido del registro de palabra de memoria

AC (o *Accumulator*): contenido del acumulador.

SZVC (SR): estado de las *flags* o banderas contenidas en el *status register*.

A0A: contenido de la palabra de memoria A0A.

A0B: contenido de la palabra de memoria A0B.

A0C: contenido de la palabra de memoria A0C.

000: contenido de la palabra de memoria 000.

001: contenido de la palabra de memoria 001.

002: contenido de la palabra de memoria 002.

003: contenido de la palabra de memoria 003.

W: WORD indica, en forma genérica, una palabra de memoria cualquiera.

Tabla 7-1. Esquema de la evolución de ejecución de las instrucciones.

Estado inicial	F	PC	IR	MAR	MDR	AC	SZVC	A0A	A0B	A0C	000	001	002	003	004	H
	1	000			000			0004	002		1A0A	2A0B	AXXX	3A0C	0XXX	0
IP → MAR W → MDR MDR → IR IP+1 → IP 0 → F	0	001	1A0A		1ADA											
DATA → MAR W → MDR MDR → AC 1 → F	1			A0A		0004	0004									
IP → MAR W → MDR MDR → IR IP+1 → IP 0 → F	1	002	2A0B		001	2A0B										
DATA → MAR W → MDR MDR+AC → AC 1 → F	1			A0B		0002		0006	0000							
IP → MAR W → MDR MDR → IR IP+1 → IP 0 → F	0	003	AXXX		002	AXXX										
AC → AC 1 → F	1						003									
IP → MAR W → IR MDR → IR IP+1 → IP 0 → F	0	004	3A0C		003	3A0C										
DATA → MAR AC → MDR MDR → W 1 → F	1			A0C		0003					003					
IP → MAR W → MDR MDR → IR IP+1 → IP 0 → F	1	005	0XXX		004	0XXX										

Tabla 7-2. Esquema simplificado por etapas.

	<i>IP</i>	<i>IR</i>	<i>MAR</i>	<i>MDR</i>	<i>AC</i>	<i>SR</i>	<i>F</i>
Estado inicial	000	XXXX	XXX	XXXX	XXXX	x	1
Fetch	001	1A0A	000	1A0A			0
Execute LDA			A0A	0004	004		1
Fetch	002	2A0B	001	2A0B			0
Execute ADA			A0B	0002	006	0	1
Fetch	003	AXXX	002	AXXX			
Execute SHR			No afecta a mem.		003	0	1
Fetch	004	3A0C	003	3A0C			0
Execute STA			A0C	0003			0
Fetch	005	Instrucc de fin	004	Instrucc de fin		X	

7.4.4 Juego completo de instrucciones de “X”

Para cubrir las necesidades de programación de “X” a continuación se presentan otras instrucciones y los enunciados de algunas tareas de aplicación de ellas.

“X” tiene tres tipos de instrucciones definidos con clividad:

1. Las instrucciones que afectan datos almacenados en la memoria. Por ejemplo, LDA *A0A*.
2. Las instrucciones que afectan datos almacenados en registros que no pertenecen a la memoria (*AC*, *STATUS REGISTER*, etc.). Por ejemplo, SHR.
3. Las instrucciones que permiten la entrada de un dato a “X” desde el exterior, o la salida de un dato desde “X” al exterior. Por ejemplo, INP.

Para la representación de instrucciones se adopta un lenguaje que permite que el usuario exprese sus programas, sin necesidad de recordar el código binario asignado a cada una de las instrucciones. Este lenguaje asume, por convención, un mnemónico de tres letras para definir el COP y tres dígitos hexadecimales para definir la dirección del operando en memoria en aquellas instrucciones de referencia a memoria. En la tabla 7-3 se muestra la expresión simbólica, la representación binaria en hexadecimal, el código binario ejecutable y las microoperaciones de ejecución.

Algunas de las instrucciones de la tabla 7-3 fueron analizadas para la ejecución del programa P, las demás, tal vez requieran una explicación adicional.

JMP HHH	Es una instrucción de salto incondicional, o sea que usted debe usarla cuando necesite romper el orden secuencial del programa. HHH es la dirección de la palabra de la instrucción a la que quiere saltar. Por eso el control transfiere directamente esta dirección al PC, de modo que la próxima etapa de búsqueda lo encuentre actualizado.
ANA HHH	Es una instrucción que permite asociar al acumulador y al dato obtenido mediante el operador lógico AND, de modo que el contenido final del acumulador sea el resultado de la operación “producto lógico” para cada par de valores bit a bit.
XOA HHH	Es una instrucción que permite asociar al acumulador y al dato obtenido mediante el operador lógico OR EXCLUSIVE, de modo que el contenido final del acumulador sea el resultado de la operación “suma exclusiva” para cada par de valores bit a bit.

Tabla 7-3. Microoperaciones de ejecución.

	<i>Símbolo</i>	<i>Taquigráfico decimal</i>	<i>Ejecutable binario</i>	<i>Microoperaciones de ejecución</i>	
<i>Referencia a memoria</i>	LDA HHH	1HHH	0001bbbbbbbbbbbb	DATA W MDR	MAR MDR AC
	ADA HHH	2HHH	0010bbbbbbbbbbbb	DATA W MDR	MAR MDR AC
	STA HHH	3HHH	0011bbbbbbbbbbbb	DATA AC MDR	MAR MDR MW
	JMP HHH <i>(jump)</i>	4HHH	0100bbbbbbbbbbbb	DATA	IP
	ANA HHH <i>(and accumulator)</i>	5HHH	0101bbbbbbbbbbbb	DATA W MDR [^] AC	MAR MDR AC
	XOA HHH <i>(or exclusive accumulator)</i>	6HHH	0110bbbbbbbbbbbb	DATA W MDR + AC	MAR MDR AC
<i>Referencia a registro</i>	HLT	0XXX	0000XXXXXXXXXXXX	1 → H	
	CLA	7XXX	0111XXXXXXXXXXXX	0 → AC	
	CMA	8XXX	1000XXXXXXXXXXXX	AC → AC	
	INC	9XXX	1001XXXXXXXXXXXX	AC + 1 → AC	
	SHR	AXXX	1010XXXXXXXXXXXX	AC → AC	
	SNA	BHHH	1011bbbbbbbbbbbb	SI AC < 0	
	SZA	CHHH	1100bbbbbbbbbbbb	SI AC = 0	→ DATA → IP
	SCA	DHHH	1101bbbbbbbbbbbb	SI C ₁₅ = 1	
<i>Entrada y salida</i>	INP	EXXX	1110XXXXXXXXXXXX	ENT → AC, AC → SAL,	0 → FEN 0 → FSA
	OUT	FXXX	1111XXXXXXXXXXXX		

HLT

Es la instrucción que determina el fin del programa (*halt*). Es imprescindible para que no se sigan buscando posiciones de memoria que no almacenan instrucciones, evitando así resultados imprevistos. "X" lo logra al inhibir la generación de señales de tiempo.

Se ha considerado apropiado asignarle el valor 0, de modo que si por error se ejecuta una instrucción almacenada en una palabra de memoria puesta a 0, el control genera la misma microoperación que para ejecutar HALT. Ésta y todas las instrucciones de referencia a registro o de E/S ignoran el campo del código de instrucción DATA.

CLA

Permite la puesta a 0 del acumulador (*clear accumulator*).

CMA

Permite la inversión de los bits del acumulador.

Calcula el complemento restringido del dato almacenado en el acumulador (*complement accumulator*).

INC

Permite sumar una unidad al dato cargado en el acumulador (*increment accumulator*).

SHR

Permite desplazar un bit del dato almacenado en el acumulador.

En el caso de SHR (*shift right*) el desplazamiento se produce a derecha.

SNA HHH	Es una instrucción de salto condicionado, que permite tomar una decisión en el programa, en función de una condición preestablecida. En este caso, la condición es que el dato almacenado en el acumulador sea negativo, esto es, que para efectuar el salto la instrucción considera el valor 1 de la flag "S" en el registro de estado. Si $S = 1$, entonces, modifica el contenido actual del PC con la dirección de salto especificada en el campo DATA, lo que provoca que la próxima fase <i>fetch</i> no busque la instrucción siguiente al SNA, sino la indicada en esta instrucción. Si $S = 0$, significa que el dato almacenado en el acumulador no es negativo, por lo tanto, no se satisface la condición y no se produce ninguna microoperación de ejecución, continuando la secuencia normal del programa.
SZA HHH	Es una instrucción de salto condicional igual que las anteriores. En este caso, la condición que se ha de verificar es que el dato almacenado en el acumulador sea 0 , por lo tanto, la bandera consultada es la Z . Con $Z = 1$ la condición se satisface.
SCA HHH	Es una instrucción de salto condicional igual que las anteriores. En este caso la condición que se ha de verificar es que el último acarreo sea 1 , para considerar que ésta se encuentra satisfecha.
INP	Esta instrucción permite la entrada de un dato (<i>input</i>) desde un periférico de entrada. Como se verá más adelante, la transferencia del dato se realiza entre un registro propio del periférico ENT y el acumulador. "X" está dotada de un solo periférico de entrada, por lo tanto, el campo DATA del código de instrucción no identifica direcciones de periféricos, entonces se lo ignora. La siguiente instrucción al <i>input</i> no se ejecutará hasta tanto no se haya hecho efectiva la entrada del dato. El control "consulta" la bandera de estado FEN. Si $FEN = 0$ no habilita la transferencia, porque significa que el periférico no envió información al registro ENT. Cuando $FEN = 1$, recién habilita la transferencia y vuelve la bandera FEN a 0 . O sea que FEN se pone en 1 cuando hay información para transferir; en este caso, el que pone FEN en 1 es el periférico. El dato ingresado queda en el acumulador, por lo general, el programador lo transfiere a la memoria con una instrucción STA HHH.
OUT	Esta instrucción permite la salida de un dato almacenado con anterioridad en el acumulador hacia un dispositivo periférico. Por lo tanto, la transferencia se realiza entre el acumulador y un registro propio del periférico SAL. "X" sólo tiene asignado un dispositivo de salida, por lo tanto, el campo DATA es ignorado. La instrucción siguiente no se ejecutará hasta que no se haya hecho efectiva la salida. El control "consulta" la bandera FSA. Si $FSA = 0$ no habilita la transferencia, porque significa que el periférico aún no está listo. El periférico pone la bandera FSA en 1 cuando está listo para la recepción. La bandera vuelve a 0 cuando se carga la información en el registro SAL para impedir que se siga enviando información antes de que el periférico la muestre.

7.4.5 Unidad de control y sincronización del tiempo

Ahora la pregunta puede ser, ¿cómo se controla el flujo secuencial de microoperaciones para llevar a cabo la tarea del procesamiento?

Piense en la ejecución de la instrucción LDA XXX ¿Cómo se sincronizan las microoperaciones para que la orden de lectura ($M \rightarrow MDR$) no se produzca antes de haber seleccionado la

palabra (DATA → MAR) o para que la transferencia al acumulador (MDR → AC) no se produzca antes de que la palabra haya sido cargada en el MDR? Es obvio que cada microoperación debe ser controlada por una variable fundamental: el tiempo. Tomando el último ejemplo, la palabra no estará disponible en el MDR hasta que se haya cumplido el tiempo estimado de respuesta de memoria; antes sería inútil intentar transferir su contenido al acumulador.

Las señales de tiempo que afectan a las microoperaciones están reguladas según el tiempo de respuesta de los registros que involucran.

En la mayoría de las computadoras las señales de tiempo son generadas por un sistema de reloj. Éste se encuentra constituido por un oscilador y circuitos asociados que generan pulsos, cuyo ancho y separación son determinados en forma precisa. Se denomina **ciclo de reloj** o **ciclo menor** al intervalo entre dos pulsos consecutivos de reloj (0 y 1), como lo muestra la figura 7.18.



Ciclo de reloj: es el tiempo que transcurre entre dos pulsos adyacentes.



Fig. 7.18. Ciclo de reloj.

En el diagrama de la figura 7.18 se ve la oscilación de la señal entre los valores 0 y 1, la frecuencia del reloj se controla por un oscilador y cada ciclo de reloj se identifica como 1 hercio, Hz (hertz o ciclo). O sea: $1\text{Hz} = 1\text{ ciclo / 1 segundo}$.

$$\text{frecuencia} = \frac{\text{cantidad de ciclos de reloj}}{\text{seg}}$$

Dado que las unidades de tiempo de una CPU están en el orden de los nanosegundos (10^{-9} seg, ns), si se quiere averiguar cuántos ns "tarda" un ciclo de reloj para una computadora que trabaja con una frecuencia de 25 millones de Hz o MHz:

$$\begin{aligned} 25\text{ MHz} &= \frac{25.000.000}{\text{seg}} \\ \text{luego} \quad 1\text{ ciclo} &= \frac{1\text{ seg}}{25.000.000} = \frac{1\text{ seg}}{25 \cdot 10^6} = 0,04\text{ seg} \cdot 10^{-6} = \\ &= 40 \cdot 10^{-6} \cdot 10^{-3}\text{ seg} = \\ &= 40 \cdot 10^{-9}\text{ seg} = 40\text{ nanoseg} \end{aligned}$$

Una computadora controlada por un sistema de reloj se denomina **sincrónica**.

En estas computadoras se proporcionan secuencias de tiempo repetitivas. Una secuencia repetitiva puede estar formada por 4, 8 o 16 señales de tiempo, que se diferenciarán por el valor de su subíndice (t_0, t_1, \dots, t_{15}). El tiempo de una secuencia repetitiva constituye el **ciclo de la computadora** o **ciclo de máquina**. Éste debería ser compatible con el ciclo de memoria. Recordemos que **ciclo de memoria** es igual al tiempo de acceso a la memoria, si se trata de una memoria de lectura no destructiva, y será igual al tiempo de acceso más el tiempo de restauración, en el caso de una memoria de lectura destructiva.

El **tiempo de acceso a memoria** es el tiempo que tarda la CU en buscar la información en la memoria y dejarla disponible en el MDR.

0 1 2 3 4 5 6 7

 /-/ "pulso de reloj"
 /---/ "ciclo menor" o "ciclo de reloj"
 /-----/ "ciclo de la computadora" o "ciclo mayor" o
 "ciclo de máquina"

"X" es una computadora sincrónica con un sistema de reloj muy elemental, como se observa en la figura 7.19, que genera ocho señales de tiempo: $t_0, t_1, t_2, t_3, t_4, t_5, t_6, t_7$. El reloj maestro está asociado a un registro de desplazamiento circular, cuya información inicial es 10000000. Por cada pulso del maestro, el único bit "uno" se desplaza un lugar a la derecha. Cada biestable del registro genera una señal de tiempo cada ocho pulsos del reloj maestro.

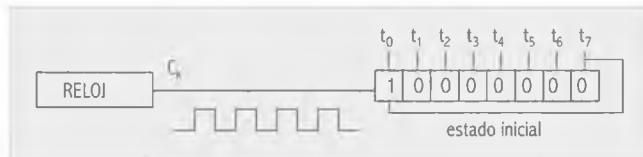


Fig. 7.19. Generación de señales de tiempo.

El registro se carga con el estado inicial mediante un comando externo controlado por el usuario, que indica el comienzo del procesamiento, llamémoslo *ON*, y se resetea mediante otro comando externo, también controlado por el usuario, que indica fin del procesamiento, llamémoslo *OFF*. Puede ser causa de reseteo de este registro un evento interno del procesamiento en curso, por ejemplo, la interpretación de una instrucción de fin de programa. Es obvio que el reseteo de este registro provoca que no se generen señales de tiempo y, por lo tanto, la interrupción de la actividad en "X".

Ahora que está resuelto el problema de sincronización de operaciones de "X", se puede completar el esquema de su unidad de control y evaluar las condiciones que permiten que el control "interprete" los códigos de instrucción y genere las microoperaciones.

Repasemos los componentes de "X": memoria de 4096 palabras de 16 bits cada una; sumador binario asociado al acumulador de 16 bits y al MDR de 16 bits; tres banderas de estado de operación en el *status register* (*S*, *Z* y *C*); puntero de instrucción de 12 bits, llamado contador de programa, que indica la próxima instrucción que ha de ejecutarse.

Convengamos que la primera instrucción del programa siempre se carga en la palabra 000H, por lo tanto, es éste el valor inicial de PC. El registro de instrucción de 16 bits almacena la instrucción que está siendo decodificada y ejecutada por el control y se encuentra dividido en dos partes: cuatro bits para el código de operación (COP) y 12 bits para la dirección del operando (DATA).

El usuario presiona la tecla *ON* para iniciar la ejecución del programa, lo que desencadena una serie de operaciones internas para la búsqueda de la primera instrucción de memoria. A partir de ese momento, la unidad de control estará en uno de dos estados: búsqueda de la instrucción (o fase *fetch*) o ejecución de la instrucción (o fase *execute*). La unidad de control actualiza la *flag* de control de fase para pasar de un estado a otro. Acordemos que el valor de este biestable es 1 para la fase de búsqueda y 0 para la fase de ejecución. Todos estos componentes brindan cierta información al control que genera, considerando estas variables, las "funciones de control" necesarias para llevar a cabo el procesamiento: las microoperaciones.

El esquema de la figura 7.20 muestra en forma general la CU de "X" formada por dos bloques: el decodificador de instrucción y el secuenciador que genera microoperaciones desfasadas en el tiempo. El primero puede obtenerse a partir de un decodificador 4 · 16. Las cuatro entradas al circuito varian entre 0000 y 1111, según la instrucción almacenada en el IR. Las salidas indican con un 1 cuál de las diecisésis combinaciones se dio en la entrada.

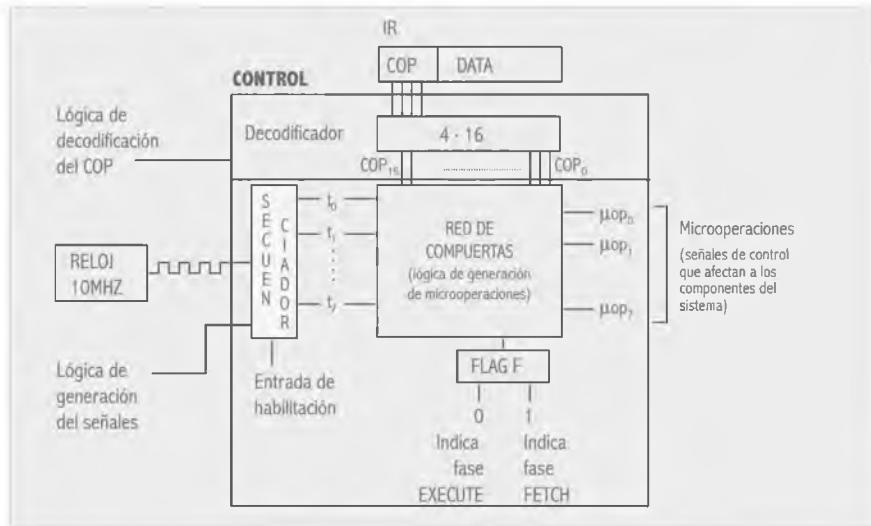


Fig. 7.20. Esquema interno de CU.

7.4.5.1 Diseño parcial de la CU para la orden de lectura de memoria

La variable COP_i está asociada en la lógica de ejecución a un juego determinado de microoperaciones.

En la fase de ejecución las compuertas dependen de la variable de control de fase negada, \bar{f}_i , de las variables que indican cuál de los 16 COP se almacena en el IR y de las variables de tiempo para secuenciar las microoperaciones.

Las instrucciones que requieren un operando de memoria son:

```

LDA XXX COP0
ADA XXX COP2
STA XXX COP3
ANA XXX COP5
XOA XXX COP6

```

Las microoperaciones que permiten la carga del operando en el MDR para cualquiera de ellas son:

```

DATA → MAR
WORD → MDR

```

En consecuencia, deben producirse en los primeros dos tiempos del ciclo de la computadora. Las dos compuertas que generan en su salida una de estas microoperaciones dependen, entonces, de las variables siguientes:

$$\begin{aligned}\muop_0 &= DATA \rightarrow MAR = \bar{f} \cdot t_0 \cdot (COP_1 + COP_2 + COP_3 + COP_4 + COP_5) \\ \muop_1 &= WORD \rightarrow MDR = \bar{f} \cdot t_1 \cdot (COP_1 + COP_2 + COP_4 + COP_5) \\ \muop_2 &= \boxed{\quad} = \bar{f} \cdot t_5 \cdot COP_1 \\ &\text{ejecución de la instrucción} \\ \muop_3 &= 1 \rightarrow F = \bar{f} \cdot t_7\end{aligned}$$

En consecuencia, para la instrucción LDA XXX la

$$\muop_2 = MDR \rightarrow AC = \bar{f} \cdot t_5 \cdot COP_1 \quad \text{microoperación que habilita la transferencia entre los registros MDR y AC.}$$

Es importante destacar que todas las instrucciones se ejecutan en un ciclo de computadora, aun cuando no utilicen todas las variables de tiempo para su ejecución.

7.4.5.2 Diseño parcial del módulo del control asociado a la fase execute de algunas instrucciones

Para finalizar, en la figura 7.21 se observa un diseño parcial del control considerando solamente la función de control WORD—MDR (orden de lectura de memoria). Si se analizan los casos de lectura en los ejemplos desarrollados, se notará que la CU sólo dispara una orden de lectura en la fase de búsqueda de la instrucción (lectura de la palabra que contiene la instrucción) y en la fase de ejecución de aquellas instrucciones que necesiten un operando de memoria (lectura de la palabra que contiene el operando).

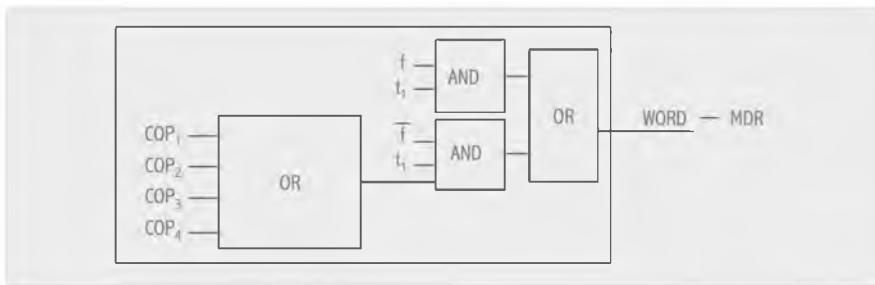


Fig. 7.21. Diseño parcial del control para la orden de lectura de la memoria.

Si la computadora "X" tuviese sólo las cinco instrucciones del programa P , podríamos armar la red de compuertas del control para generar las distintas microoperaciones. Es importante analizar que la *flag* indicadora de fase cambia a 0 en el último paso de ejecución de cualquier instrucción, o sea que el estado de F es independiente del valor del COP_i y puede producirse en el último t_i del ciclo de la computadora, que puede ser t_7 , esto es, $SET F = t_7$. El registro MDR recibe la transferencia del PC sólo en la fase de búsqueda, o sea:

$$SETF = f_1 = F \cdot t_7$$

recibe la transferencia del campo DATA en f_8 , f_{10} y f_{16} que se puede expresar como:

$$\begin{aligned}DATAMAR &= f_6 + f_{10} + f_{16} = \bar{F} \cdot t_0 \cdot COP_1 + \bar{F} \cdot t_0 \cdot COP_2 + \bar{F} \cdot t_0 \cdot COP_3 = \\ &= \bar{F} \cdot t_0 \cdot (COP_1 + COP_2 + COP_3)\end{aligned}$$

El registro MDR recibe la transferencia de la palabra de memoria en f_7 y f_{11} , que se puede expresar como:

$$\begin{aligned} WORDMDR &= f_x + f_y + f_{11} = F \cdot t_1 + \bar{F} \cdot t_1 \cdot COP_1 + \bar{F} \cdot t_1 \cdot COP_2 = \\ &= F \cdot t_1 + \bar{F} \cdot t_1 (COP_1 + COP_2) \end{aligned}$$

y recibe la transferencia del acumulador en f_{12} , entonces:

$$ACMDR = \bar{F} \cdot t_1 \cdot COP_3$$

El acumulador recibe una orden de desplazamiento en f_{14} , entonces:

$$SHRAC = \bar{F} \cdot t_0 \cdot COP_{10}$$

y una orden de recibir el resultado de la suma en f_{12} :

$$SUMAC = \bar{F} \cdot t_5 \cdot COP_2$$

El secuenciador de tiempos puede detenerse en el primer tiempo de la fase *execute*, entonces,

$$DISABLE = \bar{F} \cdot t_0$$

7.4.6 El módulo de cálculo: unidad aritmético-lógica

El módulo de tratamiento de datos es la **unidad aritmético-lógica** (ALU). En este módulo los datos se tratan según órdenes de la **unidad de control** (CU), que interpreta la instrucción durante su estado de ejecución.

Las instrucciones de "X" que hacen referencia a la ALU son ADA, INC, CMA, SHR y SHL, correspondientes a los operadores aritméticos "suma" (+), "incremento" (+1), "complemento", "división por la base binaria" y "multiplicación por la base binaria". ANA y XOA responden a los operadores lógicos "y" (AND) y "o excluyente" (or exclusive).

Las instrucciones LDA, STA y CLA permiten el control del registro acumulador perteneciente a este módulo, gestionando su carga, descarga y puesta a 0.

Como se ve, la mayor parte de las instrucciones de "X" requiere la intervención de la ALU.

La unidad aritmético-lógica de "X" está implementada en torno del registro acumulador. Cuando la operación se realiza sobre un dato, éste se almacena en el acumulador y el resultado de la operación permanece en este registro. Las instrucciones que generan operaciones sobre un solo dato en la ALU son:

$$CLA \quad CMA \quad INC \quad SHR \quad SHL$$

Cuando aparece el código de operación correspondiente a alguna de ellas, el control genera una señal que habilita alguna de las funciones del acumulador, por lo tanto, este registro debe tener capacidad de "puesta a 0", "complemento", "incremento", "desplazamiento a derecha" y "desplazamiento a izquierda".

El resto de las operaciones aritméticas o lógicas afectan a dos datos. En consecuencia, el primer dato se almacena en el acumulador y el segundo permanece en el separador de memoria (MDR), mientras que el resultado final se almacena en el acumulador. Las instrucciones de este tipo son

$$ADA, ANA, XOA$$

que requieren una lógica adicional en la ALU. Considérese un sumador binario paralelo armado, en este caso, con 16 sumadores completos convencionales. Veamos la segunda opción, de modo que las tres funciones queden implementadas, según la figura 7.22.

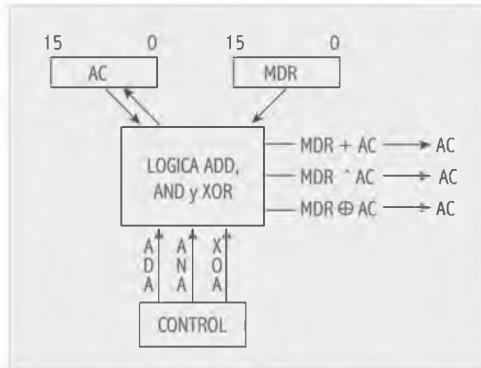


Fig. 7.22. Diagrama de relaciones entre los operandos y las operaciones.

Es importante destacar el valor de las banderas (*flags*) del registro de estado asociado a esta unidad (*status register* o SR). El estado de las banderas permite que el programador controle la condición final de una operación para establecer o no una ruptura de la secuencia normal del programa. Cada bandera se actualiza después de una operación en la ALU (asumiendo un valor 0 o 1), según la lógica presentada en la figura 7.23.

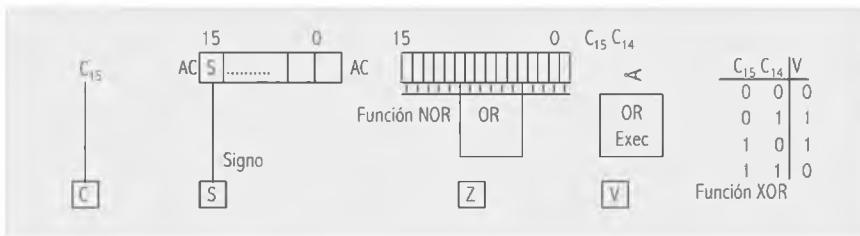


Fig. 7.23. Lógica de actualización de las banderas.

El *status register* de "X" representa sólo tres *flags*, que son:

- N –a veces denominado S por signo– (resultado del acumulador negativo) que vale 1 cuando $S(AC_{15}) = 1$.
- Z (resultado = cero) que vale 1 cuando todos los AC_i son iguales a 0.
- C (*carry* o acarreo que se produce al sumar el último par de bits del acumulador) que asume el valor de C_{15} .

El *overflow* se produce cuando el resultado excede la capacidad del acumulador y, por lo tanto, es incorrecto; cuando se produce *overflow*, "X" detiene la ejecución del programa; como no hay instrucción de consulta de *overflow*, se puede detectar consultando si $C \neq N$.

Las instrucciones de salto condicional SNA, SZA y SCA provocan salto cuando la condición de la bandera es verdadera. Por lo tanto, la lógica que genera la microoperación en el control depende de ellas y se representa en la figura 7.24.

En la figura 7.25 se presenta el esquema global de la ALU de "X".

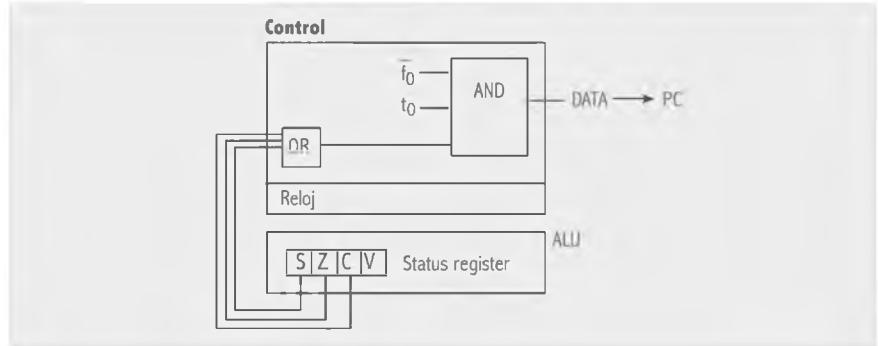


Fig. 7.24. Lógica de actualización de las banderas.

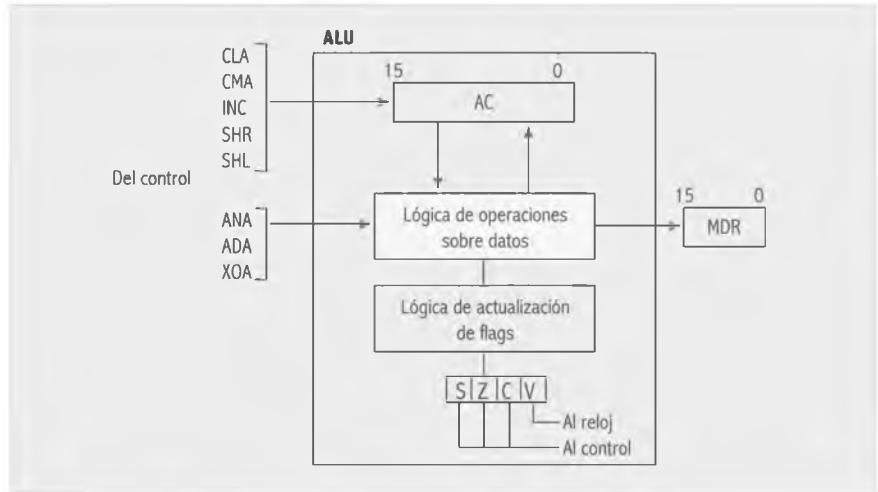


Fig. 7.25. Instrucciones aritméticas y lógicas en la ALU.

El ejemplo siguiente le permitirá ver una aplicación del efecto que causa el estado final de una operación para la toma de decisiones en un programa. Considere una porción de programa Assembler, donde el resultado de comparar dos números por mayor, menor e igual permite que el programador se bifurque a otros puntos del programa. El método usado para la comparación es la resta de los datos.

010 LDA 3A2	
011 CMA	resta (3A1 - 3A2)
012 INC	
013 ADA 3A1	
014 SZA 100	si son iguales salta a 100
015 SNA 200	si 3A1 < 3A2 salta a 200 (si Z = 0 y S = 1)
016 JMP 300	salta a 300

7.4.6.1 El desplazamiento de bits en los registros de la ALU

Cada desplazamiento a derecha o a izquierda de los bits de un registro implica, en principio, el corrimiento de un solo bit, pero trae aparejado el arrastre de todos los demás. Los desplazamientos a derecha o izquierda son otras dos operaciones accesibles gracias a la ALU, donde los operandos residen temporalmente en registros asociados al sumador.

En este momento, cabe preguntarse por qué razón se desplazan bits en un registro. Básicamente, por tres razones esenciales:

- Para multiplicar.
- Para dividir.
- Para “testear” un bit.

En efecto, usted sabe que hemos remarcado que en las operaciones aritméticas definidas para la ALU la multiplicación y la división no figuran. Una multiplicación se define por medio de una serie de instrucciones en código de máquina, que implican sumas y desplazamientos sucesivos. Por ejemplo, supóngase en un formato de sólo tres dígitos decimales el valor siguiente:

$$\begin{array}{r} 080 \\ \rightarrow 008 \\ \hline 080 \\ \leftarrow 800 \end{array}$$

Un desplazamiento a derecha divide el valor 080 por 10 (008), mientras que uno a izquierda lo multiplica por 10 (800). En sistemas de numeración de cualquier base, el desplazamiento permite multiplicar o dividir un número por su base (siempre representada por 10). De la misma manera, en binario, para multiplicar un número 2 (la base) es necesario agregarle un 0 a derecha, que es equivalente a desplazar todos sus bits un bit a izquierda; para multiplicarlo por 4, se necesitan dos desplazamientos. Veamos el ejemplo siguiente, que se trata de multiplicar el número 6 por 5:

$$6 \cdot 5 = ((6 \cdot 2) \cdot 2) + 6 = 30$$

Así, en formato de 8 bits, los registros contendrán los valores que se detallan en la figura 7.26,

S	0 0000110	+6
desplazamiento	0 0001100	+12
desplazamiento	0 0011000	+24
sumo +6	0 0011110	+30

Fig. 7.26. Evolución del cálculo.

lo que nos da un algoritmo válido para la multiplicación, utilizando desplazamientos sucesivos y sumas.

Es posible que en el desplazamiento también intervenga el bit de acarreo. Se pueden hacer tantos desplazamientos como los necesarios para que un bit que se quiera testear llegue al acarreo; de este modo, se pregunta el valor del bit con la instrucción en código de máquina SCA.

7.4.6.2 Comparación mediante resta

La comparación también es función de la ALU. Cuando se quieren comparar dos operandos, se efectúa una sustracción sin necesidad de almacenar el resultado. Las instrucciones en código de máquina permiten preguntar por la bandera indicadora de zero o de signo, de modo

que, si $Z = 0$ y $S = 1$, el primer operando es menor que el segundo, y si $Z = 0$ y $S = 0$, el primer operando será mayor que el segundo.

7.4.6.3 Incremento y decremento de un registro

Incrementar el valor de un operando implica aumentarlo en una unidad o, por el contrario, decrementar significa restarle una unidad. La operación se realiza sobre un solo operando y, por lo general, afecta al registro acumulador.

7.4.6.4 Operaciones lógicas

Examinemos las cuatro operaciones lógicas que permite realizar la ALU.

La complementación: es una operación que se aplica a un solo operando; así, si el operando mide un octeto, la operación invertirá el valor de cada uno de los 8 bits:

$$\begin{array}{r} 10100101 \\ \quad \quad \quad | \\ 01011010 \end{array}$$

Una de las aplicaciones tiene lugar en el cálculo de una resta, $X - Y = Z$, mediante la suma del complemento del sustraendo. Ejemplo:

<i>LDA, Y</i>	
<i>CMA</i>	<i>complemento</i>
<i>INC</i>	<i>suma 1</i>
<i>ADA, X</i>	
<i>STA, Z</i>	

El "y" lógico: es una operación que se aplica a dos operandos $A \wedge B = Z$. La operación consiste en hallar el producto lógico (bit a bit), no aritmético, de cada par de bits. Entonces:

$$\begin{array}{r} 00110010 A \\ \wedge \underline{00001111} B \\ \hline 00000010 Z \end{array}$$

El producto lógico consiste en multiplicar $a_i \cdot b_i$ para $7 \geq i \geq 0$. Ejemplo:

<i>LDA, A</i>
<i>ANA, B</i>
<i>STA, Z</i>

Una de las aplicaciones es el uso de máscara para indicar qué grupo de bits del operando A se va a reconocer como información en el operando Z.

En este caso, la máscara 0001111 permite que sobre Z se "copien" los primeros 4 bits de A y se anule el resto de la información.

El "o" lógico: es una operación que se aplica a dos operandos $A \vee B = Z$. La operación consiste en hallar la suma lógica (no aritmética) de cada par de bits.

$$\begin{array}{r} 0000\ 1000 A \\ \vee \underline{0011\ 0000} B \\ \hline 0011\ 1000 Z \\ \quad \quad \quad | \\ \quad \quad \quad 3 \quad 8 \end{array}$$

Aplicación: permite agregar los bits de zona para transformar un dígito BCD en el correspondiente carácter ASCII.

El "o" exclusivo: es una operación lógica que se aplica a dos operandos $A \text{ } \underline{\text{o}} \text{ } B = C$. La operación es similar a la suma aritmética, sólo que no se consideran los acarreos al operar cada par de bits.

$$\begin{array}{r} 10101100 A \\ \underline{v} \quad \underline{10101100 B} \\ 00000000 C \end{array}$$

Aplicación: permite comparar si dos operandos son iguales, en cuyo caso el resultado es 0.

7.4.6.5 El registro de estado asociado a la ALU

Cada vez que la ALU realiza una operación elemental, mantiene cierta información que indica el estado final de esa operación. Las instrucciones de código de máquina pueden testear el estado de cada uno de estos bits para tomar decisiones dentro del programa, esto significa que todas estas instrucciones provocan una ruptura de secuencia según sea el valor de la flag 0 o 1.

Esta información queda retenida en un "registro", que se denomina **registro de estado** (*status register*), cuya medida en general depende de la medida de palabra de la computadora. En este registro, cada bit se denomina **flag** (bandera) e indica con un "1" la condición de verdad y con "0" la condición de falsedad de un estado. Por ejemplo, en una computadora de palabra de 8 bits (el ejemplo responde al microprocesador 8080) el registro de estado se encuentra constituido por las *flags* que se detallan en el esquema de la figura 7.27.



Fig. 7.27. Banderas del *status register*.

La *flag S* o *N* es el indicador del signo del resultado. $S = 1$ indica que el resultado es negativo y $S = 0$, que es positivo.

La *flag Z* es el indicador de zero. $Z = 1$ indica que el resultado es cero y $Z = 0$, que es distinto de 0.

La *flag Vu O* es el indicador de *overflow*. $O = 1$ indica que el resultado supera la capacidad prevista para ese formato; $O = 0$, que no se produjo desborde.

La *flag H* es el indicador de acarreo de los 4 primeros bits (utilizado cuando la ALU suma dos dígitos BCD); indica el valor que se tomará como acarreo para el próximo par de dígitos BCD.

La *flag P* es el indicador de paridad (par o impar). Indica 1 si la cantidad de 1 del resultado es impar (en el caso de que la ALU opere con paridad par) y 0 en caso contrario.

La *flag I* es el indicador de interrupción e indica con un 1 si hay un requerimiento de atención de la CPU por parte de un dispositivo externo a ella.

La *flag C* es el indicador del último acarreo (*carry*) de una operación y puede llegar a intervenir en los desplazamientos como un bit más asociado al acumulador.

En la presentación de nuestro modelo de arquitectura se indicó que "X" no era una computadora concreta del mercado actual. Suponemos que "X" lo habrá ayudado a familiarizarse con algunos conceptos importantes sobre el procesamiento de datos con computadoras; sin

embargo, es necesario considerar las variaciones que permiten una mayor eficiencia de esta tarea en computadoras que se alejan del modelo básico. Estas variaciones afectan tanto a los elementos de hardware como a los de software.

7.5 Resumen

Los conceptos enunciados en este capítulo son fundamentales para la interpretación de cada uno de los que siguen; es importante destacar que la función de la CPU en cualquier computadora digital es interpretar y ejecutar el código binario de las instrucciones. Estas instrucciones han sido el producto de un proceso de compilación o de interpretación. El programa original o fuente fue escrito por un programador de aplicaciones en lenguaje de alto nivel, que, por ejemplo, hace treinta años fue "Fortran" y que hoy es alguno de los denominados "lenguajes de cuarta generación", como el ABAP; o bien, "lenguajes de quinta generación" que son aquellos orientados a la inferencia de conocimiento. Todos ellos son "compilables" para la generación del código ejecutable. La CPU también interpreta y ejecuta las instrucciones de los programas del sistema operativo, pues aquellas instrucciones reservadas para determinadas operaciones (y vedadas al programador de aplicaciones) están incluidas en el set de instrucciones que interpreta el procesador.

Las CPU actuales utilizan múltiples registros para el almacenamiento temporal de datos y de referencias a memoria. Están categorizados como registros invisibles, registros de uso exclusivo de las instrucciones privilegiadas del sistema operativo, en los cuales los bits almacenados se relacionan con la administración de la dinámica del sistema. Por otra parte, están categorizados como registros visibles o de propósito general, que son aquellos utilizados por las instrucciones de las aplicaciones. También se agregan a los procesadores actuales memorias de almacenamiento ultra-rápidas denominadas "*caché* de instrucciones" o "*caché* de datos"; estos subsistemas de almacenamiento están incluidos con el fin de adelantar la búsqueda de ambos tipos de objetos (instrucciones y datos) desde su almacenamiento en la memoria principal. El almacenamiento en *cachés* brinda a la CPU una disponibilidad casi continua de instrucciones y datos, esto mejora la velocidad de ejecución, pues obtiene objetos en tiempos acordes a sus necesidades (existe una diferencia notable entre ésta y los tiempos de respuesta de las tecnologías que se utilizan en la memoria principal). La organización de las memorias, las diversas tecnologías y la manera en la que el sistema operativo puede dividir territorios en las distintas áreas de almacenamiento se verán en el capítulo Memorias.

En relación con las instrucciones de máquina o nativas, casi todas ellas se pueden clasificar en instrucciones del tipo memoria-registro, o del tipo registro-registro como veremos en el capítulo Instrucciones. Por ejemplo, si previamente se cargan dos operandos en registros de cálculo de la CPU, se utilizan instrucciones de transferencia memoria-registro (MOVER reg₁, mem), que luego se podrán "operar" con instrucciones aritméticas o lógicas de tipo registro-registro (SUMAR reg₁, reg₂). Puede ocurrir también que una única instrucción involucre tanto la carga en CPU de dos operandos, como la operación entre ellos y el posterior almacenamiento del resultado. Cuanto más rápido se pueda realizar este ciclo, mejor. La velocidad de procesamiento no sólo tiene que ver con la velocidad del reloj, también es necesario un modelo de ejecución adecuado. Esto último tiene que ver con los paradigmas presentados para mejorar el ciclo de ejecución de una instrucción. Al paradigma original enunciado por Von Neumann, se agregó un nuevo modelo de ejecución, RISC (*Reduced Set Instruction Set Code*), cuyos referentes principales fueron, David Patterson, de la Universidad de Berkeley, y John Hennessy, de la Universidad de Stanford.

7.6 Ejercicios propuestos

1) Problema de programación:

En la memoria de nuestra computadora ("X") se almacenaron 3 valores en representación de punto fijo con negativos complementados a 2 en las direcciones: 03A, 03B y 03C.

a) Realizar el programa en Assembler, sumando los 3 valores y almacenando el resultado en la dirección 0FE, si el valor es mayor o igual que 0, o en la dirección 0FF, en caso contrario.

b) Codificar el programa en código de máquina representando los valores binarios en hexadecimal.

c) En una tabla indicar las microoperaciones que permiten la ejecución de cada instrucción y representar los contenidos de los registros especiales AC, PC, MAR, MDR, IR, S, Z, V y C, y en las palabras de memoria 03A, 03B, 03C, 0FE y 0FF.

DATOS:

(03A) = 7F00

(03B) = 91A2

(03C) = 6AB1

Palabra inicial de carga de programa A0A.

2) Ejecutar el programa enunciado en 1) bajo *Debug*, haciendo las modificaciones necesarias respecto de las direcciones de memoria y de las instrucciones, ejecutando los *trace* que permitan ver el contenido de los registros luego de cada instrucción.

Comparar con el desarrollo manual respecto del resultado final y enunciar la conclusión que se desprende de la comparación.

3) Diseñar un control (gráfico de compuertas del circuito) que indique las microoperaciones necesarias para obtener el promedio de 2 números, según el programa:

LDA XXX

ADD XXX

SHR XXX

STA XXX.

Tenga en cuenta que cada microoperación tarda un ciclo de computadora (para "X" 8 señales de tiempo).

4) Diseñe un circuito que, acoplado al reloj maestro, genere una secuencia repetitiva de cuatro señales de tiempo.

5) Diseñe un semicircuito incluido en el control para la habilitación de la microoperación PC + 1 PC.

6) Programe en Assembler la estructura condicional de un lenguaje de alto nivel similar a la siguiente semántica. Considérese que las referencias a memoria para A, B y C son 300, 301 y 302

IF A = B THEN C=A + B ELSE C=A-B

7) Programe en Assembler la estructura iterativa de un lenguaje de alto nivel similar a la semántica siguiente:

FOR I= 1 to 10

x = x +1

NEXT I

8) Represente el algoritmo de Booth para la suma de números que utilicen negativos complementados a 2 en un diagrama de flujo y realice el programa en el Assembler presentado para esta computadora.



7.7 Contenido de la página Web de apoyo

El material marcado con asterisco (*) sólo está disponible para docentes.

Resumen gráfico del capítulo

Animación

Demostración de las distintas fases de la CPU

Autoevaluación

Video explicativo (01:53 minutos aprox.)

Audio explicativo (01:53 minutos aprox.)

Evaluaciones Propuestas*

Presentaciones*

8

Microprocesadores

Contenido

8.1 Introducción	180
8.2 Microprocesadores y microcontroladores.....	180
8.3 Longitud de palabra	182
8.4 Capacidad de direccionamiento.....	182
8.5 Número de instrucciones.....	183
8.6 Número de registros internos.....	184
8.7 Velocidad del microprocesador	188
8.8 Ciclo de instrucciones	188
8.9 Capacidad de Interrupción	192
8.10 Alimentación	199
8.11 Tecnología	200
8.12 Resumen	202
8.13 Ejercicios propuestos.....	203
8.14 Contenido de la página Web de apoyo.....	205

Objetivos

Lograr el aprendizaje de las funciones internas y de entorno de los microprocesadores, relacionadas con la ejecución de programas.



En la página Web de apoyo encontrará un breve comentario de la autora sobre este capítulo.

8.1 Introducción

En este capítulo se estudiarán las características que se han de tener en cuenta para evaluar el rendimiento de un microporcesador.

La mayor preocupación del usuario de una computadora tiene que ver con la velocidad de su CPU. Este parámetro mide la cantidad de trabajo que se realiza por ciclo de reloj; cuando decimos trabajo nos referimos a cuántas operaciones se pueden ejecutar por unidad de tiempo.

Es habitual que se considere que cuanto mayor es la frecuencia del reloj que sincroniza las operaciones, más veloz es la ejecución de instrucciones. Sin quitar la validez de este concepto, se debe considerar que no es el único parámetro que incide en la velocidad de ejecución; por ejemplo, la eficiencia del microcódigo establece la cantidad de operaciones invertidas en la ejecución de una instrucción, puede ser que una CPU realice una multiplicación en muchas menos operaciones que otra y esto mejora la velocidad de ejecución, independientemente del ritmo que impone el reloj. La unidad de coma flotante mejora la velocidad de operaciones que involucran números con coma o números muy grandes respecto de la unidad de coma fija, por lo tanto, cuantas más unidades de coma flotante se repliquen en el diseño, mayor es la posibilidad de ejecución de estas operaciones en paralelo.

Otra característica que representa mayor rendimiento es la cantidad de bits que se pueden operar simultáneamente; es decir que sumar dos números de 64 bits en una CPU con registros de 32 bits, implica que la suma se realice en dos etapas. La cantidad de bits que se operan simultáneamente se denomina palabra de CPU.

Otra alternativa para mejorar el rendimiento es lograr la ejecución de instrucciones en paralelo, esto se conoce como paralelismo a nivel instrucción o técnica *pipeline*. Si un microporcesador incluye memorias internas más veloces que la memoria principal, se pueden anticipar instrucciones y datos que son transferidos desde ella hacia estas memorias temporales conocidas como memorias caché.

En relación con el entorno del microporcesador, hay dos parámetros importantes: la cantidad de bits que se pueden trasferir en paralelo, vía el bus de dato, y la cantidad de bits que puede transferir el bus de direcciones que, generalmente, determina el potencial espacio de direccionamiento en memoria principal al que puede acceder el microporcesador.

8.2 Microporcesadores y microcontroladores

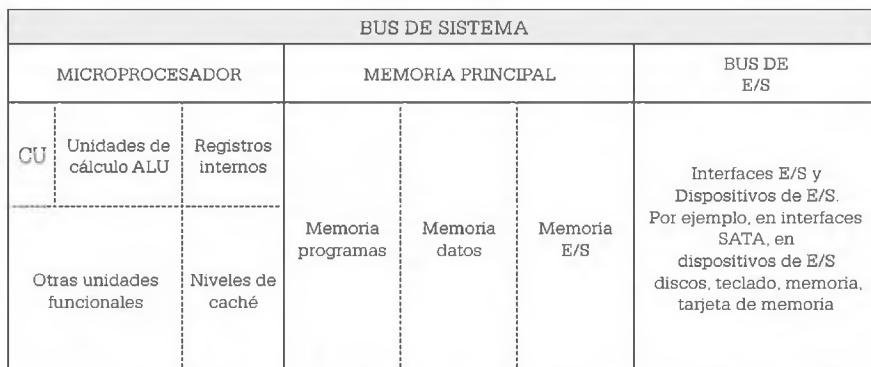
Los circuitos integrados (IC o *Integrated Circuits*), también llamados chips, están constituidos por una base de silicio en la que se encuentran millones de dispositivos electrónicos interconectados (diodos, transistores, resistencias, capacitores, etc.).

Un microporcesador es un chip que junto con la memoria principal, los buses de sistema, los módulos de E/S y los buses de E/S constituyen la estructura de una computadora completa que se integra en una **arquitectura abierta**, ya que es configurable según la necesidad del negocio, tanto desde el punto de vista del hardware como del software.

En la Figura 8.1, se observa el esquema de una computadora basada en un microporcesador en el nivel más abstracto.

Un microcontrolador es una computadora completa con un programa de propósito específico, no configurable para el negocio sino diseñada para suministrar una tarea predeterminada. Los microcontroladores son circuitos integrados que se implementan en el sistema que

controla; pertenecen a lo que se denomina arquitectura cerrada. Un ejemplo de ello son los controladores PLC (*Programmable Logic Controller* o controlador lógico programable)



Un microcontrolador es una computadora completa con un programa de propósito específico

Fig. 8.1. Esquema de una computadora basada en un microporcesador.

8.2.1 Chips y microporcesadores

Los microporcesadores son ejemplos de chips muy avanzados que varían en velocidad y capacidad de direccionamiento a memoria, registros, niveles de caché, capacidad de transferencia de datos, etcétera. También se encuentran microchips en las placas de memoria (por ejemplo, placas de memoria dinámica) y en las placas de expansión (por ejemplo, placa gráfica).

Como ya indicamos en el capítulo Evolución del procesamiento de datos, Von Neumann estaba constituida por componentes básicos: una memoria, una unidad aritmético-lógica, una unidad de control del programa y los dispositivos de E/S. En un ejemplo clásico de computadora Von Neumann, si la memoria estaba constituida por 4096 palabras de 40 bits, cada palabra podía contener dos instrucciones de 20 bits o un número entero de 39 bits y su signo; las instrucciones tenían 8 bits dedicados al verbo o código de operación, o "lo que la CU debe ejecutar", y los 12 bits restantes para dirigir una entre las 4096 palabras de la memoria principal.

En la única unidad aritmético-lógica había un único registro interno denominado acumulador de 40 bits. Esta ALU no operaba con una unidad de coma flotante porque la resolución de los algoritmos de estas operaciones se sustentaba con la ejecución de rutinas software (p. ej., una suma en punto flotante implicaba alinear las mantis en instrucciones de desplazamiento e incrementar uno de los exponentes hasta conseguir que los exponentes fueran iguales, en cuyo caso se procedía a la suma de las mantis).

En este ejemplo, el procesador de la máquina Von Neumann contiene las tres partes básicas: CU, ALU y un registro AC, funciones agrupadas bajo el término CPU.

En la actualidad, un microporcesador tiene más componentes que la CPU enunciada, por ejemplo, un banco de registros, varias unidades de coma flotante (que operan a modo de "co-procesador", asistiendo a la CPU con este tipo de operaciones) y demás funciones que iremos detallando a continuación.

Para sustentar físicamente al microporcesador y otros chips, por ejemplo, en el caso de una computadora personal, se incluye una placa madre o placa principal (*mother-board*), que se utiliza como soporte y elemento de comunicación entre los distintos componentes que se encuentran en el entorno del microporcesador.

8.3 Longitud de palabra

Un microporcesador "procesa" bits que obtiene de una memoria, los opera y luego almacena el resultado de nuevo en memoria. Hablamos de "una memoria", puesto que ya sabemos que la memoria de una computadora es la memoria principal, los datos surgen inicialmente de allí y los resultados terminan allí de manera definitiva, aun cuando el microporcesador cuente con sus propios registros e incluso con memorias caché y alojadas en su interior. La cantidad de estos bits que "procesa" como grupo es su longitud de palabra.

Básicamente en el mercado actual un microporcesador puede procesar internamente grupos de 32 o 64 bits. "En teoría" se puede pensar que un microporcesador que procesa grupos de 64 bits es 2 veces más eficiente que uno de 32 bits. Por otra parte, se puede hacer referencia a software de 32 bits o de 64 bits, en relación con la manera en que se ha compilado un programa para generar su código de máquina, y es entonces que muchos microporcesadores de 64 bits admiten la ejecución de software compilado para arquitecturas de 32 bits. Por supuesto que un sistema de 64 bits ejecutando un programa de 32, desaprovecha su rendimiento, pues, para dar un ejemplo, 2 operandos de 64 bits que se quieran sumar lo harán sobre registros de 32 bits en 2 etapas. Esto significa que muchas veces el aumento de la longitud de palabra no significa incremento en la velocidad de ejecución. En la actualidad, muchos de los microporcesadores distribuidos por los fabricantes Intel y AMD, por citar sólo 2, son de palabra de 64 bits, como el AMD 64 y el Intel Itanium.

En el pasado, INTEL fabricaba un microporcesador de 8 bits, luego desarrolló el "286" de 16 bits; después marcó profundamente su ubicación privilegiada en el mercado con la IA-32, que se inicia con el microporcesador "386" y llega hasta nuestros días, sobre todo en microporcesadores instalados en computadoras de escritorio y portátiles. Como vemos, la longitud de palabra se relaciona con la capacidad de trabajo simultáneo expresado en bits y con el desarrollo de nuevas tecnologías.

Es momento de indicar que el término "palabra", que en el lenguaje cotidiano se refiere a un conjunto de letras con sentido propio, en Informática tiene varios significados.

El término "palabra" se utiliza también cuando hablamos de estructuras de datos; en ese caso, se considera una "palabra" un grupo de 16 bits. Así, en el capítulo Representación de datos en la computadora, se mencionaron los términos palabra, doble palabra, cuádruple palabra, como "tipos de datos", y es necesario no confundir el concepto de esta definición, asociado con datos, con el término palabra de microporcesador, asociado con su capacidad de trabajo.

A su vez, se indica que una "palabra de memoria" es la cantidad de bits a los que se puede acceder por vez, lo que no necesariamente debe coincidir con el concepto de palabra al que nos referimos aquí.

Un programa que esté compilado con un compilador de 32 bits "considera" que un dato u operando de 64 bits está almacenado en dos palabras de memoria de 32. Entonces, el código ejecutable generará dos instrucciones de lectura a memoria en lugar de una, o sea que para un buen uso de la longitud de palabra del microporcesador lo sugerido es recomilar los programas fuente con "compiladores de 64 bits".

8.4 Capacidad de direccionamiento

La capacidad de direccionamiento del microporcesador tiene relación con el acceso a líneas que transfieren direcciones (las denominadas direcciones físicas, absolutas o efectivas) a la memoria. Una dirección permite individualizar la zona de memoria principal que

se va a leer o escribir. Estas líneas identificadas como bus de direcciones determinan el potencial de direccionamiento del microprocesador, que significa “cuánta memoria como máximo podría visualizar el microprocesador”. Siguiendo la regla, una dirección de dos bits permite determinar 4 direcciones diferentes, 3 a 8 y 4 a 16. Concluimos que con n bits del bus de direcciones se obtiene un mapa de direcciones de 2^n combinaciones distintas o “direcciones” distintas; por eso, con 32 líneas de bus por las que se transfieren 32 bits de dirección se podría acceder a $2^{32} = 2^{22} \cdot 2^{30} = 4 \text{ giga}$ direcciones, y con 33 a $2^{33} = 2^3 \cdot 2^{30} = 8 \text{ giga}$ direcciones. Esto permite deducir que si el microprocesador aumenta en una la cantidad de líneas o, mejor expresado, a pines asociados a una línea, entonces duplica su posible capacidad de direccionamiento.

8.5 Número de instrucciones

La cantidad de instrucciones diferentes que un microprocesador “puede entender” o, en términos técnicos, decodificar, y por lo tanto ejecutar, en la mayoría de los casos implica su mejor capacidad para “hacer cosas distintas”. Demos un ejemplo básico. Si un microprocesador tiene instrucciones de desplazamiento (SAR), suma (ADD) y consulta de banderas (JZ) puede realizar una multiplicación de operandos con sumas sucesivas y desplazamientos.

Hacemos referencia aquí al algoritmo de Booth para la multiplicación de enteros signados. La representación en código de máquina de este algoritmo implica la ejecución de varias instrucciones que involucran numerosos ciclos de reloj y múltiples microoperaciones para llevarlas a cabo. Si, por ejemplo, la secuencia de microoperaciones del mismo algoritmo se programara en una memoria ROM, es posible que la cantidad de ellas disminuya en grado notable.

Lo que sucede es que cada instrucción tiene un número predeterminado de microoperaciones y si se las agrupa como los “ladrillos” o “bloques” de construcción de un programa, serán consideradas paquetes de microoperaciones, lo que tal vez genere una secuencia de microoperaciones única y equivalente, en cuanto a su función, a las de las instrucciones originales en el algoritmo, no todas ellas necesarias cuando se las unifica. Por lo tanto, es posible que diseñando el algoritmo a nivel microoperación resulte más eficiente simplificar el número de **órdenes** (aquí no entramos en discusión respecto del tiempo que se tarda en acceder a la ROM).

Una vez que el algoritmo está programado en la ROM de microcódigos, tenemos una instrucción nueva que se incluirá en el set de instrucciones aritméticas; ésta contará con su propio código de instrucción y se llamará, por ejemplo, MUL. Cuando el programador deba realizar una multiplicación, ya no debe programar el algoritmo sino que simplemente indica “*MUL op1, op2*”. A continuación, veamos la primera propuesta generada por Maurice Wilkes para el diseño de unidades de control microprogramadas.

En la unidad de control microprogramada del **modelo de Wilkes** se propone un nivel de programación inferior a la del programa ejecutable, que consiste en la programación de microoperaciones, o programación de señales de control o programación de microcódigo.

La microprogramación fue propuesta por Maurice V. Wilkes en la década de 1950 como un modelo de diseño para unidades de control de computadoras.

En una memoria ROM se codificaban todas las microinstrucciones pertenecientes a una instrucción de máquina. La dirección inicial, o dirección de inicio del microcódigo de esa instrucción, depende del código de operación que ingresa en el decodificador de la ROM $m \cdot n$. Los m bits de las palabras de la memoria ROM de control tenían dos campos, uno con las señales de control u órdenes (las microoperaciones) y el otro para indicar la próxima dirección de



Vínculo a la página personal de Maurice V. Wilkes disponible en la página Web de apoyo.

la ROM para acceder. Cada palabra de la ROM se denominó microinstrucción y el conjunto de ellas que permitían la ejecución de una instrucción de máquina, microcódigo. El valor de la dirección de cada microinstrucción convocaba a la próxima y, eventualmente, podía modificarse en función de las banderas de estado, por ejemplo, si la instrucción era de salto condicionado. El modelo de Wilkes permitió que los procesadores aumentaran la cantidad de instrucciones diferentes que la CU podía interpretar, y a su vez facilitó el aumento de la complejidad de las nuevas. De esta manera se desarrollaron los procesadores CISC o *Complex Instruction Set Computer*.

El número de instrucciones de un microporcesador tiene relación con su tecnología RISC, CISC y EPIC, como veremos en el apartado Tecnología.



Se define como registros visibles a aquellos que pueden ser accedidos por el repertorio de instrucciones disponibles para el programador de aplicaciones.

8.6 Número de registros internos

Se refiere a la cantidad de registros con los que cuenta el microporcesador, cuya función es sustentar las necesidades de almacenamiento temporal durante la ejecución. Al conjunto de registros que pueden actualizarse por las aplicaciones se lo denomina registros para el programador de aplicaciones o registros visibles. Sin embargo, existen registros que se utilizan para la administración del sistema y que son de alcance exclusivo de instrucciones de mayor nivel de privilegio. A estos registros, a los que se puede acceder por medio de los programas del sistema operativo, se los denomina registros para el programador de sistema o registros invisibles.

A continuación, se presentan dos casos ejemplos: uno sobre un microporcesador ya en desuso de 16 bits y su correspondencia con uno del mercado actual de 32 bits; y el otro con los registros de un microporcesador de 64 bits. Ambos casos son del fabricante Intel, elegido para observar la evolución en la misma línea y, además, dado que los registros se identifican con un nombre, para evitar cambios de identificación.

Caso ejemplo 1

Se presenta este caso por ser el de más amplia difusión y porque, aún en el presente, es posible "visualizar" los registros en computadoras personales con versiones de sistemas operativos que admitan el modo 16 bits, o modo consola, o lo emulen con un software provisto por el sistema operativo.

En las plataformas de 16 bits del fabricante Intel se describió un juego de 14 registros de uso del programador de aplicaciones. Esta descripción mejora la presentada en el capítulo Diseño de una computadora digital, y la acerca a una realidad del mercado no tan lejana pero, a la vez, no tan compleja como la de los microporcesadores de 64 bits, que si bien tienen más registros, éstos cumplen conceptualmente las mismas funciones. Se dará un ejemplo de ellos en el caso ejemplo 2.

Desde su concepción, el microporcesador de 16 bits permitió realizar dos acciones críticas en forma paralela en el ciclo de ejecución de instrucciones, que son la búsqueda y la transferencia de instrucciones (*fetch*) y su ejecución (*execute*), para lograr maximizar el rendimiento general de la CPU.

Para lograrlo se utilizó una estructura interna de ejecución en cadena u operación en cascada, técnica conocida como estructura tubular o pipeline, o en cascada o de cola, que consiste en dividir el procesamiento de cada instrucción en etapas y que éstas operen en paralelo. De este modo, en forma simultánea con la etapa final de ejecución de una instrucción n se puede buscar en memoria la instrucción $n+1$. Este concepto se utiliza en los microporcesadores actuales y por eso daremos el ejemplo más sencillo para conocer su funcionamiento.

El X86 contaba con dos unidades funcionales. Una se llamaba **unidad de ejecución** (EU o *Execution Unit*), que específicamente se encargaba de decodificar y ejecutar las instrucciones.

Durante la ejecución de un programa, la EU utilizaba los registros de cálculo y el registro de estado como almacenamiento temporal de datos e información de estado final de operaciones o de estado del microprocesador. Como esta unidad no tenía conexión directa con los buses, se lo solicitaba a la **unidad de interfaz** con el bus. Esta función, BIU (*Bus Interface Unit*), se encargaba no sólo de obtener y almacenar datos en memoria principal, sino también de obtener las instrucciones del programa.

En este caso de estudio las instrucciones son leídas previamente por la BIU y almacenadas en una cola de instrucciones con estructura FIFO (*First In, First Out*), o sea, la primera que ingresa en la cola es la primera que sale y se ejecuta. Hacemos esta referencia para indicar su relación con los registros que presentaremos más adelante. La BIU se relaciona con los registros de segmento y con los registros punteros (p. ej., el IP o *Instruction Pointer* o puntero de instrucción).

El siguiente es el esquema de registros que presenta el microprocesador de 16 bits y del mismo fabricante; su extensión para la plataforma de longitud de palabra de CPU es de 32 bits. Tenga en cuenta que una arquitectura de 32 bits requiere un tamaño de registros de 32 bits. La diferencia de nombre para uno u otro registro es que se antepone una E a los registros "extendidos".

Debe aclararse que el software compilado para sistemas de 16 bits sólo "visualizará" los registros de 16 bits, aun cuando la arquitectura sea de 32, y que el software compilado para sistemas de 32 bits "visualizará" tanto unos como otros. Así, una instrucción en modo 16 puede ser "*ADD AX, BX*"; en modo 32 será idéntica, puesto que sólo hace referencia a la operación de 16 bits en registros del mismo tamaño (p. ej., la suma de dos variables *integer*). Esta instrucción sólo podrá sumar los valores alojados en el registro *AX* y *BX*, dejando un resultado en *AX*. En cambio, una instrucción que suma en forma simultánea los valores binarios de dos registros de 32 bits debe escribirse "*ADD EAX, EBX*", haciendo referencia a los "registros extendidos".

Aquí volvemos a indicar que un programa compilado en 16 bits puede ejecutarse en un micro de 32, pero toda operación que involucre datos de 32 bits se realizará en dos pasos des aprovechando la capacidad del micro de 32 que lo haría sólo en uno.

8.6.1 Registros de uso general IA-16 e IA-32

Presentaremos primero los registros de cálculo de una IA-16 y luego el conjunto completo para una IA-32.

8.6.1.1. Registros de cálculo de 16 bits

Tabla 8-1. Registros de cálculos de 16 bits.

15	8	7	0	
AH		AL		Ax Registro acumulador (operación E/S y de cadena)
BH		BL		Bx Registro base (registro base para direccionamiento)
CH		CL		Cx Registro contador (para bucles, iteraciones, desplazamientos y rotaciones)
DH		DL		Dx Registro para datos (almacenado de datos, direcciones de puertos, extensión de Ax en multiplicación y división)



Debe aclararse que el software compilado para sistemas de 16 bits sólo "visualizará" los registros de 16 bits aun cuando la arquitectura sea de 32.

Todos los registros, independientemente del detalle de función indicado antes, se pueden utilizar en operaciones de cálculo aritmético o lógico. Es factible emplearlos con instrucciones que los afecten en su totalidad, *AX*, o que afecten la parte alta, *AH*, o la baja, *AL*.

	15	8 7	0
MOV AX, 1234	AX	12	34
MOV AH, 12	AH	12	
MOV AL, 34	AL		34

En particular, los registros de cálculo en arquitecturas de 32 bits se utilizan de forma análoga a los presentados, excepto que tienen el doble de capacidad de almacenamiento.

Estos registros son capaces de soportar datos de 8, 16 y 32 bits. Los cuatro registros de cálculo en una IA-32 son *EAX* (acumulador), *EBX* (base), *ECX* (contador) y *EDX* (datos).

8.6.1.2 Registros punteros

Son registros utilizados para desplazarse dentro de un bloque o zona de memoria. Si el bloque es de código, "IP" o "EIP", si es un bloque de pila, "SP" o "ESP" y "BP" o "EBP", y si es un bloque de dato que tiene definida una estructura de dato que se puede recorrer con un índice, "SI" o "ESI", "DI" o "EDI".

Tabla 8-2. Registros punteros.

	31	16	15	0	
EIP					IP Registro puntero de instrucción.
ESP					SP Registro puntero de pila. Modo de direccionamiento a la pila.
EBP					BP Registro base para pila. Modo de direccionamiento base a la pila.
ESI					SI Registro índice fuente. Modo de direccionamiento indexado.
EDI					DI Registro índice destino. Modo de direccionamiento indexado.

8.6.1.3 Registro de estado

	31	16	15	0	
EFLAGS					F

En este registro se alojan, por nombrar algunas, todas las banderas aritméticas, banderas de modo de trabajo del microprocesador, banderas asociadas a interrupciones, etcétera.

8.6.1.4 Registros de segmentos

Son registros concebidos para brindar soporte para aquellos sistemas operativos que administran la memoria como una agrupación de segmentos, esto es, que utilizan un modelo segmentado. Un segmento es un bloque lógico de tamaño ajustado al objeto que contiene. Se denomina objeto al código de un programa, los datos que utiliza y una estructura de dato denominada pila. Es común que todo programa ejecutable almacenado en memoria bajo un



Registro de estado: en este registro se alojan, por nombrar algunas, todas las banderas aritméticas, banderas de modo de trabajo del microprocesador, banderas asociadas a interrupciones, etc.

modelo segmentado esté constituido por un segmento de código, de uno a cuatro segmentos de datos y un segmento de pila.

Los registros de segmento almacenan la referencia binaria a la base de un segmento en memoria, esto es, donde empieza la zona de memoria para ese objeto, tendremos seis de ellos.

Tabla 8-3. Registros de segmento

15	0	
		Registro de base de segmento de código
		Registro de base de segmento de pila o <i>stack</i>
		DS
		ES
		FS
		GS

} Registros de base de segmentos de datos

Los registros de segmento almacenan la referencia binaria a la base de un segmento en memoria, esto es, donde empieza la zona de memoria para ese objeto, tendremos seis de ellos.

Al ser el segmento un bloque lógico en memoria, además de la base requiere un desplazamiento que permita "recorrerlo". En las IA-16 los registros de segmento hacían referencia al comienzo de un área de memoria de hasta 64 KB y el puntero de instrucciones IP "recorriá" el segmento de código, al tener 16 bits, y podía alcanzar un desplazamiento máximo de 65535 direcciones. Cada una de ellas identifica un byte.

Durante la fase de búsqueda de instrucción se utilizaba el registro CS como base y el registro IP, generando una "dirección segmentada" con los valores del par CS:IP, para así obtener la dirección física "dentro" del segmento de código.

Las instrucciones de referencia a dato alojado en un segmento de dato utilizan los registros DS y ES y el campo DATA de la instrucción en ejecución (DS: DATA y ES: DATA). Las instrucciones de referencia a dato alojado en un segmento de pila utilizan el par SS:SP.

Luego, en las arquitecturas IA-32 se utilizan registros de EIP y ESP de 32 bits para instrucciones y pila, respectivamente, mientras que para dato los registros de segmento son cuatro, DS, ES, FS, GS, utilizando el campo EDATA como desplazamiento. Esto permite a cada programa asociarse a un segmento de código, uno de pila y 4 para contener sus datos como máximo.

En las arquitecturas IA-32 se utilizan registros de EIP y ESP de 32 bits para instrucciones y pila.

8.6.1.5. Relación entre los registros y el modo de direccionamiento a datos

En el capítulo Instrucciones se enunciará lo que se describe a continuación; allí, desde la concepción de la obtención de datos durante la ejecución de instrucciones; aquí, desde la funcionalidad de los registros que actúan de soporte para que esto sea posible. La menor cantidad de modos de direccionamiento indicada en los manuales del microprocesador Intel de 16 bits es de 25, descriptos en una tabla del capítulo Memorias; éstos surgen de su combinación de acuerdo con la fórmula siguiente:

$$\text{direc. de segmento} + \text{direc. base} + \text{índice} + \text{desplazamiento}$$

La base de un segmento (que no es de código, porque los modos sólo afectan los datos) se encuentra en los registros de segmento DS, ES o SS y se multiplica por 16 antes de sumarse al resto de los argumentos enunciados en la fórmula.

El valor de una base para direccionamiento base "dentro" del segmento se almacena en cualquiera de los registros base BX o BP; el valor del índice que recorre una estructura de dato se encuentra en el uno de los registros índice SI o DI y el desplazamiento puede ser una

constante de *B*, de 16 bits o de *I* bit en el caso de que no exista ese argumento en el esquema de direccionamiento.

La base y el índice son valores que se pueden modificar durante el procesamiento, ya que se encuentran como registros de propósito general para el programador de aplicaciones, pero el desplazamiento es una constante que se define en el momento del ensamblado del programa y no puede cambiarse durante su ejecución. Un ejemplo con una instrucción de transferencia puede ser así:

MOVAX, [BX+SI+0A]

Caso ejemplo 2. Registros en microporcesadores Itanium de 64 bits

Son 128 de 64 + 1 bits, identificados como "*CRn*"; 64 bits disponibles para el valor y un bit extra que indica si la información en él es válida. Para coma flotante son 128 de 82 bits, identificados como "*FRn*", que reutilizan para los datos que serán "operando" en cálculos de coma flotante.

Los registros de predicción son 64 de 1 bit cada uno, identificados como "*PRn*", y se utilizan para el control de ejecución de instrucciones de salto condicional.

Los registros "rama" son 8 de 64 bits y especifican destinos en las direcciones de ramas llamados *BR0 BR1 BR2 BR3 ... BR7*.

Hay un registro de control de iteraciones llamado *loop counter*.

También, un registro puntero de instrucción de 64 bits denominado "*XIP*" y un registro de banderas denominado "*XPCR*".

Los registros de gran longitud reducen la complejidad del hardware; disminuyen la expansión del código fuente que utiliza tres registros base rotativos. Su uso minimiza la cantidad de accesos a memoria caché.

Además de los registros descriptos, se agregan todos los utilizados en las arquitecturas anteriores y que ya se detallaron. Así, este microporcesador admite la ejecución del software de 32 bits.

La familia de procesadores Itanium es el grupo de productos de más alto desempeño y más confiable que ofrece Intel para los sistemas MP (multiprocesados) o de varios núcleos.

Los nuevos registros de este microporcesador, tanto los registros rama (*branch*) como los de predicción, merecen una explicación en detalle en el apartado EPIC.

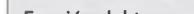
8.7 Velocidad del microporcesador

Cada microporcesador tiene su propio reloj interno, cuya frecuencia indica con qué velocidad puede procesar bits; ésta se expresa en *Hz*, más precisamente en nuestros días, en millones de *Hz* = *MHz*, o miles de millones de *Hz* = *GHz*.

Sin embargo, se advierte que la velocidad no es el único factor que asegura la velocidad de ejecución de programas, sino que es un parámetro más; por lo tanto, no se debería utilizar como único para comparar microporcesadores diferentes. Por ejemplo, microporcesadores de tecnología CISC difieren respecto de los de tecnología RISC en su rendimiento; esta característica se tratará en el apartado Tecnología.

8.8 Ciclo de instrucciones

En el capítulo Diseño de una computadora digital, se describió un ciclo de instrucción para las n instrucciones de un programa a un nivel de abstracción como el que se muestra en el esquema siguiente.



Función del temporizador (*timer*)

El temporizador es una función que se aloja en el *chipset* de la placa base, que incorporamos en este momento porque su función depende del reloj. La función específica del *timer* es mantener una cuenta de tiempo basada en el reloj. Se utiliza para conteo, cálculo de intervalos de pausas o para generar pausas. O sea que se puede medir el tiempo real de CPU que utiliza un programa al ejecutarse (cálculo de intervalo de tiempo) o, para dar un ejemplo de función de conteo, también puede indicarse que consiste en cargar un registro con cierto valor e ir decrementándolo por cada ciclo de reloj hasta que llegue a 0, momento en el que el *timer* genera una interrupción. Esta función de cuenta se puede utilizar para mantener actualizadas la hora y la fecha; la actualización se produce cada vez que se genera la interrupción y la cantidad de veces que se actualiza por segundo depende de la velocidad del reloj y el valor cargado en el registro.



Fig. 8.2. Ciclo de instrucción para las n instrucciones de un programa.

En los microprocesadores del Caso ejemplo 1 mencionado, el ciclo también dura dos fases, más complejas por las características del microprocesador, pero que pueden representarse con el mismo esquema.

La primera fase consiste en la captura de la instrucción desde memoria principal y está a cargo de la mencionada BIU, que entrega la instrucción a la unidad de ejecución, EU. Sin embargo, la ejecución del programa descripto a continuación en código de máquina y Assembler de 16 bits nos permitirá introducirnos en un nuevo concepto que mejora la habilidad del microprocesador para optimizar este ciclo de ejecución de instrucción, el ya mencionado *pipeline* o segmentación de instrucciones.

Éste es el programa de ejemplo:

1531:0100	8B1E0002	MOV	BX,[0200]
1531:0104	0307	ADD	AX,[BX]
1531:0106	D1E0	SHL	AX,1
1531:0108	83C302	ADD	BX,+02

Este programa se halla almacenado en un segmento de código, cuya base es 1531. La primera instrucción es de transferencia y está ubicada a partir del desplazamiento 0100 respecto de 15310 (recuérdese que el cálculo de dirección física en modo 16 bits multiplica por 16 en decimal o 10 en hexadecimal, en consecuencia, $1531 \cdot 10 = 15310$).

La instrucción ocupa cuatro bytes, 0100, 0101, 0102, 0103; los dos primeros corresponden al código de operación 8B1E y los dos últimos al campo DATA almacenado según la convención *little endian* 0002 (o almacenamiento invertido que, es equivalente a 0200). En una instrucción que requiere la lectura de un dato de memoria alojado en el desplazamiento 0200 y su carga posterior en el registro general BX, el modo de direccionamiento sería directo.

La segunda instrucción es aritmética, comienza en el byte 0104 y ocupa dos bytes, el 0104 y el 0105, que corresponden al código de operación 0307. Su ejecución suma al registro general AX el valor de BX (los corchetes de la instrucción indican "contenido de") y no accede a memoria para buscar dato; el modo de direccionamiento sería directo de registro.

La tercera instrucción es de desplazamiento, comienza en el byte 0106 y ocupa dos bytes, el 0106 y el 0107, correspondientes al código de operación D1E0, que permite el desplazamiento

to de 1 bit a la izquierda (*shift left*) del registro AX, y, por lo tanto, no accede a la búsqueda de un dato en memoria principal; el modo de direccionamiento sería inmediato a registro.

La cuarta instrucción es aritmética. Comienza en el byte *0108* y ocupa tres bytes, el *0108*, el *0109* y el *01A*, los dos primeros correspondientes al código de operación *83C3* que efectúa la suma del valor *02* al registro BX; no accede a memoria para obtener el dato y pertenece al modo de direccionamiento inmediato a registro.

El programa no tiene sentido lógico, pues es una rutina de un programa mayor. Ésta ocupa entonces once bytes en el segmento de código en la memoria principal. Consideremos que asociada a la BIU se encuentra una estructura cola (FIFO), utilizada como *buffer* de 6 bytes, de modo que la BIU vaya cargando en ella de a 2 bytes por vez, si consideramos un bus de datos de 16 bits, como se puede observar en el esquema secuencia de llenado de la cola. La cola es una línea de espera de instrucciones, lo que implica que éstas llegan al microprocesador antes de lo necesario.

8B1E					
	0002				
		0307			
			D1E0		
				83C3	
					02XX

8.8.1 Secuencia de llenado de la cola

De lo expresado surge que la BIU se encarga de controlar la transferencia entre el entorno (en este caso la memoria principal) y el microprocesador, y a su vez "entrega" las instrucciones a la unidad de decodificación a medida que se necesiten. A continuación, se describe el llenado y el vaciado de la cola hasta que no hay más instrucciones de la rutina de ejemplo. Consideraremos una cola de sólo 6 bytes y una única rutina de instrucciones para ejecutar.

Secuencia de llenado y vaciado de la cola de 6 bytes de a 2 bytes, en función de la demanda de ejecución del bus de dato de 16 bits:

Tabla 8-4. Secuencia de llenado y vaciado de la cola de 6 bytes de a 2 bytes.

	XX vacía	Estado inicial					
Hacia unidad de decodificación	XX vacía	XX vacía	XX vacía	XX vacía	8B	1E	Desde memoria
	XX vacía	XX vacía	8B	1E	00	02	
	8B	1E	00	02	03	07	
	00	02	03	07	D1	E0	
	03	07	D1	E0	83	C3	
	D1	E0	83	C3	02	XX vacía	
	83	C3	02	XX vacía	XX vacía	XX vacía	
	02	XX vacía					
	XX vacía	Estado final					

8.8.2 Etapas de ejecución de la rutina ejemplo

Tabla 8-5. Etapas de ejecución de la rutina ejemplo.

Ciclo 1	Ciclo 2	Ciclo 3	Ciclo 4	Ciclo 5	Ciclo 6	Ciclo 7	Ciclo 8	Ciclo 9	Ciclo 10	Ciclo 11
FetchI1 8B1E		FetchI1 0002	FetchI2 0307		FetchI3 D1E0		FetchI4 83C3		FetchI4 02XX	
	DECO I1			DECO I2		DECO I3		DECO I4		
			EU I1	EU I1	EU I1	EU I2	EU I2	EU I3		EU I4

La unidad de decodificación, asociada a la BIU por un extremo y a la EU, es la que interpreta el código de operación y, por lo tanto, "reconoce" el verbo de la instrucción, cuántos bytes mide en total para solicitarlos si es necesario y cómo se obtiene el dato según el modo de direccionamiento especificado en el código de operación. En función de estas actividades genera la demanda de cola. Como esta unidad es independiente de la EU puede darse el caso de que mientras se decodifica una instrucción todavía se esté ejecutando la anterior, lo que produce así el solapamiento en la fase *execute* de dos instrucciones; esta particularidad genera el **paralelismo** a nivel ejecución de instrucciones, ya que las dos están en la misma fase pero en diferentes instancias de ella.

Como se observa en la tabla anterior, la unidad de decodificación (en adelante DECO) solicita a la BIU 2 bytes (*8B1E*) durante el ciclo 1; una vez decodificado durante el ciclo 2 "reconoce" que la instrucción tiene 2 bytes más y los solicita a la BIU (*0002*), ciclo 3; con ese pedido la EU puede calcular la dirección del dato en la memoria principal, genera la lectura de esta dirección y la carga en el registro BX. Se consideró que esta etapa de ejecución que implica el acceso a dato en memoria tarda tres ciclos de ciclo 4, ciclo 5 y ciclo 6.

Mientras transcurre el ciclo 3 la DECO no puede decodificar nada, pues todavía le falta una parte de la instrucción 1, y la EU no puede ejecutar nada hasta no tener el campo DATA de la instrucción que le permite calcular la dirección del dato. Durante este ciclo ambas unidades permanecen ociosas.

En el ciclo 4 la BIU vacía la cola con los siguientes 2 bytes y durante el ciclo 5 se decodifica la instrucción 2 en paralelo a la ejecución de la instrucción 1, que aún no concluyó. Cada vez que DECO finaliza una decodificación, solicita 2 bytes más de la cola. Todas las instrucciones que siguen operan de la misma manera y, de este modo, las cuatro instrucciones se terminan de ejecutar en 11 ciclos de reloj. En un modelo que no fuese segmentado como es éste, cada instrucción debería estar ejecutada por completo para pasar a la búsqueda de la siguiente. De acuerdo con los tiempos establecidos en la tabla anterior, si no hubiera habido paralelismo, el programa hubiera tardado 6 ciclos para la instrucción 1, 4 para la 2, 3 para la 3 y 4 para la 4. Un total de 17 ciclos. Por lo tanto, se deduce que a mayor cantidad de unidades funcionales que puedan "operar" distintas etapas del ciclo de ejecución de la instrucción, más probabilidad de solapamiento y, en consecuencia, un mayor grado de paralelismo.

En una situación "ideal", la tendencia es lograr la ejecución de una instrucción por ciclo de reloj. Esto implica instrucciones de igual complejidad y tamaño. Así, cuando en un ciclo de reloj T_i entra una instrucción al *pipeline*, en el mismo T_i se termina de ejecutar otra. Resulta que si bien no se reduce el tiempo de ejecución de una instrucción (cada una requiere n ciclos de reloj), en cada ciclo T_i se ejecuta una de las n etapas de instrucciones distintas, lo que permite aprovechar "de modo óptimo" cada ciclo de reloj y, por lo tanto, aumenta la velocidad de ejecución global.

El objetivo es extraer una instrucción de memoria (que es lo que más tiempo tarda en llevarse a cabo, ya que depende del tiempo de acceso a memoria), decodificarla y redirigirla a unidades funcionales de la EU. Mientras estas unidades "ejecutan" la instrucción, deberá extraerse y decodificarse otra instrucción para redirigirla a otras unidades de la EU no comprometidas en la ejecución de la anterior. Así, cuantas más unidades formen parte de la EU mayor será la posibilidad de "ejecución en paralelo". Este procedimiento es casi óptimo hasta que aparece una instrucción de salto. Según vimos en el capítulo 6, una instrucción de salto o bifurcación rompe la secuencia normal de ejecución, esto es, de "una tras la otra". Un salto implica que la próxima instrucción que se ha de ejecutar no es la siguiente sino que hay que buscar aquella cuya referencia a memoria suele estar indicada en la propia instrucción de



La unidad de decodificación, asociada a la BIU por un extremo y a la EU, es la que interpreta el código de operación y, por lo tanto, "reconoce" el verbo de la instrucción.

salto. Por lo tanto, todas las instrucciones que están en "fase de ejecución" posteriores a la de salto "se ejecutaron en vano". Si se considera posible que entre 25% y 30% de las instrucciones en un programa es de salto y que se puede aumentar la velocidad de ejecución a causa del *pipeline* entre 4 y 5 veces más, entonces la pérdida se equilibra con la ganancia.

Es importante entonces el estudio de los distintos saltos que puedan producirse y de las posibles estrategias que se han de seguir para minimizar su impacto.

Una primera observación que podemos hacer es que hay saltos incondicionales, *JMP XXX*, saltos condicionales, *JZ XXX*, además de retornos de rutinas RET, saltos por iteraciones, *LOOP XXX*, *CALL XXX*, o llamado a rutina.

Se puede utilizar una estrategia de "penalización por salto". Esto implica que se considera que ningún salto se va a producir y que en el caso de que se produzca, se perderá lo ejecutado *a posteriori* y se vaciará el cauce de ejecución. La EU debe contar con registros especiales e invisibles al programador de aplicaciones, para resguardar el entorno de ejecución al momento en que se produzca el salto.



Las interrupciones y las excepciones son acontecimientos causados tanto por los dispositivos de E/S como por el programa que se ejecuta en el microporcesador.

8.9 Capacidad de interrupción

Las interrupciones y las excepciones son acontecimientos causados tanto por los dispositivos de E/S como por el programa que se ejecuta en el microporcesador y su efecto produce una suspensión de la actividad actual del micro, para pasar a ejecutar un servicio que "interprete el manejo de esa interrupción".

Los dispositivos externos utilizan interrupciones para informar su estado o solicitar la ejecución de actividades que le son necesarias. Por ejemplo, "solicito que la CPU ejecute mi driver". Los programas a su vez solicitan información de los dispositivos de E/S, por ejemplo, para obtener un bloque de información de un disco.

Cada interrupción está asociada con un número que la identifica; éste permite convocar al servicio que la atiende, que puede ser provisto por el sistema operativo o, en las computadoras que albergan una BIOS, por un servicio de la BIOS (*Basic Input Output System*).

Si un programa quiere provocar una interrupción para acceder a un disco o a otro dispositivo de E/S, utiliza una instrucción especial del set de instrucciones (p. ej., el mnemónico INT #). El microporcesador decodifica el código de operación, en este caso "CD", y ejecuta el servicio identificado por el número "#". Para exemplificar lo enunciado, a continuación se muestra cómo en una computadora personal se produjo un "desensamblado" (*unassembler*) de una instrucción ingresada en modo 16 bits.

La instrucción es INT 08; si se ingresa en la computadora utilizando un "emulador modo 16", éste la ensambla en forma automática pero no la ejecuta; si luego queremos desensamblarla para verla, entonces utilizamos un comando de este modo:

```
-u 0100 <ENTER>
1531:0100 CD08
```

La "u" representa el emulador. El comando "desensamble" 0100 es la referencia a memoria donde se ingresó la instrucción Assembler INT 08. La "ejecución de este comando" (no la instrucción) muestra que la base del segmento de código es "1531". La referencia o desplazamiento respecto de la base es, como ya se indicó, "0100". Luego del espacio viene el código de operación "CD" de un byte, y a continuación "08", que es el identificador de interrupción que se utilizó en el ejemplo y que a los fines de esta explicación no interesa saber qué servicio convoca, o sea que el código de instrucción completo es "CD08". Éste es uno de los

tantos números de interrupción que se puede programar. Para saber qué servicio brinda cada número debemos recurrir a un manual de servicios de interrupción del sistema operativo o al manual de referencias técnicas de las BIOS. Las interrupciones programadas se denominan exactamente **interrupciones internas, interrupciones programadas o interrupciones software**, y causan la suspensión momentánea del programa que las convoca para bifurcar el servicio solicitado; éste se ejecuta y se retoma el programa interrumpido.

Cuando el microprocesador "recibe" una señal de interrupción "desde afuera", deja la ejecución del programa actual y bifurca al servicio residente en la memoria principal. Estos servicios se denominan servicios de interrupción o de dispositivo; estas interrupciones se clasifican como **externas o hardware**.

En el cuadro siguiente se observan las diferencias entre distintos tipos de interrupciones.



Las interrupciones programadas se denominan exactamente interrupciones internas, interrupciones programadas o interrupciones software.

Tabla 8-6. Diferencias entre distintos tipos de interrupciones.

Externas o hardware: son convocadas en forma asincrónica, no dependen del programa en ejecución.	No enmascarables (NMI) La CPU es avisada por una señal de control llamada comúnmente NMI (<i>Non Maskable Interrupt</i>).	Siempre son atendidas. Se consideran de máxima importancia, pues las provoca el hardware ante acontecimientos que no pueden ser por lo menos "avisados" (p. ej., la caída de tensión de la alimentación).
	Enmascarables La CPU es avisada por otra señal que la diferencia de la NMI.	Se consulta la señal de interrupción por cada ciclo de ejecución de instrucción. Como no siempre debe ser atendida, se consulta también una bandera de estado de habilitación de interrupciones. Si la bandera lo autoriza, entonces la CPU suspende de manera momentánea la ejecución de programa activo y ejecuta el servicio de interrupción. Una función hardware externa, determina la prioridad que tiene esta interrupción respecto de otras peticiones pendientes y le da curso (p. ej., recepción de un byte en un puerto). Si la bandera de interrupción está desactivada, la CPU no la tiene en cuenta y continúa la ejecución del programa activo; de ahí deriva el nombre de "mascarable" o "enmascarable".
Internas o software: son convocadas por el programa.		Se pueden producir por la ejecución de una instrucción específica dentro de un programa, para solicitar una interrupción que cumpla con alguna función determinada. La forma de convocarla es INT #, donde # corresponde al número que identifica la interrupción.
Excepciones: son provocadas como consecuencia de anomalías que se producen y detectan durante la ejecución del programa y a causa de ella.	Faltas o errores	Son las que se pueden detectar y corregir antes de que se produzca la ejecución de una instrucción determinada (p. ej., se produce una falla al invocar una página que no está en memoria principal). La atención de esta interrupción provoca que el sistema operativo localice la página faltante y la cargue.
	Trampas	Son las que se detectan una vez ejecutada la instrucción que las provoca (p. ej., las que de alguna manera incorpora el usuario en el programa, como sobreflujo u <i>overflow</i>).
	Abortos	Son las que se detectan sin localizar la instrucción que las provoca, abortando la ejecución del programa (p. ej., un valor no válido en un registro de sistema).

Antes de invocar al servicio de interrupción se deben colocar "determinados" parámetros en "ciertos" registros del microprocesador. Luego se ejecuta el servicio, que en general también devuelve parámetros o estados en otros o en los mismos registros, o en las banderas del registro de estado.

Cuando se produce una interrupción y cuando ésta no provoca la finalización del programa en ejecución, se debe resguardar la información que se aloja en todos los registros del micro-

y que se relaciona con la ejecución del programa interrumpido (p. ej., no se sabrá hasta qué instrucción se ejecutó si no se resguarda el contenido del registro IP o EIP, según el modo de operación del microprocesador). Toda la información de la CPU asociada con la ejecución se almacena en memoria principal, en una estructura de dato denominada pila asociada al programa. Este procedimiento permite **resguardar el entorno de CPU** para luego reanudar la ejecución a partir del momento en que se produjo la interrupción. El programa pasa a un estado de espera que durará como mínimo el tiempo que tarde en ejecutarse el servicio de interrupción.

Cuando el servicio se ejecutó por completo, el programa interrumpido debe continuar su ejecución, pero los registros internos contienen información que no le sirve, es entonces el momento de restaurar el entorno de CPU, "rescatando" la información que tenían los registros internos desde la pila. Este procedimiento se conoce como **restauración de contexto de CPU**.

O sea que si queremos indicar qué ocurre cuando se detecta una interrupción y ésta puede ser atendida, podemos indicar tres pasos:

a) Programa en ejecución.

b) Presentación de la interrupción.

1. Resguardo de contexto de CPU en la pila.

2. Ejecución del servicio de atención de interrupción.

3. Restauración del contexto de CPU.

c) Programa nuevamente en ejecución.

Los servicios que atienden a las diversas interrupciones y excepciones que se pueden producir se alojan en la memoria principal y, obviamente, "comienzan" en lugares de memoria diferentes. Para conocer dónde se aloja un servicio, también se mantiene en memoria una tabla de vectores de interrupción de n entradas. Cada una de éstas se corresponde con el número de interrupción; así, la INT 10, encuentra su vector en la entrada 10 hexadecimal. Hay tantas entradas como servicios definidos; cada vector "contiene" la posición del servicio. El término vector debe asumirse como señalador o puntero. Estos vectores señalan zonas de memoria RAM o ROM. Al conjunto de servicios se lo denomina manejadores de interrupciones (*Interrupt Drivers*). Para encontrar en una computadora hogareña las descripciones de las interrupciones que controlan tanto la ROM-BIOS como el sistema operativo, se debe recurrir al manual de referencia técnica de la BIOS y al manual de referencia técnica del sistema operativo instalado. Por simple curiosidad, daremos el ejemplo de cómo se representaba la tabla de vectores de interrupción en modo 16 bits; como el ejemplo es ilustrativo, está sombreado y su lectura se puede obviar.

Búsqueda de la tabla de vectores de interrupción que se encuentra en las posiciones más bajas de la memoria (primeras 1024 posiciones):

```
C:\>debug
-d0:0
0000:0000 0A 2B 57 1C 65 04 70 00-16 00 1B 10 65 04 70 00 .+W.e.p.....e.p.
0000:0010 65 04 70 00 54 FF 00 F0-4C E1 00 F0 6F EF 00 F0 e.p.T...L....o...
0000:0020 00 00 6F 20 28 00 1B 10-6F EF 00 F0 6F EF 00 F0 ..o (...o....o...
0000:0030 0B 3A E2 17 6F EF 00 F0-9A 00 1B 10 65 04 70 00 .:.o.....e.p.
0000:0040 0B 00 F4 20 4D F8 00 F0-41 F8 00 F0 47 25 5D FD ... M...A...G%].
0000:0050 39 E7 00 F0 5A 05 A5 0A-2D 04 70 00 28 0A 31CC 9...Z...-.p.(.1.
0000:0060 D4 E3 00 F0 2F 00 DF 10-6E FE 00 F0 04 06 31CC ..../.n....1.
0000:0070 1D 00 6F 20 A4 F0 00 F0-22 05 00 00 C2 D7 FF FF ..o ....".....
```

Éste es un vuelco de memoria (dirección *0000:0000 - 0000:007F*) que permite visualizar el contenido de una porción de la tabla de vectores de interrupción. Cada vector usa 4 bytes. Para encontrar la dirección de memoria que corresponde, *Interruption Driver*, deberá seguir los pasos siguientes:

- Multiplicar el número de interrupción por 4.
- Tomar los 4 bytes que se encuentran en esa localidad, que están invertidos, y asumirlos como dos entidades separadas de 2 bytes.
- Convertirlos a segmento: desplazamiento.
- Invertir los bytes de cada palabra, ya que en memoria el almacenamiento se encuentra en orden inverso.

Si, por ejemplo, usted quiere encontrar la dirección de memoria donde se halla el manejador de la interrupción #10 (interrupción de la BIOS para el manejo del video), debe multiplicar *10h* por 4 (que es la cantidad de bytes por vector). Obtendrá como resultado *40h*. Tome los 2 bytes que se encuentran a partir de esa posición *0B00* y *F420* y conviértalos a segmento: desplazamiento, esto es, *F420:0B00*. Por último, invierta los bytes de cada palabra. Pida un *unassembler* desde esa dirección y aparecerá el servicio en cuestión.

Si bien a simple vista no podría entender el contenido de esta porción de memoria (para hacerlo necesitaría descifrar el código de máquina que encierra), se puede asegurar que ésta, por lo menos, es una parte del programa que maneja o se encarga de ejecutar la interrupción #10.

8.9.1 Concepto de pila

La pila es una estructura de dato en memoria de acceso LIFO. Veremos un ejemplo en modo 16 bits, ya que hemos referenciado los registros en el apartado “Número de registros internos”, Caso ejemplo 1.

El registro de segmento SS (*Stack Segment*) o segmento de pila, se accede con criterio LIFO (*Last In First Out*) o último en entrar, primero en salir. El lector se puede imaginar la pila como un conjunto de platos (locaciones de memoria) apilados, donde para sacar el último, primero deberá sacar los que están por encima de él, comenzando desde el que se encuentra “más arriba”. La que se encarga del acceso a la pila es la CPU, ejecutando instrucciones “propias de esta estructura”, por ejemplo, PUSH Y POP. La CPU utiliza la pila para:

- Almacenar la dirección de retorno (IP) y, eventualmente, el CS (segmento de código) cuando se llama a un procedimiento, también conocido como subrutina.
- Almacenar el estado del procesador cuando se produce una interrupción. Los registros obligados que apila son el CS, el IP y el estado de los *flags* (*status register*) y, eventualmente, los registros de cálculo, etcétera.
- Para pasar parámetros entre dos procedimientos.

El acceso a la pila se realiza a través de los registros punteros SP y BP. El SP (*Stack Pointer* o puntero de pila) es el registro que contiene la dirección del próximo elemento de la pila vacío; por este motivo, cualquier acceso a la pila utilizará el contexto del caso 1 especificado, por el par base de segmento: desplazamiento, esto es, por el valor contenido en el par de registros SS:SP.

La carga (escritura) o extracción (lectura) de datos de la pila es un procedimiento software. Estas operaciones se llevan a cabo incrementando o decrementando el registro SP. Si la pila

La pila es una estructura de dato en memoria de acceso LIFO.

La carga (escritura) o extracción (lectura) de datos de la pila es un procedimiento software..

trabaja a nivel palabra (2 bytes), cada vez que se extraiga un dato se autoincrementará ($SP = SP + 2$) y cada vez que se cargue se autodecrementará ($SP = SP - 2$) en 2 unidades.

Las instrucciones que a causa de su ejecución afectan la pila son: CALL, INT, RET e IRET.

Call y Ret son instrucciones que sirven para invocar y dar el retorno a un procedimiento o subrutina, mientras que INT e IRET cumplen la misma función cuando se invoca una subrutina de interrupción.

Las instrucciones que permiten el acceso explícito a la pila son:

- PUSH, que empuja o produce el almacenamiento de la palabra en la pila y luego decremente el SP.
- POP, que extrae o saca una palabra de la pila y "desafecta" las posiciones de memoria incrementando el SP.

En el programa del siguiente ejemplo se podrá observar que las primeras acciones corresponden a "cargar" el contenido de las localidades de memoria 1100 y 1101 con los valores hexadecimales D7 y D8, respectivamente. En realidad, estas instrucciones no inciden en el manejo de la pila, pero servirán para visualizar, más adelante, cómo se transfieren esos contenidos de memoria desde un segmento de datos a uno de pila.

Las tres primeras instrucciones del programa transfieren a los registros AX, BX y CX los valores hexadecimales D5D6, D3D4 y D1D2, respectivamente. A partir de este momento es cuando se comienza a "cargar" información en la estructura.

Las instrucciones que siguen "empujan" a la pila el contenido de la dirección de memoria [1100] —y por arrastre la [1101], dado que son 2 bytes por acceso—, luego se "empuja" el contenido de los registros AX, BX y CX con sucesivos PUSH. Cuando con ulterioridad se analice la traza del programa, se observará que estos pasos se fueron cumplimentando sin lugar a dudas.

Las subsecuentes instrucciones POP extraen de la pila los cuatro valores almacenados en ella y los depositan en los registros AX, BX, CX y DX, en ese orden.

La última instrucción INT 20 convoca al sistema operativo de este microporcesador ejemplo.

Manejo de la pila emulando el modo 16 bits

Con el siguiente comando se convoca a un emulador de modo 16 bits, para una computadora personal es de 32 bits. El lector puede realizar la misma tarea si su computadora es IA-32 compatible y si su sistema operativo es Windows XP, ya que plataformas mayores o Windows Vista no admiten la emulación del modo 16 bits.

```
C:\>debug
```

Los dos comandos "e" "escriben" las localidades 1100 y 1101 de la memoria principal con los valores D7 y D8, respectivamente.

```
-e 1100  
2974:1100 26.D7      se almacena D7 en 1100
```

```
-e 1101  
2974:1101 83.D8      se almacena D8 en 1101
```

El siguiente comando "a" permite la edición y la "compilación" de un corto programa Assembler 80X86.

```

-a
2974:0100 mov ax, D5D6 ;transfiere las constantes hexadecimales
                      D5D6 al registro de cálculo AX
2974:0103 mov bx, D3D4 ;transfiere las constantes hexadecimales
                      D3D4 al registro de cálculo BX
2974:0106 mov cx, D1D2 ;transfiere las constantes hexadecimales
                      D1D2 al registro de cálculo CX
2974:0109 push [1100] ;primera instrucción de carga a la pila,
                      se transfiere el contenido de 1100
2974:010D push ax     ;segunda instrucción de carga a la pila,
                      se transfiere el contenido de AX
2974:010E push bx     ;tercera instrucción de carga a la pila,
                      se transfiere el contenido de BX
2974:010F push cx     ;cuarta instrucción de carga a la pila,
                      se transfiere el contenido de CX
2974:0110 pop ax      ;primera instrucción de
                      extracción de la pila
2974:0111 pop bx      ;segunda instrucción de extracción de la
                      pila
2974:0112 pop cx      ;tercera instrucción de extracción de la
                      pila
2974:0113 pop dx      ;cuarta instrucción de extracción de la
                      pila
2974:0114 int 20      ;fin de programa
2974:0116

```

Los siguientes comandos "t" permiten visualizar la traza de ejecución, instrucción por instrucción, para ver cómo se van actualizando los registros luego de cada ejecución. La primera instrucción MOV ax, D5D6 se ejecuta así:

```

-t
AX=D5D6 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=2974 ES=2974 SS=2974 CS=2974 IP=0103 NV UP EI PL NZ NA PO NC

```

Se actualizó AX e IP.

Próxima Instrucción BBD4D3 MOV BX, D3D4

```

-t
AX=D5D6 BX=D3D4 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000
DI=0000
DS=2974 ES=2974 SS=2974 CS=2974 IP=0106 NV UP EI PL NZ
NA PO NC

```

Se actualizó BX e IP.

Próxima Instrucción B9D2D1 MOV CX, D1D2

```

-t
AX=D5D6 BX=D3D4 CX=D1D2 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=2974 ES=2974 SS=2974 CS=2974 IP=0109 NV UP EI PL NZ NA PO NC

```

Se actualizó CX e IP.

Próxima Instrucción FF360011 PUSH [1100]
 -t
 AX=D5D6 BX=D3D4 CX=D1D2 DX=0000 SP=FFEC BP=0000 SI=0000 DI=0000
 DS=2974 ES=2974 SS=2974 CS=2974 IP=010D NV UP EI PL NZ NA PO NC

Se actualizó el IP, la pila no se ve pues no es un registro, sólo se ve el decremento del puntero de pila SP.

Próxima Instrucción 50 PUSH AX
 -t
 AX=D5D6 BX=D3D4 CX=D1D2 DX=0000 SP=FFE8 BP=0000 SI=0000 DI=0000
 DS=2974 ES=2974 SS=2974 CS=2974 IP=010E NV UP EI PL NZ NA PO NC

Se actualizó el IP, la pila no se ve pues no es un registro, sólo se ve el decremento del puntero de pila SP.

Próxima instrucción 53 PUSH BX
 -t
 AX=D5D6 BX=D3D4 CX=D1D2 DX=0000 SP=FFE8 BP=0000 SI=0000 DI=0000
 DS=2974 ES=2974 SS=2974 CS=2974 IP=010F NV UP EI PL NZ NA PO NC
 2974:010F 51 PUSH CX

Se actualizó el IP, la pila no se ve pues no es un registro, sólo se ve el decremento del puntero de pila SP.

-t
 AX=D5D6 BX=D3D4 CX=D1D2 DX=0000 SP=FFE6 BP=0000 SI=0000 DI=0000
 DS=2974 ES=2974 SS=2974 CS=2974 IP=0110 NV UP EI PL NZ NA PO NC

Próxima instrucción 58 POP AX

Éste es el momento de representar "un mapa parcial" de memoria para comprender la disposición de los datos que se ingresaron en la memoria, teniendo en cuenta que si bien los registros de CPU son de 16 bits, las locaciones de memoria son sólo de 8 bits.

	FFFF	.
La pila crece de localidades altas a localidades bajas	FFED	D8
	FFEC	D7
	FFEB	D5 (SS)
	FFEA	D6 Segmento de pila
	FFE9	D3
	FFE8	D4
	FFE7	D1
SP (puntero de pila)	FFE6	D2
PUSH (SP = SP - 2)	*	Estructura
POP (SP = SP + 2)	*	LIFO
	*	*
	1101	D8 (DS)
	1100	D7 Segmento de datos
	0000	*

Tenga en cuenta que el registro SP se va decrementando antes de que se ingresen los datos en la pila (en total 8 bytes), y luego de extraerlos se incrementa hasta llegar, en este caso, al valor con que inició lo que significa "pila" vacía. Continuemos con la visualización de la traza:

```
Próxima instrucción 58          POP      AX
-t
AX=D1D2 BX=D3D4 CX=D1D2 DX=0000 SP=FFE8 BP=0000 SI=0000 DI=0000
DS=2974 ES=2974 SS=2974 CS=2974 IP=0111 NV UP EI PL NZ NA PO NC
```

Se actualizó el IP, el AX y se ve el incremento del puntero de pila SP .

```
Próxima instrucción 5B          POP      BX
-t
AX=D1D2 BX=D3D4 CX=D1D2 DX=0000 SP=FFEA BP=0000 SI=0000 DI=0000
DS=2974 ES=2974 SS=2974 CS=2974 IP=0112 NV UP EI PL NZ NA PO NC
```

Se actualizó el IP, el BX y se ve el incremento del puntero de pila SP .

```
Próxima instrucción 59          POP      CX
-t
AX=D1D2 BX=D3D4 CX=D5D6 DX=0000 SP=FFEC BP=0000 SI=0000 DI=0000
DS=2974 ES=2974 SS=2974 CS=2974 IP=0113 NV UP EI PL NZ NA PO NC
2974:0113 5A          POP      DX
```

Se actualizó el IP, el CX y se ve el incremento del puntero de pila SP .

```
Próxima instrucción 5A          POP      DX
-t
AX=D1D2 BX=D3D4 CX=D5D6 DX=D8D7 SP=FFEE BP=0000 SI=0000 DI=0000
DS=2974 ES=2974 SS=2974 CS=2974 IP=0114 NV UP EI PL NZ NA PO NC
```

Se actualizó el IP, el DX y se ve el incremento del puntero de pila SP , que volvió al estado de "pila vacía".

Si observa con detenimiento el manejo de la pila, podrá comprender por qué estos registros intercambiaron su valor y, por último, por qué el contenido de la palabra [1100] se halla en el registro DX.

Como conclusión, y debido al criterio de estructuración que utiliza la pila (LIFO), los valores que se ingresan en la pila se extraen en orden inverso respecto del que tenían cuando entraron

8.10 Alimentación

La alimentación de los distintos componentes de una computadora proviene de una fuente. Ésta es un dispositivo que transforma la corriente que nos entrega la red eléctrica para que sea aceptable para los circuitos electrónicos; el proceso de transformación genera de una entrada de 220V, una salida de, por ejemplo, 5V. La corriente suministrada por la red es alterna y, por lo tanto, se requiere un procedimiento de rectificación para convertirla a continua. Asimismo, para "suavizar" más la señal se la filtra utilizando condensadores que retienen la corriente y la liberan con más lentitud; esto provoca un aplanamiento de la señal que la hace menos oscilante.

8.11 Tecnología

8.11.1 CISC

Del inglés, *Complex Instruction Set Computer*, esta denominación se debe a que se pueden ejecutar instrucciones simples MOV AX, BX, por ejemplo, la que involucra una copia entre dos registros “de bx a AX” en este caso, o complejas MOVS dest, src, que copia bytes desde la dirección formada por la base en DS y el desplazamiento tomado del registro SI al lugar de memoria identificado por la base en ES y el desplazamiento en DI, actualizando el valor del los registros índice de a un byte por vez, basados en el tamaño del *string*; (esta última, muy utilizada en los programas de procesamiento de textos). Mientras la primera puede ejecutarse en un par de ciclos de reloj, la segunda puede llegar a varios, según la longitud del dato. Como ya vimos, “los pasos” de una ejecución están representados por una línea de código binario, que se presenta como salida de la unidad de control por cada pulso de reloj. Una instrucción compleja utiliza varias microinstrucciones y, por tanto, una unidad de control para un set de instrucciones CISC utiliza una ROM de microcódigo. Por otro lado, y como ya se verá en el capítulo Memorias, un set de instrucciones CISC admite múltiples modos de obtención del dato y, por lo tanto, un mismo verbo tendrá en el set tantas instrucciones distintas como modos de direccionamiento acepte el verbo. Veamos como ejemplo la instrucción ADD del set CISC, que utilizamos hasta ahora durante todo el texto como set de referencia. A continuación, se muestran varias “versiones” de ADD como resultado de la utilización del emulador en una computadora. Observe los diferentes códigos de instrucción:

Tabla 8-7. Códigos de instrucción.

-U	100 10D		
Dirección	Código de instrucción	Mnemónico	
1531:0103	03060300	ADD	AX,[0003]
1531:0107	01C0	ADD	AX,AX
1531:0109	0304	ADD	AX,[SI]
1531:010B	0302	ADD	AX,[BP+SI]
1531:010D	034203	ADD	AX,[BP+SI+03]

Cuando un set de instrucciones admite tanta variedad de modos de direccionamiento, aumenta el número de instrucciones del set, además de incrementarse su complejidad en la medida en que aumenta la del procedimiento para acceder al dato.

Como vemos en el ejemplo, el mismo verbo sumar genera instrucciones de distinto tamaño o formato y, por lo tanto, la instrucción no se obtendrá de la memoria en la fase *fetch* en el mismo número de ciclos; esto agregado a los diversos tiempos en el acceso al dato, debido a los distintos modos de direccionamiento mencionados, atenta contra el rendimiento “ideal” del procesamiento *pipeline*, que en términos óptimos supone que todas las instrucciones ocupen prácticamente la misma cantidad de ciclos de reloj para lograr mayor eficiencia.

8.11.2 RISC

En el ejemplo presentado para “X”, la sentencia $C := A + B$ requería las instrucciones, ahora descriptas en Assembler: LDA, ADD, STA; esto es, una única sentencia generaba varias instrucciones de máquina para dar una correspondencia 1 a n. También vimos que cada instrucción se ejecutaba con una serie de microoperaciones que se producían en ciclos de reloj diferidos. Por lo tanto, la ejecución de una “sentencia” involucra en CISC varios ciclos de reloj. La idea de concebir un procesador RISC (*Reduced Instruction Set Computer*) es que cada instrucción tenga la mayor cantidad de microoperaciones solapadas posibles, de modo

que la mayoría de ellas se ejecute en un ciclo de reloj. A su vez, en una arquitectura RISC todas las instrucciones tienen el mismo tamaño, lo que facilita el funcionamiento del *pipeline*.

Cada unidad de ejecución está "cableada" (como en el modelo del capítulo Diseño de una computadora digital) para producir un resultado en relación con el código de operación que recibe; por lo tanto, se disminuye el tiempo de ejecución, también beneficiado por no acceder a la ROM de control. En los microprocesadores RISC todas las funciones y el control están "cableados" (*hardwired*), para lograr velocidad y eficiencia máximas.

Hay sólo dos instrucciones para acceder a memoria, Load y Store, que se utilizan para cargar la gran cantidad de registros disponibles en esta arquitectura para almacenar operandos. Las instrucciones que realizan operaciones aritméticas son de referencia a registro y, en general, tienen especificados dos para los operandos y uno para el resultado. Esta característica permite una independencia entre las instrucciones de acceso a memoria y las de cálculo, lo que facilita la concurrencia de ambas actividades y favorece el procesamiento segmentado.

Al tener instrucciones de tres operandos, el resultado no afecta a ninguno de éstos; en consecuencia, de ser necesaria la reutilización del operando en la ejecución de próximas instrucciones (y si el registro no fue actualizado) ya está disponible en el registro sin necesidad de acceder a memoria de nuevo.

Formato típico de una instrucción aritmética RISC:

Código de operación	Registro 1	Registro 2	Registro 3
---------------------	------------	------------	------------

Las instrucciones con formatos fijos permiten que los campos de códigos de operación y de los registros que contienen los operandos estén siempre codificados en las mismas posiciones, permitiendo el acceso a los registros de instrucciones en etapa de ejecución, al mismo tiempo que se decodifica el código de operación actual.

8.11.3 EPIC

La característica más importante de esta arquitectura es que permite agrupar instrucciones para ejecutarlas de manera paralela en forma explícita. EPIC (*Explicitly Parallel Instruction Computing* o computación de instrucciones paralelas explícitas) designa un nuevo tipo de arquitectura diferente al de las computadoras RISC y CISC.

Las arquitecturas enunciadas demuestran su paralelismo mediante código de máquina secuencial, lo que implica un paralelismo sólo a nivel de ejecución. En una arquitectura EPIC se organizan la ejecución de instrucciones de bifurcación o salto condicionado, que son las que cambian el flujo secuencial de ejecución y los "retríeves" de memoria.

Esta tecnología tiene como características fundamentales una gran cantidad de registros y paralelismo explícito en el código de máquina (o sea que, por ejemplo, la dependencia entre instrucciones es encontrada y manejada por el compilador, no por el procesador, por lo que el programa ejecutable ya está "organizado" antes de su ejecución).

Las instrucciones de las distintas ramas de un salto condicionado son marcadas por registros de atributo (*predicate registers*), que, como vimos en el apartado "Número de registros internos", son los 64 PRn. Predicación es un método para manejar saltos condicionales, que en EPIC se denominan "ramificaciones condicionales". La idea principal del método es que el compilador planifique ambos caminos posibles de la ramificación, para que ambas ramas



La característica más importante de esta arquitectura es que permite agrupar instrucciones para ejecutarlas de manera paralela en forma explícita.

se ejecuten en el procesador en forma simultánea. Cuando un compilador encuentre una expresión de salto condicionado en el código fuente, marcará todas las instrucciones que representan cada camino de la ramificación con un identificador único llamado predicado. En cada instrucción hay un campo de predicado. Cuando la CPU encuentre, en tiempo de ejecución, una ramificación predicable, comenzará a ejecutar el código de los dos destinos de la ramificación. Sin embargo, no guardará el resultado mientras no estén definidos los valores de los registros del predicado. Una vez que se evalúe la condición, el procesador guardará un *1* en el registro del predicado, que corresponde a destino verdadero y *0* en los otros. Antes de guardar los resultados, la CPU corrobora cada registro del predicado de cada instrucción. Si el registro contiene un *1*, las instrucciones son válidas, así que la CPU retirará la instrucción y almacenará el resultado. Si el registro contiene un *0*, la instrucción es inválida, así que la CPU descartará el resultado.

Otra característica es el concepto de carga especulativa. La especulación trata de aprovechar el microporcesador cuando se encuentra en períodos de latencia, en ese momento "especula" sobre las instrucciones y los datos que va a necesitar más adelante y los carga. De esta forma, por una parte se hace uso del procesador cuando no es necesario para otra tarea, y por otra, se acelera la velocidad, ya que los datos que se necesitarán ya estarán cargados en el momento en que se precisen. En esta tarea también está involucrado el compilador, que planifica cargas de datos a nivel compilación.

8.12 Resumen

Como las velocidades aumentan, cada vez se hace más difícil ejecutar una instrucción por ciclo de reloj. Una forma de lograrlo es el paralelismo a nivel ejecución o *pipeline*. Se divide la instrucción en etapas, por ejemplo, buscar la instrucción, decodificarla, obtener los operandos, realizar la operación o ejecutar.

Esto se ve con claridad con el ejemplo de un ADD AH, [0200], que en Assembler Intel 80X86 implica buscar la instrucción en memoria, que en código de máquina es 02260000 (donde 0226 significa "sumar al registro AH un operando que se aloja en memoria"). Esta búsqueda la puede realizar una unidad hardware que se encargue de manera específica de la prebúsqueda de instrucciones. Si el microporcesador cuenta con una memoria interna para albergar las instrucciones que se van trayendo, y si, además, lo hace en forma independiente del resto de las operaciones que se llevan a cabo en la CPU, se indica que se va "llenando anticipadamente" una "reserva de instrucciones". En este caso se gana tiempo, pues los tiempos de acceso a memoria pasan inadvertidos, mientras en la CPU se producen otras actividades solapadas. Para dividir la tarea de ejecutar una instrucción en "etapas", los fabricantes de chips pueden dividir las unidades de trabajo del chip en "áreas especializadas", como en una línea de producción. Cuanto más cantidad de etapas mayor probabilidad de solapamiento de actividad y mejor aprovechamiento del ciclo de reloj.

Es de destacar que IBM tiene el crédito de los desarrollos de las tecnologías CISC y RISC. Ambas arquitecturas se basan en los desarrollos pioneros que alcanzaron el mercado desde los ámbitos académicos. En este sentido, los académicos más importantes en la creación de modelos son John von Neumann (computadora de programa almacenado e intercambiable), Maurice Wilkes (microporgramación y diseños RISC) y Seymour Cray (primeras supercomputadoras que emplearon los principios RISC). El inicio en la evolución de la tecnología RISC surge del entorno universitario en 1980, cuando en la Universidad de Berkeley (California) el Dr. David A. Patterson inició un proyecto denominado RISC I. Le siguieron los proyectos RISC II, SOAR (*Smalltalk on a RISC*) y SPUR (*Symbolic Processing on a RISC*). La consecuencia inmediata, además de los aportes a la Ingeniería, como ciencia, y a los fundamentos del diseño de microporcesadores, fue la creación de una máquina capaz de mayores velocidades de ejecución a menores velocidades de reloj y que requería menores esfuerzos de diseño. En la misma época, en la Universidad de

Stanford, el Dr. John Hennessy también inició un proyecto de implementación RISC, denominado MIPS, que permitió demostrar las capacidades de velocidad de la arquitectura RISC. En el ámbito comercial, estos proyectos generaron la creación de MIPS Computer Systems, a cargo de Hennessy, y de SPARC, desarrollado también en colaboración con David Patterson.

8.13 Ejercicios propuestos

- 1- Sabiendo que una CPU genera una dirección de 33 bits que permiten acceder a la memoria principal

¿Cuál es el tamaño potencial de la memoria principal?

Si a la memoria DRAM se le habilitan 4 Giga ¿Cuántos bits son significativos para direccionarla?

2. REGISTRO AX = 001B

REGISTRO SP = 003C

REGISTRO IP = 0103 R

REGISTRO CS = 0040 BX = 001C

REGISTRO DS = 0050

REGISTRO CX = 001D

REGISTRO SS = 0060

REGISTRO SI = 0003

Indique cuál es el valor del registro AH y del registro AL, en relación con el valor de AX arriba mencionado.

Calcule la dirección física de la próxima instrucción que se ha de ejecutar, considerando los valores de los registros que correspondan de los arriba mencionados.

Calcule la dirección física del próximo lugar vacío en la pila, considerando los valores de los registros que correspondan de los arriba mencionados.

Calcule la dirección física del dato si considera la siguiente instrucción y los valores de los registros que correspondan de los arriba mencionados.

MOV AX, [BX+SI+1]

3. Contestar con verdadero o falso y justificar su respuesta.

La excepción de división por 0 detecta un error de cálculo

Las excepciones se atienden siempre que estén habilitadas por un flag del registro de estado.

La tabla de vectores de interrupción en modo real o modo 16 bits puede contener como máximo 256 entradas

- 4- Observe la línea de código presentada a continuación AB55:0103 MOV AH,[0333]

Considere que la siguiente instrucción a la descripta arriba se aloja en la dirección física AB656, luego calcule cual es el tamaño de la instrucción y cuántos de estos bytes le corresponden al código de operación.

Calcule las direcciones efectivas: de la instrucción y del operando, suponiendo que CS = DS.

5- Este ejercicio le permitirá comprender en qué consiste un servicio de atención de interrupción, en este caso se toma el servicio que detecta el error de división por cero, y le permitirá comprender la dinámica de manejo de la pila cuando se trata de resguardar y restaurar el entorno de ejecución para que la "utilice" otro programa. Además le permite familiarizarse con instrucciones Assembler antes de abordar el estudio de formatos de instrucción que se describen en el capítulo Instrucciones.

a. Ingrese en su computador personal en modo 16 bits (en Windows XP la secuencia es EJECUTAR y dentro de la ventana CMD) o con un emulador. Ejecute la aplicación DEBUG

b. Utilícelo el comando D 0000.0000 para realizar el "vuelco" de memoria a partir de ésta dirección, que es la del primer byte del primer mega byte de su memoria principal donde se alojan los vectores de interrupción.

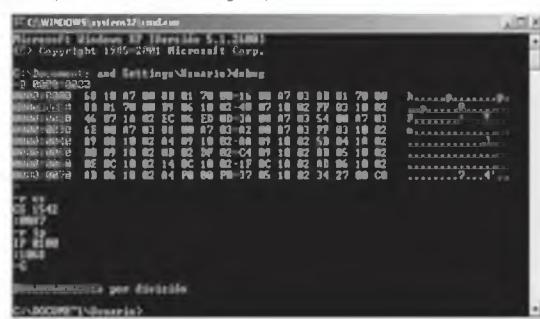
c. El primer vector se referencia en la computadora con el valor 68 10 A7 00 (los cuatro primeros bytes del vuelco corresponden al vector 0). Como esta es la dirección del servicio que avisa que se produjo la excepción división por cero, esta almacenada en convenio little-endian, por lo tanto, lo invierto "de a byte": 00 A7 10 68; los primeros dos bytes se asumen como la base y los segundos dos como el desplazamiento de la dirección segmentada del primer byte de la primera instrucción del servicio alojado en 00A7: 1068

d. Ingrese el comando - R CS para modificar el valor actual del CS y cargue el nuevo valor de base (que en la computadora es 00A7) y que cambie en la suya.

e. Ingrese el comando - R IP para modificar el valor actual del IP y cargue el nuevo valor (que en la computadora es 1068) como desplazamiento.

f. Ahora tiene los dos registros CS e IP que "están apuntando" a la primera instrucción del servicio de interrupción.

g. Ingrese el comando - G para ejecutar el servicio y observe que el programa lo único que hace es emitir el mensaje del error producido. Vea el ejemplo:



Para corroborar como es un servicio de atención de interrupción, se pide que ejecute el comando – U para desensamblar las instrucciones de máquina como se muestra en el recorte de pantalla de abajo y que indique:

¿Cuál es el motivo por el cual es necesario “desensamblar” las instrucciones si ya están almacenadas en memoria en código de máquina?

¿Qué significa el mnemónico NOP y a qué set de instrucciones pertenece? (consulte cualquier manual referenciado como *Developers Manual* de Intel)

¿Qué significa CALL 115B que aparece luego de las dos instrucciones NOP almacenadas a partir de la dirección 00A7: 1068.



Proceda a “desensamblar” el código a partir de esa dirección. Como se vé se hace un push a la pila del registro de banderas y un call a 1161



¿Cuál es el motivo por el cuál se realiza el PUSHF?

¿A que clasificación de instrucciones pertenece ésta instrucción?

¿Cuál es el motivo por el cuál luego del CALL se ejecuta un POPF?

¿Cuál es el motivo por el cuál siempre que hay un PUSH aparece un POP en la secuencia lógica de programación?

Proceda a “desensamblar” el código a partir de la dirección 1161, según observe en el siguiente vuelco de pantalla.



Observe que las primeras instrucciones siguen resguardando el entorno de CPU (PUSH) y las siguientes corresponden a la lógica del programa que despliega el mensaje. Luego de la instrucción JZ, se restauran (en orden inverso los registros POP) hasta el RET.

¿Por qué este programa sólo almacena en la pila cinco registros de 16 bits de todos los presentados en el capítulo?

Si las instrucciones push y pop resguardan el contexto de CPU en la pila ¿En qué instrucción comienza y en cuál termina la lógica que permite que este programa despliegue en pantalla el mensaje de error “desbordamiento por división”?

La instrucción RET es una instrucción de bifurcación a la instrucción siguiente al llamado CALL 1161

¿Cuál es la dirección segmentada de la próxima instrucción que ejecutará la CPU?

¿Cuál es simbólico de máquina que le corresponde?

¿Cuál es el código de máquina que le corresponde y cuánto byte ocupa en memoria?

¿Qué registro de CPU se modificó a causa de su ejecución?

¿Cuál es la próxima instrucción que se ha de ejecutar expresada en Assembler?

Luego de ésta ¿En qué dirección segmentada se encuentra la próxima instrucción que se ha de ejecutar?

¿Es posible visualizarla en el vuelco de pantalla que corresponde?. En caso afirmativo, exprésela en Assembler, en caso negativo, indique “no se muestra en el vuelco”.

8.14 Contenido de la página Web de apoyo

El material marcado con asterisco (*) sólo está disponible para docentes.

Resumen gráfico del capítulo

Autoevaluación

Video explicativo (02:04 minutos aprox.)

Audio explicativo (02:04 minutos aprox.)

Evaluaciones Propuestas*

Presentaciones*

9

Memorias

Contenido

9.1 Introducción	208
9.2 Clasificación de memorias	209
9.3 Dimensión de la memoria	210
9.4 Memorias RAM estáticas y dinámicas	211
9.5 Jerarquía de memorias	215
9.6 Memorias caché	217
9.7 Memoria principal	227
9.8 La memoria como en un espacio lógico	233
9.9 Administración de memorias externas	243
9.10 Resumen.....	254
9.11 Ejercicios propuestos.....	255
9.12 Contenido de la página Web de apoyo.....	256

Objetivos

- Aprender las características de la tecnología de memorias de semiconductores.
- Comprender las ventajas de una organización de memoria jerarquizada.
- Describir e interpretar a nivel bloque las relaciones entre distintos chips de un módulo de memoria.
- Obtener una visión generalizada basada en los distintos tipos de clasificación de las memorias.
- Estudiar en detalle las técnicas de memoria virtual y los mecanismos de traducción de direcciones.



En la página Web de apoyo encontrará un breve comentario de la autora sobre este capítulo.



La memoria de semiconductor almacena bits en celdas binarias constituidas por organizaciones de componentes electrónicos, como los transistores y los condensadores.

9.1 Introducción

Las memorias de las computadoras se utilizan para retener los bits que constituyen el alfabeto digital para representar programas y datos o cualquier otra entidad operable para una computadora. Uno de los parámetros más importantes para medir el rendimiento es la capacidad del procesador para gestionar los accesos a sus memorias.

Imagine que usted lee este capítulo y devuelve el libro a la biblioteca. Es difícil que luego de un tiempo pueda recordar su contenido. Sin embargo, nunca olvidará cómo respirar; parece que esta actividad permanece inalterable, como en una memoria fija durante toda su vida.

Podemos indicar que los seres humanos tenemos básicamente dos tipos de memoria: una transitoria, cuyo contenido puede variar, y otra fija, que se mantiene inalterable atendiendo actividades imprescindibles y no modificables. Con estos ejemplos sencillos se intenta una semejanza con los dos tipos básicos de almacenamiento que se encuentran en una computadora, y que constituyen lo que se denomina memoria principal o interna. Una inalterable y otra modificable. En las memorias de semiconductores el primer caso se corresponde con las tecnologías ROM y el segundo, con las tecnologías RAM.

Las memorias pueden ser volátiles o perennes. Cuando se quieren procesar programas o datos intercambiables se utilizan memorias volátiles, que se pueden leer y escribir, son de tecnología de semiconductores y constituyen el área de trabajo de una computadora. Su nombre en el mercado se ha generalizado como memoria RAM. El término volátil referido a una memoria indica que la información representada en los bits almacenados se pierde cuando la computadora se apaga (o sea que las celdas binarias que almacenan bits dependen del suministro de corriente).

Los programas y los datos fijos, asociados en forma directa a la CPU, se almacenan en memorias no volátiles (o perennes) de semiconductores, que sólo se pueden leer, por lo que su contenido no se altera. Estas memorias no necesitan estar conectadas en forma permanente a la fuente de suministro para retener sus bits y su nombre en el mercado es ROM.

La RAM y la ROM son memorias a las que por lo general se accede al azar (o son de acceso aleatorio o de acceso *random*); o sea que se puede ubicar una unidad de información sin necesidad de acceder previamente a otras. A los efectos de dar un ejemplo más claro, debemos imaginar los bits almacenados en un bloque de disco (aunque estos no son memorias de semiconductores, sino que representan un bit con un campo magnético); en el disco cualquier byte se deberá localizar en una zona magnetizada hasta encontrarlo (por ejemplo, un sector o *cluster*), esto significa que no se accede al azar dentro del bloque sino en forma secuencial. Por lo tanto, el tiempo para localizar ese byte depende de su posición respecto del momento en el que éste comenzó a leer el sector.

En las memorias de acceso aleatorio, la información se ubica con una dirección decodificada por un decodificador asociado a la memoria, que permite la selección de sólo una unidad de información por vez. En este caso no se requiere el acceso previo a otras unidades de información.

Otra manera de memorizar datos es hacerlo en forma externa, por ejemplo, en un disco. Se denominan dispositivos de almacenamiento masivo a las unidades de disco –CD, DVD y otros medios– que almacenan los bits utilizando tecnología magnética u óptica, y que permiten el almacenamiento “masivo” de bits por su bajo costo. El soporte de los bits de este tipo constituye memorias perennes o no volátiles, que en su mayoría se pueden leer y escribir, y reciben el nombre de memorias externas, masivas o secundarias. Si bien es necesario que estas memorias estén asociadas a la CPU para acceder a ellas, el soporte de bits, por ejemplo, un CD, puede guardarse en un cajón sin que se pierda la información grabada en él. Podemos imaginarlas como la copia en papel de este capítulo (memoria secundaria, de masa o externa), que mientras la leemos se almacena de manera temporal en nuestra memoria humana (memoria principal o interna).

El término **unidad de información** se utiliza para identificar un grupo de bytes a los que se accede en forma conjunta según el tipo de soporte. Para las memorias de semiconductores (RAM o ROM) también se utiliza el término **palabra o posición de memoria** y para las memorias sobre medio magnético, por lo general, **bloque**.

Considerando el tiempo que involucra la actividad de "leer memoria", se denomina **tiempo de acceso** al lapso transcurrido entre la orden de lectura y la disponibilidad efectiva de la unidad de información buscada.

El **modo de acceso** es la técnica que permite la búsqueda de la unidad de información en una memoria.

El siguiente diagrama de flujo muestra la relación más común que se establece entre distintos soportes de memoria en una computadora. El ejemplo pretende establecer una jerarquía entre ellos (desde la más rápida hacia la más lenta, o desde la de menor capacidad hasta la de mayor capacidad).

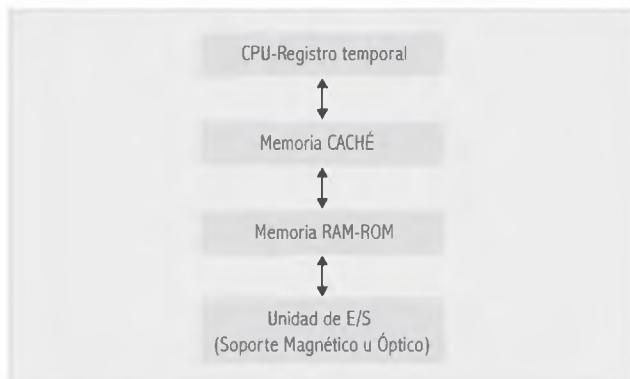


Fig. 9.1. Relación más frecuente que se establece entre distintos soportes de memoria en una computadora.

9.2 Clasificación de memorias

Para abarcar su estudio es necesario clasificarlas según ciertos parámetros que las diferencian:

1. El modo de acceso a la unidad de información.
2. Las operaciones que aceptan por cada acceso.
3. La duración de la información en el soporte.

9.2.1 Clasificación según el modo de acceso a la unidad de información

Una memoria es de **acceso aleatorio** cuando un **componente de selección** habilita una palabra (o posición) e inhabilita a las demás. En cada acceso el componente de selección de palabra (o decodificador de direcciones) recibe "el identificador" (que es único) de la unidad de información implicada. El tiempo de acceso es independiente del lugar físico que ocupa la unidad de información en el soporte.

Una memoria es de **acceso secuencial** cuando para acceder a una unidad de información se establece una posición de referencia, a partir de la cual comienza un rastreo de la unidad de información que consiste en la lectura de todas las unidades que la precedan, hasta lograr



Unidad de información: se utiliza para identificar un grupo de bytes a los que se accede en forma conjunta según el tipo de soporte.



Una memoria es de **acceso aleatorio** cuando un **componente de selección** habilita una palabra (o posición) e inhabilita a las demás.

Una memoria es de **acceso secuencial** cuando para acceder a una unidad de información se establece una posición de referencia, a partir de la cual comienza un rastreo de la unidad de información que consiste en la lectura de todas las unidades que la precedan.



Memoria de acceso asociativo: es cuando la búsqueda de la unidad de información implica la comparación de un grupo de bits de la unidad de información con el contenido de una posición de memoria.

la buscada. En este caso, el tiempo de acceso depende de la distancia entre la posición inicial y la unidad de información.

Una memoria es de **acceso asociativo** cuando la búsqueda de la unidad de información implica la comparación de un grupo de bits de la unidad de información con el contenido de una posición de memoria. Las palabras no se identifican por su dirección y el objetivo de su lectura es verificar si este grupo de bits coincide con el contenido de alguna de ellas. Para lograr esto se envía a la memoria el grupo de bits que actúa de rótulo (*label*), que se compara con todas las unidades de información con el fin de hallar una coincidencia. Ésta se logra si se verifica la igualdad entre los bits del rótulo y los comparados.

9.2.2 Clasificación según las operaciones que aceptan por cada acceso

Una memoria es de **lectura/escritura** cuando admite ambas operaciones (algunos autores las denominan "vivas") y es **sólo de lectura** cuando permite únicamente esta operación (algunos autores las denominan "muertas").

9.2.3 Clasificación según la duración de la información

Las memorias son **volátiles** cuando pierden su información con el corte de suministro de corriente y **perennes, permanentes o no volátiles**, en caso contrario.

La clasificación también comprende una tecnología de memoria RAM denominada dinámica, que aunque sus celdas binarias degradan la información con el tiempo, el proceso de recuperación es gestionado por hardware, y, por lo tanto, la "pérdida momentánea" de información se hace transparente al entorno.

9.3 Dimensión de la memoria

Se denomina **capacidad** de memoria a la cantidad de información que se puede almacenar en ella. La capacidad se puede expresar en bits, bytes o unidades que los agrupen:

BIT

La capacidad medida en bits se utiliza para los registros, que son los soportes de información de menor capacidad. Por ejemplo, se puede decir "un registro de 16 bits" o un "registro de 64 bits".

BYTE

De la misma manera, es aplicable a registros. Se puede decir "un registro de dos bytes" o "un registro de 8 bytes". La capacidad de memorias mayores se expresa en unidades que agrupan bytes.

KILOBYTE (KB o K)

El KB (kilobyte), o simplemente *K*, permite definir una agrupación de 1024 bytes.

$1\ KB = 2^{10}\ bytes = 1024\ bytes$.

Cuando se dice "64 K de memoria", significa que su capacidad de almacenamiento es de 64 por 1024, o sea, 65536 bytes.

MEGABYTE (MB, M o MEGA)

Un *MB* (megabyte), o simplemente *M*, equivale a $1024 K$, esto es, un *K* por *K*, 1024 veces 1024, o 1.048.576 bytes. Alrededor de un millón de bytes de almacenamiento (2^{20}).

GIGABYTE (GB o GIGA)

Se refiere a 1024 mega, o sea, un $K \times K \times K \times K$ (2^{30})

TERABYTE (TB o TERA)

Se refiere a 1024 giga, o sea, un $K \times K \times K \times K \times K$ (2^{40}).

Si bien las unidades descriptas en general se asocian con bytes, se pueden utilizar para grupos de bits o de palabra. Por ejemplo, 1 *Mb/seg* significa "un megabit por segundo".

Otras unidades que se utilizan con menos frecuencia son:

Peta (*P*) = 2^{50} ; *Exa* (*E*) = 2^{60} ; *Zetta* (*Z*) = 2^{70} y *Yotta* (*Y*) = 2^{80}

9.4 Memorias RAM estáticas y dinámicas

En una memoria de semiconductores se dice que cada bit se aloja en una **celda binaria**

9.4.1 Memorias SRAM (Static Random Access Memory)

Las SRAM de semiconductores son memorias vivas, volátiles y estáticas. Cada celda es un elemento biestable diseñado con compuertas, que puede modificarse sólo con un proceso de escritura, si no, permanece inalterable siempre que esté conectada a una fuente de suministro.

9.4.2 Memorias DRAM (Dynamic Random Access Memory)

Las DRAM son memorias vivas, volátiles y dinámicas. El término dinámica hace referencia a la característica ya descripta, en cuanto a que estas memorias degradan su información con el transcurso del tiempo, aun cuando están conectadas a la fuente de suministro.

Cada celda almacena un "1" que se representa con la carga de un condensador. Éste conserva su carga con cierto nivel de deterioro durante un período preestablecido; por lo tanto, antes de que la información se pierda hay que restablecer la carga. Este proceso de regeneración se denomina **ciclo de refresco** (*refresh cycle*) y es un procedimiento físico que se produce a intervalos regulares. El control del acceso y el ciclo de refresco deben estar a cargo del **controlador de memoria**, cuya función es simplificar su complejidad para los dispositivos que la accedan.

Son memorias más lentas que las SRAM, pero tienen mayor capacidad. Considere que una celda binaria de tipo estático está constituida por seis transistores, luego el costo por bit de almacenamiento es mucho menor en las DRAM que en las SRAM, debido a que se gana mayor densidad de almacenamiento de bits en el mismo espacio.

9.4.3 RAM con acceso directo

Para permitir el acceso a la información en forma *random* o al azar una memoria se organiza de manera matricial en filas y columnas ($p \cdot q$). Cada unidad de información o palabra de memoria es una fila y está asociada a un componente de selección; por ejemplo, un decodificador n a 2^n , para una memoria de $2^n = p$ palabras. Cada fila o palabra de memoria o posición tiene igual cantidad de bits (q) para toda la matriz, esto es, q columnas. Las columnas están formadas por los bits de igual peso de todas las filas.



Una celda binaria es un elemento que se puede mantener estable en uno de dos estados, 0 o 1.



Dirección física: número ordinal que le corresponde dentro de la matriz.

Para acceder a cada palabra de memoria, cada una de ellas se identifica con su número de fila dentro de la estructura matricial. El número que identifica la palabra en un acceso *random* o al azar se denomina **dirección física** y representa en realidad al número ordinal que le corresponde dentro de la matriz, comenzando de 0 hasta $p-1$.

Caso ejemplo de RAM estática

Para dar una idea aproximada de su relación con el entorno se presenta un dibujo esquemático que muestra con la figura del rectángulo central un supuesto chip de 1024 palabras de 8 bits cada una (la capacidad de almacenamiento total es de 1 KB); las siglas asociadas representan la función y la cantidad de las líneas que accederían al chip.

VCC		GND
D0		A0
D1		A1
D2		A2
D3		A3
D4	1024 x 8	A4
D5		A5
D6		A6
D7		A7
WE		A8
EN		A9

La línea de alimentación VCC suministra el voltaje de régimen para el suministro de corriente y GND, su conexión a tierra.

Las líneas de datos coinciden con la cantidad de bits que se leen o escriben por cada acceso *random*. El sentido de estas líneas es bidireccional, lo que indica que se puede leer o escribir una palabra de 8 bits ($D_0 \dots D_7$), o sea, "q" bits por vez.

Las líneas de dirección son diez ($A_0 \dots A_9$). La combinación de diez bits permite numerar $2^{10} = 1024$ palabras; estos números son las direcciones 0 a 1023 (0 a $p-1$). El dispositivo que acceda al chip deberá "conocer" la dirección de la palabra que se ha de acceder. Las líneas se conectan a las n entradas del componente de selección, de manera que permitan habilitar una palabra e inhibir el acceso a las demás. La línea WE indica que con un "1" en esta línea se da una orden de escritura (*write enable*), entonces con un "0" se da una orden de lectura.

La línea EN (*enable*) indica con un "1" que este chip se habilitó para su acceso.

Para realizar una lectura sobre el chip se deben secuenciar los siguientes pasos:

1. Habilitar el chip y enviar la dirección de la palabra.
2. Dar orden de lectura.
3. Transferir el contenido de la palabra desde el chip hacia el dispositivo que solicita la información, utilizando las líneas de dato.

Para realizar una escritura sobre el chip se deben secuenciar los siguientes pasos:

1. Habilitar el chip y enviar la dirección de la palabra.
2. Transferir el contenido de la palabra desde el dispositivo que solicita el almacenamiento de la información, utilizando las líneas de dato.
3. Dar orden de escritura.

El esquema siguiente representa la conexión necesaria para observar la relación entre distintos chips que forman una matriz final de $1K \times 8$ (idéntica capacidad de almacenamiento que en la figura anterior, pero organizado con chips de menor capacidad).

Cada palabra se accede con una dirección de 10 bits que se transfieren a través de 10 líneas (las ocho de orden inferior son las líneas de selección de palabra y las dos de orden superior corresponden a las líneas de selección de chip). Un decodificador 2×4 habilita uno de los cuatro chips e inhabilita los restantes (entrada EN –enable–).

La señal WE habilita con 1 la escritura y con 0 la lectura. Esta línea llega a todos los chips pero sólo se accederá al activado por EN.

9.4.3.1 Diagrama de interconexión

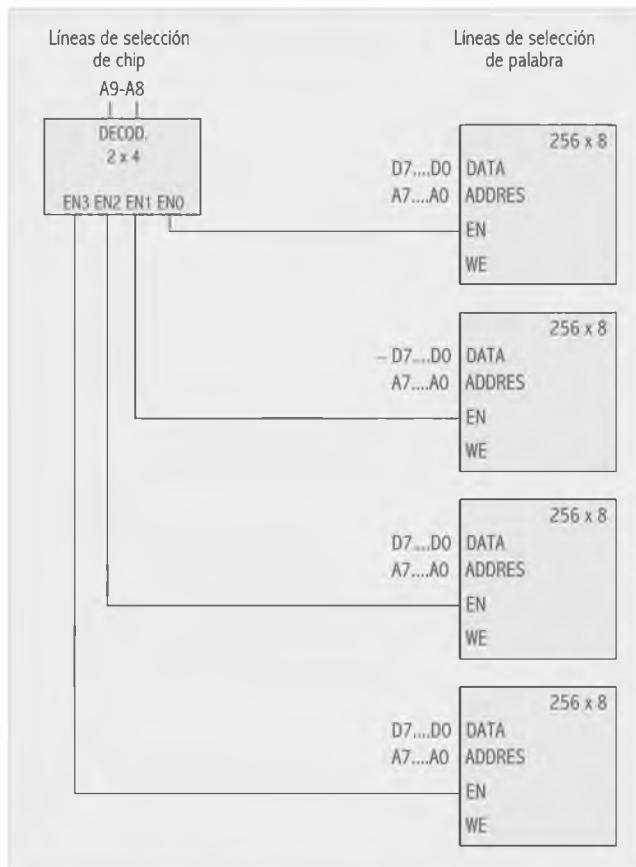


Fig. 9.2. Diagrama de interconexión.

También podríamos pensar en un módulo de memoria RAM de $4096 (2^{12}) \times 8$, que podría estar constituido por 4 chips separados de 1024×8 . En este caso, todos los chips serían seleccionados por las mismas líneas de selección de chip y compartirían las mismas líneas de selección de palabra A9 a A0 y dos para la selección del chip A10 y A11.

Un módulo de memoria es una agrupación de chips distribuidos sobre una placa. Estos chips también suelen llamarse cápsulas o pastillas de memoria.

9.4.3.2 Biestable asociado a una matriz

Una memoria estática está constituida por biestables, en cambio, en las RAM dinámicas los bits se almacenan en condensadores, o sea, un "1" lógico se almacena con el condensador cargado y un "0" lógico, cuando no hay carga. La ventaja principal es que esta celda binaria ocupa menos espacio en un circuito integrado.

El tipo de memoria de semiconductores, que utiliza biestables de varios transistores como celdas de bits ($B_{i,j}$), se denomina SRAM (*static random access memory* o memoria de acceso aleatorio de tipo estático). Como ya se mencionó, cada biestable tiene dos salidas, una para el valor normal del bit almacenado, que llamaremos Q y otra que es su complemento no Q . Este último no se representa en la matriz del ejemplo.

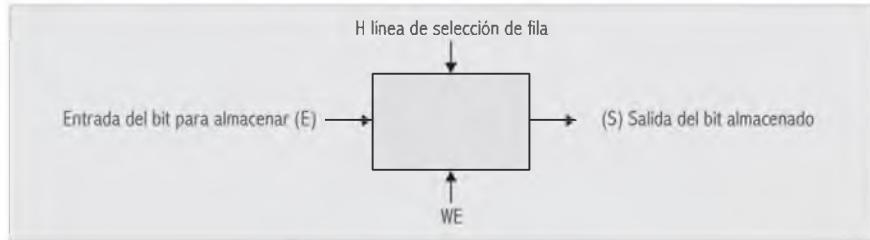


Fig. 9.3. Biestable asociado a una matriz.

Considere que el biestable $B_{i,j}$ es de tipo R-S. Recuerde su tabla de funcionamiento:

Tabla 9-1. Tabla de verdad de biestable R-S.

R	S	Q	Q'
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0

No confunda Q' con no Q . Q' es el estado posterior de Q .

Una celda SRAM puede hallarse en tres estados distintos:

1. Reposo (*stand by*): cuando no se realizan tareas de acceso al circuito.
2. Lectura (*reading*): cuando se solicitó la información.
3. Escritura (*writing*): cuando se actualizan los contenidos.

A continuación, se muestra un diagrama más detallado de la celda. Las compuertas adicionales sirven para controlar el acceso a la celda durante las operaciones de lectura o escritura. Estas relaciones son necesarias para incluirlo dentro de la matriz de la memoria:

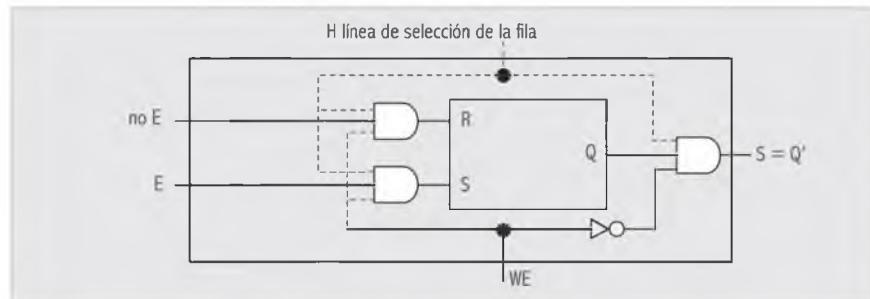


Fig. 9.4. Diagrama más detallado de la celda.

La conexión de selección del biesstable habilita con 1 la compuerta de salida (S) y las compuertas de entrada (que generan R y S) e inhabilita con 0 las tres compuertas.

La conexión WE habilita con 1 la escritura y con 0 la lectura.

Cuando la operación es lectura y se seleccionó el biestable, la conexión S depende del valor de Q .

Cuando la operación es escritura y se selecciona el biestable, si la entrada E es igual a 1 (no E es igual a 0), setea el biestable ($S = 1, R = 0$), y si la conexión E es igual a 0 (no E es igual a 1), resetea el biestable ($S = 0, R = 1$).

Cuando no se selecciona el biestable, $R = 0$ y $S = 0$, luego, Q retiene el bit almacenado en él.

9.4.4 RAM con acceso asociativo

Las memorias asociativas son accesibles por contenido. El contenido buscado se denomina **rótulo**, **descriptor** o **argumento** y puede tener la longitud en bits de la palabra que se ha de acceder.

La posibilidad de asociación requiere que todas las celdas de almacenamiento se relacionen con circuitos que permitan la comparación, razón por la que se vuelven más caras y su uso se justifica en aplicaciones en las que sea imprescindible la búsqueda por contenido.

Cada bit de la matriz (llamémoslo bi,j) se relaciona con un bit del registro de argumento o descriptor (x), por ejemplo, por la siguiente expresión algebraica: $(\overline{x_1 \oplus bi,j}) \cdot x_1 \text{xnor } bi,j$.

Para cada fila de bits de la matriz se logra un nuevo bit único sobre el bit correspondiente del registro de marcas (Z_i), que indica con un 1 la selección de la palabra, si ésta logra el apareamiento buscado.

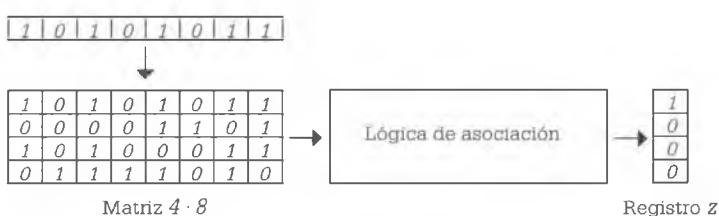


Fig. 9.5. Ejemplo del contenido de una matriz asociativa de 4 x 8.

Como podemos apreciar, para acceder en modo asociativo a una matriz estática sencilla, se le debe agregar toda la circuitería necesaria que permita la representación física de las expresiones algebraicas; así se crea un sistema de memoria más complejo. La función algebraica que corresponde a Z_0 es:

$$z_0 = (x_7 \text{ } xnor \text{ } b_{0,7}) \cdot (x_6 \text{ } xnor \text{ } b_{0,6}) \cdot (x_5 \text{ } xnor \text{ } b_{0,5}) \cdot (x_4 \text{ } xnor \text{ } b_{0,4}) \cdot \\ (x_3 \text{ } xnor \text{ } b_{0,3}) \cdot (x_2 \text{ } xnor \text{ } b_{0,2}) \cdot (x_1 \text{ } xnor \text{ } b_{0,1}) \cdot (x_0 \text{ } xnor \text{ } b_{0,0})$$

Como ejercicio diseñe el diagrama de lógica de la función z_ϑ , y comprenderá la complejidad de la lógica de comparación.

9.5 Jerarquía de memorias

La determinación de una jerarquía de memoria está dada básicamente por tres atributos: velocidad de acceso, costo de la celda (bit) y capacidad de almacenamiento. Un dispositivo de

memoria ideal es aquel que permite gran capacidad de almacenamiento a bajo costo y con el menor tiempo de acceso. Cuando se elige un medio de almacenamiento, se limita al menos una de estas características, las memorias de acceso rápido son de mayor costo y de menor capacidad de almacenamiento. En el caso del diseño de una computadora se toman en cuenta todas estas consideraciones. Si queremos establecer niveles de jerarquía válidos en cuanto a memorias de lectura y escritura (excluimos las memorias ROM y demás dispositivos de lógica programable a nivel físico), en los sistemas actuales podemos indicar los siguientes niveles:

- Registros.
- Memoria caché.
- Memoria DRAM.
- Memoria secundaria, auxiliar o externa.

El primer nivel corresponde a los registros internos, del procesador, denominados también registros de propósito general, y aquellos a los que se accede por instrucciones del sistema operativo de más alto privilegio, utilizados durante el procesamiento de instrucciones de máquina. Estos registros son de alta velocidad y los más importantes, clasificados según su función, son: aquellos que se utilizan como registros de almacenamiento de operandos o registros de cálculo, registros para almacenar información de control y estado del sistema y registros para almacenar temporalmente identificadores de posiciones en la memoria principal, que en su mayoría actúan como punteros de direccionamiento.

En el segundo y el tercer nivel encontramos soportes para el almacenamiento temporal de instrucciones y datos intercambiables, a los que accede el microprocesador en forma directa. Estos soportes se denominan en forma genérica memorias DRAM, cuyos tiempos de respuesta suelen ser mayores que los tiempos de respuesta del procesador. En función de este parámetro se subclasifican en dos tipos, uno perteneciente al segundo nivel de la jerarquía y el otro al tercero:

- La memoria caché, que es una memoria de semiconductores, es más rápida que la DRAM de mayor complejidad y, por lo tanto, de poca capacidad, cuya velocidad de respuesta se adapta a las exigencias del microprocesador.
- La memoria DRAM, que es una memoria de semiconductores lenta, de menor complejidad y, por lo tanto, utilizada para dotar al sistema de mayor capacidad de memoria.

Entonces, podemos suponer como relación válida entre ambas memorias que, mientras que la caché puede ser mucho más rápida, es probable que sea varias veces más pequeña y más cara que la DRAM.

En resumen, la memoria DRAM es de diseño más simple pero más lenta. La memoria caché tiene tecnología SRAM y presenta una mayor cantidad de componentes por celda, siendo mucho más veloz. Esto lleva a una solución intermedia que considera las ventajas de cada una: utilizar mayor capacidad de DRAM y almacenar las zonas más frecuentemente accedidas en la memoria SRAM.

El siguiente y último nivel corresponde a las memorias auxiliares, cuya capacidad de almacenamiento supera de manera increíble la capacidad de la memoria DRAM, pero no puede tener una relación directa con el procesador si no es a través de ella. La demora en el acceso a la información, producida en su mayor parte a causa de los movimientos mecánicos necesarios para el acceso al soporte, hacen que se desaproveche lastimosamente el tiempo de la CPU. Por eso se gestiona la transferencia desde la memoria auxiliar hacia la memoria DRAM, para que


En una memoria de semiconductores se denomina tiempo de acceso al lapso que transcurre desde el momento en que el módulo de memoria recibe una solicitud de datos hasta el instante en que esos datos están disponibles para su transferencia al lugar de destino.


La memoria caché es de alta velocidad y se diseñó especialmente para proporcionar al procesador las instrucciones y los datos utilizados con mayor frecuencia.

el programa pueda ejecutarse desde esta última. El soporte de bits en las memorias auxiliares corresponde a la tecnología de medios magnéticos y ópticos.

Las memorias de disco permiten acceso directo a la información y su capacidad se mide en gigabytes ($2^{10} \cdot 2^{10} \cdot 2^{10}$ bytes), terabytes ($2^{10} \cdot 2^{10} \cdot 2^{10} \cdot 2^{10}$ bytes) y petabytes (2^{50} bytes).

9.6 Memorias caché

Las memorias caché son de tecnología de semiconductor de tipo estático (SRAM = *static RAM*), cuya velocidad de respuesta se ajusta de manera muy favorable a los tiempos del procesador. El fundamento que sustenta su inclusión en un sistema es que su velocidad es compatible con las necesidades de obtención de la información por parte del procesador. El tiempo de acceso de una lectura en memoria DRAM puede ocupar varios ciclos de reloj; esto incluye el proceso de recuperación de la memoria, el pedido, la comprobación y el tiempo de acceso de los datos, típico de esta tecnología. Si el procesador accediese directamente a DRAM, debería contemplar ciclos de espera hasta obtener el contenido de la posición direccionada.

Para balancear costo, volumen de información y tiempo de acceso en la búsqueda de mejorar el rendimiento global, se utilizan ambas. Desde el punto de vista funcional, la caché se utiliza como memoria intermedia entre el procesador y la memoria DRAM y almacena en forma temporal la información a la que se accede con mayor frecuencia en esta última.



En la memoria de etiquetas o *tags* se almacenan las referencias de memoria principal asociadas a cada bloque.

¿Cómo resume usted con tres verbos lo expuesto hasta ahora?

ALMACENAR en mayor medida en la DRAM a bajo costo por bit.

INCORPORAR una pequeña memoria caché de alta velocidad.

GUARDAR copias de DRAM en caché.

En adelante, a la memoria caché la llamaremos simplemente caché y a la memoria de lectura/escritura de la memoria principal, simplemente RAM.



El esquema siguiente muestra una posible relación funcional entre ellos:

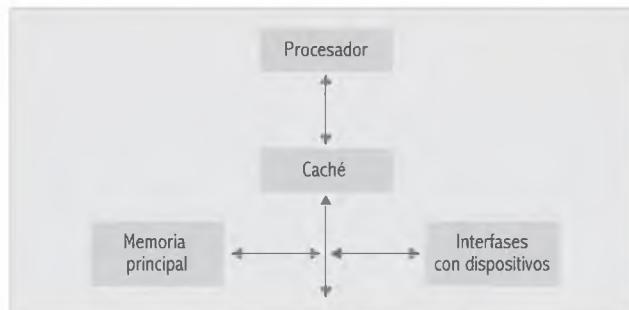


Fig. 9.6. Típica conexión de caché.

El "cerebro" de un sistema de memoria caché se llama controlador de memoria caché. Cuando éste trae una instrucción de la memoria principal, también trae las próximas instrucciones y las almacena en caché. Esto ocurre porque hay una alta probabilidad de que la próxima instrucción también se necesite. Esto incrementa la posibilidad de que la CPU encuentre la instrucción que precise en la memoria más rápida y, por lo tanto, pueda optimizar el tiempo.

Una caché es más que un simple área de almacenamiento, se trata de un subsistema constituido por una memoria de etiquetas, una memoria de datos y un controlador.

El controlador se utiliza para gestionar su actividad. Una de sus funciones es seleccionar cuáles y cuántos bytes de memoria se copiarán en su matriz de datos. Este procedimiento no es arbitrario sino que sigue algún criterio de actualización de datos o actualización de caché, que describiremos luego. Sin importar cuál fuera en este momento, debemos indicar que el controlador "observa" el espacio de almacenamiento de la memoria RAM como un conjunto de bloques, cuyo tamaño es fijo y está determinado por el controlador. Por ejemplo, una RAM de un megabyte de capacidad ($1 M \cdot 8$) puede estar dividida desde el punto de vista del controlador en bloques de 32 bytes, entonces, está dividida en 2^{16} bloques. Para calcular este número se debe dividir el total de la memoria $2^{20} = 1 \text{ mega}$ entre $2^5 = 32$. Entonces, $2^{20}/2^5 = 2^{20-5} = 2^{15} = 32768$ bloques.



La memoria de datos aloja bloques de memoria principal que pueden contener tanto datos como instrucciones.

De todos estos bloques, sólo algunos se copiarán en la memoria de datos.

Por ejemplo, si la caché asociada a esta memoria tiene una memoria de datos de "cuatro ka" = 4 K de capacidad, debe considerarse que ésta es una matriz de $p \cdot q$, donde p es la cantidad de líneas de la memoria de datos y q es la cantidad de bits por línea.

Si hacemos coincidir la cantidad de bytes del bloque de memoria principal con el tamaño de la línea, entonces cada línea es de 32 bytes y, por lo tanto, $q = 256 \text{ bits}$. Si ahora queremos calcular el total de líneas, entonces hay que dividir $2^{12} = 4 \text{ K}$ entre $2^5 = 32$. Entonces, $2^{12}/2^5 = 2^{12-5} = 2^7 = 128$ líneas, que es el valor de p . La matriz está dimensionada como de $128 \cdot 256$.

En este ejemplo no se hizo análisis alguno acerca de la relación de tamaño entre una y otra, sólo se quiere mostrar la relación numérica que indica que sólo 128 de los 32768 estarán contenidos en caché en forma simultánea.

La memoria de etiquetas es una matriz asociativa de $m \cdot n$; cada fila está asociada a una línea de la memoria de datos y se denomina etiqueta, por lo tanto, hay tantas etiquetas como líneas, esto es, 128. El contenido de la etiqueta referencia el número de bloque de memoria principal y que se almacenó en caché; este número es de 15 bits, porque $2^{15} = 32768$. Entonces la matriz está dimensionada como de $m \cdot m = 128 \cdot 15$.

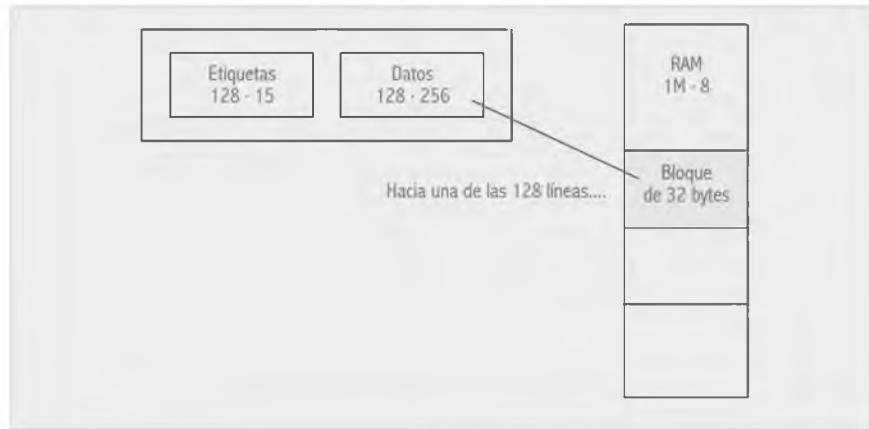


Fig. 9.7. Las tres memorias vistas por el controlador.

Se deduce que la cantidad de líneas de la caché depende de su capacidad expresada en bytes y que esta medida es el tamaño de la memoria de datos.

Por ejemplo, si es de 8 K, entonces $2^3 \cdot 2^{10} = 2^{13}$ entre $2^5 = 2^{13-5} = 2^8 = 256$ líneas de 256 bytes cada una ($256 \cdot 256/8 = 8192 = 8 \text{ K}$). Por ende, ahora la memoria de etiquetas es de $256 \cdot 15$.

Cada etiqueta hace referencia a una línea con los mismos 15 bits que determinan el número de bloque. Duplicar el tamaño de la memoria de datos aumenta la posibilidad de encontrar la información anticipadamente en ella.

Con respecto a la memoria de etiquetas, como su aumento es de acceso asociativo, incorpora mayor complejidad y, por lo tanto, mayor costo.

Realice el ejercicio anterior considerando:

Memoria RAM de 1 M.

Memoria caché de 16 K y bloques de 8 bytes.

Conexión en serie o *look-through*

De ambos tipos de conexiones, la conexión en serie es la más utilizada en los procesadores actuales. Ilustrada en la figura que sigue, podemos afirmar que la ventaja principal se observa cuando se produce un acierto, pues el flujo de información se mantiene en la parte punteada del esquema y deja el bus de sistema en relación con su entorno, aislado y libre para su uso.

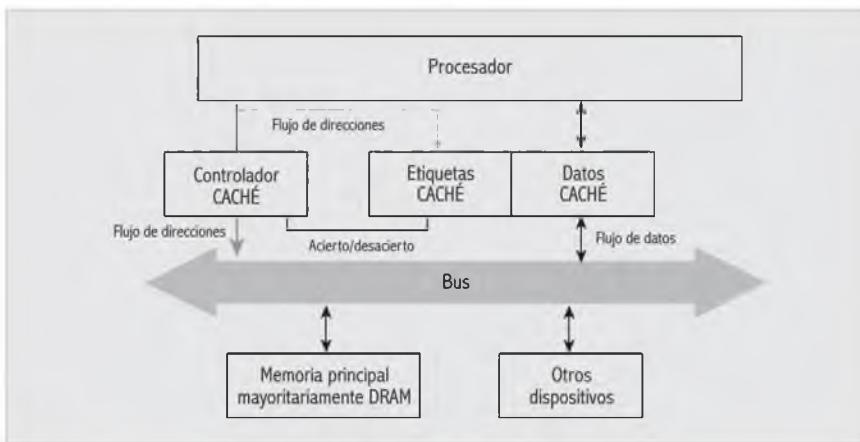


Fig. 9.8. Conexión en serie.

Debido a que el acceso a caché se realiza por un bus separado –esto es, primero al bus que la conecta con el procesador–, si la referencia no se encuentra, recién entonces se accede al bus de dato de memoria. De este modo, mientras el procesador está accediendo a caché, un maestro de bus asociado a un dispositivo de E/S puede acceder en forma simultánea a la RAM y, de esta manera, se optimiza el uso del bus de dato que permite el acceso a memoria principal.

Conexión en paralelo o *look-aside*

Sin embargo, hay sistemas que pueden contemplar un nivel de caché externo. Por esta razón se incluye esta modalidad, al solo efecto de observar las diferencias de una respecto de la otra.

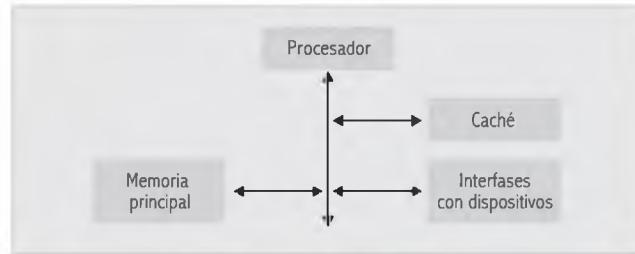


Fig. 9.9. Conexión en paralelo.

En la conexión en paralelo, todas las solicitudes de las posiciones de memoria que requiere el microprocesador llega en forma simultánea a la memoria principal y a la caché. En caso de "acuerdo" (la posición buscada se encuentra en caché), el controlador entrega la posición buscada al microprocesador y genera una señal que aborte el ciclo iniciado para la lectura en la memoria principal; en caso de falla, el ciclo no fue abortado y, por lo tanto, la posición se obtiene desde la memoria principal, y se entrega al microprocesador y a caché (para mantenerla actualizada).

Una ventaja de la conexión paralelo es que permite quitar o agregar el subsistema de caché (si no está integrado en otro hardware), sin necesidad de incluir modificaciones al sistema. Además, si la información buscada no se encuentra en caché, ya se está buscando de manera simultánea en la DRAM, por lo tanto, un desacuerdo (*miss*) no altera el tiempo que de todos modos se iba a tardar en encontrar la información. Como desventaja podemos indicar el alto tráfico al que se somete al bus de sistema ya que, a diferencia de la conexión serie, se utiliza el mismo bus para acceder a ambas memorias.

9.6.1 Principios de funcionamiento

El procesador busca información en memoria principal cuando busca una instrucción para ejecutarla o cuando busca un dato que requiera la ejecución de una instrucción, o sea que la comunicación procesador-RAM es continua. Esta comunicación se establece por medio del bus de direcciones (que transfiere la dirección física o absoluta del primer byte al que se accede) y del bus de datos (que transfiere el contenido de uno o más bytes, según la cantidad de líneas utilizadas del bus). En cualquiera de los dos tipos de conexiones enunciadas, el controlador de caché debe "capturar" la dirección para verificar si puede ofrecer al procesador su contenido. La forma en la que se captura la dirección depende del tipo de organización; como indicamos, la memoria de etiquetas hace referencia a un número de bloque de RAM almacenado con anterioridad en una línea de la memoria de datos. Por lo tanto, para el primer ejemplo presentado (un mega de RAM y bloques de 32 bytes) el controlador considera los 15 bits de orden superior de la dirección física (argumento de comparación) y verifica su existencia en la memoria de etiquetas; si el argumento coincide con una etiqueta, entonces el bloque reside en caché y el byte en cuestión se identifica con los 5 bits de dirección restantes. (Recuérdese que la dirección física para el acceso a 1 M es de 20 bits, puesto que $2^{20} = 2^{10} \cdot 2^{10} = K \cdot K = 1 M$).

El esquema muestra la etiqueta (expresada en bits) que identifica al bloque 127 y la línea correspondiente (expresada en hexadecimal).

Si el byte en cuestión es el identificado por la dirección hexadecimal 00FC0 (denominada dirección física o absoluta), 00FC0 se pasa a binario y resulta 0000 0000 1111 1100 0000, entonces, los 15 bits de orden superior resaltados en negrita muestran la etiqueta (127 decimal) y los 5 bits restantes, el número de byte dentro del bloque (0 decimal), en este caso el primero, cuyo contenido es "15".

En caché

Etiqueta del bloque 127

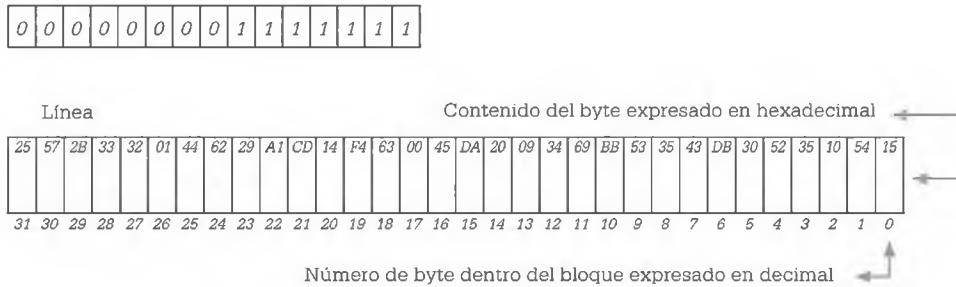


Fig. 9.10. Línea seleccionada.

Tabla 9-2. Contenido del bloque 127 en la RAM.

Nro. de byte en el bloque	Dirección	Contenido
31	00FDF	25
30	00FDE	57
29	00FDD	2B
28	00FDC	33
27	00FDB	32
26	00FDA	01
25	00FD9	44
24	00FD8	62
23	00FD7	29
22	00FD6	A1
21	00FD5	CD
20	00FD4	14
19	00FD3	F4
18	00FD2	63
17	00FD1	00
16	00FD0	45
15	00FCF	DA
14	00FCE	20
13	00FCD	09
12	00FCC	34
11	00FCB	69
10	00FCA	BB
9	00FC9	53
8	00FC8	35
7	00FC7	43
6	00FC6	DB
5	00FC5	30
4	00FC4	52
3	00FC3	35
2	00FC2	10
1	00FC1	54
0	00FC0	15

9.6.2 Caching

El *caching* es un procedimiento que, gestionado por el controlador, anticipa las necesidades de posiciones de memoria principal de acuerdo con cierto cálculo de probabilidad de uso y utilizando criterios que consideran los principios de vecindad espacial y temporal.

Consideré que los programas se ejecutan en pasos secuenciales y que las variables se alojan en zonas adyacentes, o sea que los programas requieren datos e instrucciones que se alojan en posiciones de memoria cercanas a las recientemente accedidas, esto es, que cuanto más vieja sea una referencia a dato, tanto menor será la probabilidad de que se acceda a ella en lo inmediato.

Estas situaciones se demostraron a través de la experiencia, pero para comprenderlas mejor piense en una variable de cálculo incluida en la ejecución de un bucle que se ejecuta n veces. En este caso particular la variable, se referencia varias veces (n) en un tiempo de ejecución breve, o sea, hasta que finaliza el bucle y, por lo tanto, cumple con el criterio de vecindad temporal. La capacidad de la caché varía con los avances de la tecnología; se debe tener en cuenta que su rendimiento depende tanto de la efectividad de la gestión de *caching* como de su tamaño.



La caché se divide en líneas o bloques de tamaño fijo. Cada bloque se asocia a una etiqueta que opera a modo de referencia, utilizando parte de la dirección física de la posición buscada en la RAM; además, para cada línea se puede almacenar información de estado, por ejemplo, "línea actualizada".

9.6.2.1 Traducción de la dirección física (organización interna)

Como el tamaño de la memoria DRAM no coincide con el de la caché, sus respectivos espacios de direccionamiento son distintos, por lo tanto, una dirección física deberá ser traducida o "mapeada" por el controlador, que "adapta" la dirección y comprueba si existe tal referencia en caché. La gestión de traducción puede clasificarse según tres formas de organización del subsistema:

- Totalmente asociativa.
- Asociativa de 1 vía o de correspondencia directa.
- Asociativa de conjunto o de n vías.

9.6.2.2 Mapeo totalmente asociativo

La caché utiliza una memoria de tipo asociativo en los casos descriptos, donde a cada línea le corresponde una etiqueta.

A continuación se observa un esquema de la forma en que la dirección física se analiza para el ejemplo anterior.

19	5 4	0
etiqueta		posición

Se considera la mejor organización, dado que no existe una relación entre el identificador del bloque y su posición dentro de la caché.

Ante un fracaso (no hay equiparamiento, no hay acierto), la palabra se obtiene de la memoria principal siguiendo dos caminos: hacia el procesador y hacia la caché, para agregarse como "bloque al que se accedió recientemente".

El conjunto de la memoria se puede organizar como un anillo; cada vez que se agrega un bloque, tanto las etiquetas como las líneas que ya estaban se desplazan un lugar y si la caché está completa la primera de la cola se pierde (que es precisamente la más antigua).

Por citar una desventaja, debido a la gran cantidad de etiquetas, la lógica de comparación es compleja y cara.

9.6.2.3 Mapeo asociativo de una vía o de correspondencia directa

Si bien ésta es la organización interna menos utilizada en los subsistemas actuales, en este texto se detalla por su simplicidad, para una primera aproximación a la organización más usada, que es la de correspondencia asociativa por conjuntos. La diferencia radica en la forma en que el controlador interpreta la dirección física, y en consecuencia ubica los bloques en caché.

La memoria RAM se divide en grupos del mismo tamaño e igual estructura que la línea de la memoria de datos en caché.

Suponga una memoria principal de 1 megabyte y una caché de 4 K

La caché se puede organizar en 256 líneas o bloques de 16 bytes cada una.

$$256 \text{ líneas} \cdot 16 \text{ bytes} = 4096 \text{ bytes} = 4 \text{ Kbytes}$$

Los 20 bits que permiten identificar cualquiera de las un millón de posiciones serán representados en hexadecimal por las letras XXYYZ, de modo que para esta organización pueden asumir una nueva interpretación:

- El dígito hexadecimal menos significativo indica el número de byte de 0 a $2^4 - 1$, en decimal (0;15) que denominaremos Z.
- Los dos siguientes, el número de *bloque o línea en caché*, que equivale al número sector de RAM desde 0 hasta $2^8 - 1$, en decimal (0;255), que denominaremos YY. Dos grupos pueden tener una identificación de sector idéntica, por ejemplo, 7B448 y 6C447 son distintos grupos de memoria RAM de idéntico sector y no podrían estar en caché en forma simultánea, ya que les correspondería la única línea identificada como 44
- El resto de la dirección, que denominaremos XX, se considera etiqueta, como en la organización anterior.

De este modo, hay una etiqueta por número **sector RAM o línea caché**, o sea, 256 etiquetas.

Ya que la futura ubicación del bloque está predeterminada por el valor YY, para acceder a la caché se consideran YYZ como su dirección de acceso, en este ejemplo 12 bits ($2^{12} = 4 \text{ K}$), que oscila entre 000H y FFFH, en decimal (0;4095) (acceso random a la memoria de datos). La información buscada se encuentra en caché, si para YYZ la etiqueta asociada coincide con XX.

En esta organización se puede prescindir de una memoria de carácter asociativo para la memoria de etiquetas, ya que sólo se deben comparar los bits XX de la nueva dirección con la única etiqueta asociada a la línea. Esto reduce la complejidad y el costo, además de hacer más rápido el acceso. No requiere algoritmo de sustitución, puesto que se conoce el emplazamiento de cada sector en caché (línea YY), es por esto que indicamos que dos bloques o grupos de RAM con idénticos YY no pueden estar en caché en forma simultánea.

Inténtelo con una RAM de 1 K y una caché de 8 líneas o bloques de 8 bytes cada una, esto es, de 64 bytes de memoria de datos

$$1 \text{ K} = 2^{10} \text{ dividido en } 8 \text{ líneas } (8 = 2^3) = \frac{2^{10}}{2^3} = 2^{10-3} = 2^7 = 128 \text{ grupos.}$$

La etiqueta sólo necesita los 4 bits de orden superior, pues los 6 bits restantes se utilizan de la siguiente manera: 3 para identificar el número de la línea y 3 para identificar el byte dentro del bloque.

9	6 5	3 2	0
etiqueta	línea		posición

Ejercicio 1:

Un procesador asociado a una caché de 8 líneas con 8 bytes cada una y a una RAM de 1 K realiza dos accesos de lectura sobre las direcciones 1F2 y 3CD (considere sólo los 10 bits de orden inferior cuando pase las cadenas hexadecimales a binario, puesto que en el pasaje directo se generan 12). ¿Qué bloque o bloques de RAM resultarán sustituidos? Responda con el número hexadecimal que identifica al bloque o los bloques, o NINGUNO si no existe sustitución.

	<i>Etiquetas</i>	7	6	5	4	3	2	1	0
0	1001	A4	54	79	32	45	22	F0	56
1	0111	14	52	33	8D	B5	34	45	32
2	1001	00	FF	64	31	11	A6	33	24
3	0011	32	63	CC	C3	FA	1F	33	53
4	1010	76	88	64	46	25	37	F3	FA
5	0100	DC	14	33	96	8A	7B	34	F0
6	0010	15	37	A1	85	AA	B6	42	13
7	1001	77	76	34	90	00	15	61	24

Primero se convierte la dirección hexadecimal a binario y se la mapea así:

9 8 7 6	5 4 3	2 1 0
Etiqueta	Línea	Byte

$$1F2 = 0001\ 1111\ 0010 = 01\ 1111\ 0010 = 0111\ 110\ 010$$

Por lo tanto, se busca en la línea 110, o sea, 6, la etiqueta 0111. Como no hay coincidencia, se buscará en RAM y se sustituirá el bloque previamente alojado en caché en esa línea, cuya etiqueta es 0010.

Como $1K = 2^{10}$, se divide entre 2^3 , entonces $2^{10-3} = 2^7 = 128$ bloques. El identificador del bloque está constituido por los bits de la etiqueta y la línea 0010, y 110 se constituye en un número de 7 bits = 0010110 = 16 hexadecimal o 22 en decimal.

Pruebe usted con la otra dirección.

Ejercicio 2:

Indique en hexadecimal la dirección que permite acceder con éxito a la línea identificada con el n° "3" y refiérase al byte cuyo contenido es "32".

Respuesta: 0011011111 = ODF

9.6.2.4 Mapeo asociativo de n vías o de n conjuntos

Es similar al mapeo directo, pero cada línea admite n etiquetas y n matrices de datos, esto implica una caché n veces más grande y menor posibilidad de fracaso. Se indica que las n etiquetas y las n matrices de datos de una misma línea constituyen un conjunto.

Si exemplificamos con $n = 2$, entonces el apareamiento se produce primero a la línea y luego a la etiqueta.

Tabla 9-3. Caché de dos vías o conjuntos.

Vía o conjunto 0										Vía o conjunto 1									
Etiquetas		7	6	5	4	3	2	1	0	Etiquetas		7	6	5	4	3	2	1	0
0	1001	A4	54	79	32	45	22	F0	56	0	1011	BB	45	95	69	56	11	81	63
1	0111	14	52	33	8D	B5	34	45	32	1	0100	35	54	A7	B3	B5	00	31	66
2	1001	00	FF	64	31	11	A6	33	24	2	0000	00	00	00	00	00	00	00	00
3	0011	32	63	CC	C3	FA	1F	33	53	3	0010	24	35	CD	B2	F1	31	23	32
4	1010	76	88	64	46	25	37	F3	FA	4	0110	36	99	75	64	31	33	DD	00
5	0100	DC	14	33	96	8A	7B	34	F0	5	1111	00	00	00	00	00	00	00	00
6	0010	15	37	A1	85	AA	B6	42	13	6	1010	25	36	FF	45	44	32	12	99
7	1001	77	76	34	90	00	15	61	24	7	1101	88	84	01	50	34	64	BB	C1

Para facilitar su comprensión, en la primera vía, o vía 0, se utilizó la misma matriz que en el ejemplo anterior. Como se puede observar, la segunda vía o vía 1 tiene idéntica estructura. En esta caché cada línea está asociada a dos etiquetas, por ejemplo, la línea 0 contiene dos bloques de RAM, o sea que se almacena el doble de información, a saber:

0	1001	A4	54	79	32	45	22	F0	56	1011	BB	45	95	69	56	11	81	63
---	------	----	----	----	----	----	----	----	----	------	----	----	----	----	----	----	----	----

La dirección del primero se armaría así: 1001 000 xxx; donde el 1001 es la etiqueta, el 000 es el identificador de la línea y xxx, el número de byte o posición en el bloque.

La dirección del segundo se armaría así: 1011 000 xxx; donde el 1011 es la etiqueta, el 000 es el identificador de la línea y xxx, el número de byte o posición en el bloque.

Entonces, ambos pertenecen al mismo sector RAM 000. Si queremos calcular cuál es el identificador de grupo o bloque en RAM, debemos pensar que si el KB se dividió en grupos de 8 bytes y en el ejercicio anterior ya calculamos que entonces hay 128 grupos, su identificador lo constituyen los 7 bits de orden superior de la dirección física total de 10 bits (esto es, sin los xxx), por lo tanto, el primer bloque es el 1001000 binario o 48 en hexadecimal, y el segundo es el 1011000 en binario o 58 en hexadecimal.



La asignación de una política de reemplazo afecta el rendimiento del subsistema de la misma forma que lo afecta su organización.

9.6.3 Actualización de caché

9.6.3.1 Políticas de sustitución

El momento de actualizar una caché es cuando se detecta una falla o ausencia de la palabra buscada. Esto es, una palabra no presente está en la memoria principal y desde allí el controlador debe "interceptarla" en su camino al microprocesador vía el bus de datos; una vez escrita la memoria de datos y la de etiquetas quedan actualizadas con un acceso reciente, según el criterio ya explicado. Si aún no se completó la caché, la escritura se realiza sin ningún tipo de sustitución; no obstante, por lo general ya se encuentran almacenados datos válidos con la consecuente necesidad de que el controlador asuma un criterio coherente para efectivizar el reemplazo.

En el caso de las organizaciones sectorizadas no hay otro criterio que el de reemplazar la única posición posible dentro de la caché (recuerde que por cada sector o línea se establece la relación con una sola etiqueta). Sin embargo, en las cachés con más de un nivel de asociatividad se utiliza una política de reemplazo o política de sustitución.

La asignación de una política de reemplazo afecta el rendimiento del subsistema de la misma forma que lo afecta su organización. El controlador ejecuta un algoritmo fijo y predeterminado, que se puede clasificar en por lo menos tres categorías:

1. *Least Recently Used* (LRU), la de uso menos reciente.
2. *First In First Out* (FIFO), la primera en entrar es la primera en salir.
3. *Random* (RND), aleatorio.

LRU

Sustituye la información que hace más tiempo que fue referenciada (*Least Recently Used* significa la de uso menos reciente). Entonces, se debe considerar que parte de la información almacenada asociada a cada línea se utiliza para mantener atributos; uno de esos atributos es un grupo de bits que representa esta característica y que el controlador utiliza para identificar la que lleve más tiempo sin que se haya accedido a ella, que no es necesariamente la más antigua.

FIFO

En este caso, las posiciones pueden desplazarse en el sentido de una cola, de modo que la primera que ingresó es la que primero sale, lo que es fácil de aplicar en una caché de tipo asociativo.

RANDOM

En este caso el reemplazo se realiza al azar sobre cualquier línea. Como ventaja podemos interpretar que la aplicación del algoritmo es sencilla y, por lo tanto, rápida, pero su gestión puede caer en la contradicción de sustituir una, a la que se accedió en forma reciente.

9.6.4 Actualización de la memoria principal

Una modificación en cache implica la actualización de memoria; cuando un sistema cuenta con un subsistema caché, las modificaciones a la información que gestiona el procesador se realizan en primer lugar sobre la caché. Esto es así porque se supone que es muy probable que se vuelva a acceder a aquello a lo que se accedió recientemente, según el criterio de vecindad temporal. Tome como ejemplo ilustrativo el caso de una variable incluida en varios términos de un cálculo complejo o incluida en un bucle.

Obviamente, debemos pensar que dos copias de la misma información en soportes distintos deben tener el mismo contenido. Si utilizamos este criterio, entonces por cada actualización de la caché se actualiza la memoria principal. Sin embargo, en cierto sentido este procedimiento no es muy eficiente; la escritura en caché es rápida pero la que se realiza en la memoria es lenta. Por esta razón, también existen políticas de actualización de la memoria principal que se pueden clasificar en, por lo menos, las categorías siguientes:

- Escritura inmediata (*write through*).
- Escritura obligada (*write back*).

Escritura inmediata

Este método es simple y consiste en actualizar de manera simultánea ambas memorias, de modo tal que no se genere incongruencia entre la información almacenada en ambos niveles. Como ya explicamos, esto reduce el rendimiento debido al tiempo empleado en la escritura sobre la memoria principal, con la consecuencia adicional de mantener el bus de dato ocupado por más tiempo. Sin embargo, son los subsistemas más económicos.

Escritura obligada

Este método admite sólo las actualizaciones estrictamente necesarias en memoria principal, o sea que cierta información alojada en caché puede actualizarse varias veces antes de ser efectivamente actualizada en memoria principal. Esto es razonable de acuerdo con el criterio de vecindad temporal que indica que, por ejemplo, una variable de cálculo puede sufrir varias actualizaciones durante el procesamiento sin que se tomen en cuenta sus resultados parciales; ése es el caso de una sumatoria de $1 \text{ a } n$ de una misma variable. Sin embargo, el método genera incongruencia entre la información almacenada en ambos soportes. Esto puede ser grave si no se actualiza la memoria principal cuando se permite que cualquier otro dispositivo tenga acceso a ella sin intervención directa por parte del microprocesador (que sólo se encarga de otorgar el permiso) o cuando el sistema es multiprocesador (y además varios procesadores comparten la misma memoria principal y alguno de ellos quiere acceder a esa zona de la memoria). Para evitar un acceso a información obsoleta, antes de que el microprocesador (actual maestro o

"propietario" del bus) conceda el acceso a otro dispositivo, se habilita al controlador de caché para que copie las líneas marcadas como modificadas en la memoria principal.

Otra situación de "escritura obligada" se observa en el caso de un reemplazo. Para reconocer que se actualizó cierta información, el controlador recurre al atributo que podemos identificar como "modificado/no modificado"; en el primer caso se actualiza la memoria principal y luego se realiza la sustitución, y en el segundo se efectúa el reemplazo directamente.

9.6.5 Niveles de caché

En las computadoras actuales es común la utilización de varios niveles de caché. La denominada caché de nivel 1 es de tamaño reducido y se ubica funcionalmente, interceptando los datos que entran en el microprocesador desde la memoria o los que salen ya procesados hacia la memoria.

La caché de nivel 2 es de tamaño mayor y se ubica (si se incluyó en el sistema) entre la caché de primer nivel y la memoria principal. Su carácter optativo nos obliga a considerar cuáles son los beneficios de su incorporación. En primer lugar, podemos indicar que atiende pedidos menos frecuentes que la primaria, ya que normalmente el éxito en el acceso se logrará accediendo al primer nivel; esto significa que se utiliza como ampliación de la anterior. Dada su mayor capacidad de almacenamiento, no sólo contendrá la misma información almacenada en la primaria sino unos cuantos Kbytes más.

Las ventajas de contar con varios niveles se empiezan a notar cuando las aplicaciones son muy grandes o requieren acceso frecuente a datos en la memoria principal. En la medida en que la eficiencia de un nivel secundario, esto es, en la medida en que se acceda a él con mayor frecuencia por no haber encontrado la información en la primaria, se reducirá el número de ciclos de bus utilizados en el acceso a RAM, que resulta en un mejor rendimiento global del sistema. Por otra parte, si la computadora es un servidor en una red de computadoras, el acceso a su memoria principal es elevado, así como el acceso a sus dispositivos de E/S; esto produce un alto tráfico en el bus, que se beneficia con la implementación de estos niveles de memoria.

En el caso de sistemas multiprocesadores su utilización se justifica en gran medida, ya que en estos sistemas muchos comparten la misma memoria, por lo tanto, el acceso frecuente al bus puede llegar a condiciones de saturación.

9.7 Memoria principal

El esquema de memoria presentado en el capítulo Diseño de una computadora digital, a la que se denominó "X", tiene una organización física como un espacio lineal de direcciones. En "X" la instrucción hace referencia a un dato en la memoria con el contenido del campo DATA, de 12 bits, que permite "direccional" posiciones en el rango $(0; 2^{12} - 1) = (0; 4095)$, congruente con la cantidad de posiciones de memoria, o sea, 4096. En "X" no se incluyó un subsistema caché, por lo tanto, no se realiza ningún procedimiento de mapeo de la dirección física.

Esta dirección se transfiere a la memoria vía el bus de direcciones (*address bus*), también de 12 bits, y se denomina dirección física, dado que permite el acceso a la posición física sin mapeo previo.

Por ejemplo, un procesador, asociado a un bus de direcciones de 16 bits, $(0; 2^{16} - 1) = (0; 65535)$, sólo puede disponer de una memoria organizada de manera lineal en 64 K posiciones (si en cada posición se almacenan 8 bits, entonces se indica que puede direccionar 64 Kbytes).

9.7.1 Memoria a nivel lógica digital

Aquí vemos un esquema posible para la organización en chips RAM. En él, cada una de las líneas del bus de direcciones, se identifica con las siglas A_i . Como la capacidad de almacenamiento está distribuida en cuatro chips, las dos líneas de orden superior A_{15} y A_{14} son las que identifican el número de chip: "RAM 1" con 00, "RAM 2" con 01, "RAM 3" con 10 y "RAM 4" con 11. Estas líneas ingresan en un decodificador n a 2^n , que habilita la entrada EN , la selección de chip (*chip select*) que corresponda. Las líneas A_{13} hasta A_0 , 14 líneas en total, seleccionan una posición de 2^{14} posibles y como 2^{14} es igual a 16384, ésta es la cantidad de direcciones en cada chip. Por cada posición se almacena un byte, que en el caso de una lectura será transferido por las líneas $D_7 \dots D_0$, en total 8 líneas. Si a cada dirección se la asocia a un byte, entonces la capacidad de almacenamiento será de 16384 bytes o 16 Kbytes. Como en total hay cuatro chips, entonces la memoria RAM es de 64 Kbytes. Note que 64 K es igual a 2^16 , como potencias de igual base se suman y se requieren 16 líneas para direccionar toda la memoria, que es precisamente la cantidad de líneas del bus A_{15} - A_0 .

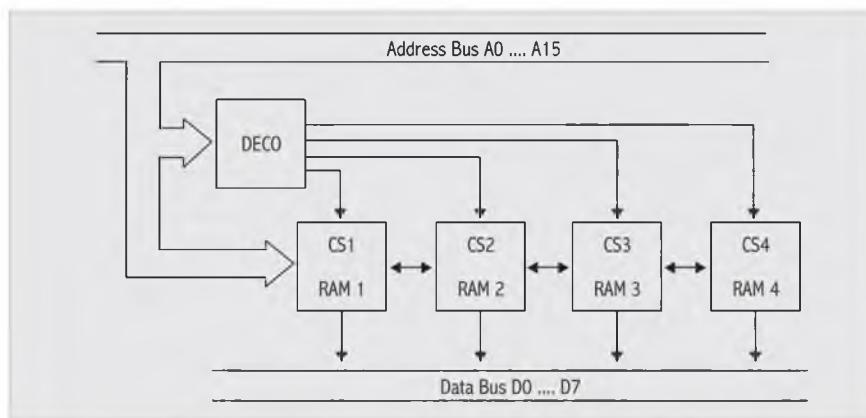


Fig. 9.10. Esquema de la distribución de 4 chips de memoria.

Si un procesador está asociado a un bus de direcciones de 32 bits, puede acceder potencialmente a 4 giga posiciones ($0; 2^{32} - 1$). Esto significa que la cantidad de bits de la dirección determina el espacio de direccionamiento máximo al que se puede acceder en forma directa.

Se debe considerar que no siempre es conveniente que en capacidades de almacenamiento muy grandes se utilice el modelo lineal de direcciones presentado para "X". Ésta será una de las tantas formas en las que se pueda acceder a memoria, pero no la única.

La administración del espacio en memoria es responsabilidad del sistema operativo, por ello, más adelante se explican los modelos lógicos para acceder a ella del modo que sea más eficiente.

Considere la memoria principal o interna como el área de trabajo del procesador, donde residen los elementos de un programa en estado de ejecución (el código del programa, los datos para procesar, los resultados parciales o finales, la pila, su bloque de control, etc.).

En relación con la velocidad de respuesta de las memorias esperadas por el procesador, el número de *GHz* es un parámetro que se ha de considerar en relación con la velocidad del procesador, en la medida en que cuando aumenta también se incrementa la necesidad de memorias más veloces asociadas con él.

La memoria principal está constituida por diversas tecnologías de semiconductores y sus fabricantes utilizan la norma JEDEC, que establece un estándar para los fabricantes de semiconductores.

Una de las necesidades más importantes del procesamiento automático de información es aumentar de manera constante la capacidad de memoria. Si bien la tecnología permite cada año incluir en cada chip una mayor cantidad de celdas de bit, los requerimientos de memoria del software (cada vez más sofisticado) también aumentan.



La JEDEC (Asociación de Tecnología de Estado Sólido) se encarga de la estandarización de la ingeniería de semiconductores.

9.7.2 Memorias RAM dinámicas

Las memorias dinámicas (DRAM) son de lectura y escritura y se utilizan para almacenar una mayor cantidad de bytes en la memoria principal de las computadoras. Esto se debe, sobre todo, a su bajo costo y a su gran capacidad de almacenamiento en relación con las memorias estáticas (cada celda binaria permite el almacenamiento de un bit con una menor cantidad de elementos). Como desventaja podemos señalar que son más lentas que las de tecnología estática.

En un chip de memoria el soporte del bit será un par condensador/transistor, que se replica tantas veces como bits se puedan almacenar en el chip. Como conclusión podemos expresar que la decisión de incluir memorias dinámicas en el sistema se debe a que utilizan menos componentes, lo que posibilita el almacenamiento de mayor cantidad de bits en un mismo espacio. Puesto que cada vez se necesita incorporar una capacidad de almacenamiento superior, las memorias dinámicas permiten aumentar la capacidad sin incrementar su costo global.

La velocidad de respuesta de una memoria depende de la tecnología aplicada en la fabricación del chip y de cómo se resuelva el control del tráfico entre el procesador y la memoria, a cargo del controlador de memoria.

Las memorias dinámicas pueden considerarse menos apropiadas para responder con eficiencia a las exigencias de un procesador, si sólo se considera como parámetro el tiempo de respuesta; por esta razón, la mayoría de los sistemas incluye uno o varios niveles de memoria caché, ubicados entre la CPU y la DRAM.

Para facilitar su comprensión, piense en una celda estática como en un recipiente de vidrio, que nunca pierde su contenido de líquido, relacionelo con una celda dinámica pensando que esta última es un filtro de papel con forma de recipiente. De este modo, podemos deducir que para regenerar el contenido de la celda habrá que verter una cuota de "líquido" a intervalos regulares de modo que se mantenga siempre un mismo nivel.

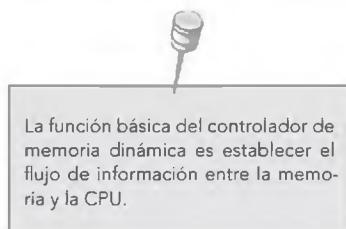
Este procedimiento de "llenado" (por supuesto, ahora olvide los términos líquido y recipiente) es físico y se denomina refresco (*refresh*).

El refresco de una memoria dinámica consiste en recargar los condensadores que tienen almacenado un 1 para evitar que la información se pierda a causa de las fugas en los condensadores.

9.7.2.1 Controlador de memoria dinámica

En este tipo de memoria se "multiplexan" las direcciones, de manera que para acceder a una celda se considera una parte de la dirección física (la de orden superior) para acceder a la fila y la otra parte para acceder a la columna en la matriz.

Como ambas direcciones se presentan al chip de memoria sobre el mismo bus de direcciones, se deben activar dos señales, una que habilite la selección de la fila y la otra que habilite la selección de la columna; estas dos señales se denominan genéricamente *RAS* y *CAS*.



RAS (*Row Access Strobe*) es la señal que habilita la selección de una fila y CAS (*Column Access Strobe*) es la que habilita la selección de una columna.

Debido a ciertas condiciones para considerar en el acceso al chip de las memorias dinámicas, se administran por un dispositivo hardware llamado "controlador de memoria dinámica", cuya función es esconder al entorno, por ejemplo, al procesador, tanto los detalles del acceso como los del mantenimiento de los bits en los chips. El controlador de memoria genera todas las señales de control (p. ej., RAS y CAS) y las de tiempo que se necesitan para la actividad en memoria; además, posibilita su inicialización o período de arranque en el que se realizan escrituras sobre todas las celdas, de modo de cargar cada una de ellas a su capacidad máxima.

Uno de los parámetros que se ha de evaluar es la latencia CAS (*CAS Latency*), que indica, expresado de un modo simple, el número de ciclos de reloj que transcurren desde que se realiza la demanda de datos hasta que éstos se ponen a disposición del bus de datos.

Para el acceso a la celda DRAM, el controlador debe activar primero la señal RAS. Esta señal habilita la decodificación de la dirección asumida entonces como dirección de fila. Luego, activará la señal CAS habilitando la decodificación de los nuevos valores sobre el bus para que se los considere dirección de columna. El tiempo que permanezca activa la señal CAS será el utilizado como parámetro para indicar el tiempo de acceso (*CAS Latency*).

Acorde con las primeras tecnologías en alguna memoria dinámica, cuando se producen dos lecturas consecutivas sobre el mismo chip el tiempo de acceso se penaliza con una cuota de tiempo adicional. Esto se debe a que en cada lectura se descargan ligeramente las celdas de la posición accedida, por lo que es necesario esperar para que se vuelvan a recargar. Este período se denomina "tiempo de precarga".

9.7.2.2 Módulos

Se trata de la forma en que se juntan los chips de memoria, del tipo que sean, para conectarse a la placa base de una computadora. Son pequeñas placas en las que se sueldan los chips de RAM (del tipo que sean), con patillas o pines en un extremo; al conjunto se lo llama módulo. La diferencia de los DIMM frente a otros módulos más antiguos, como los SIMM, es que son más largos (unos 13 cm frente a 10,5) y tienen más contactos eléctricos, además de dos ranuras para facilitar su colocación correcta.

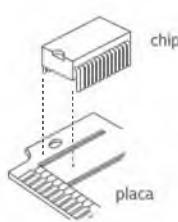


Fig. 9.11. Inserción de un chip.

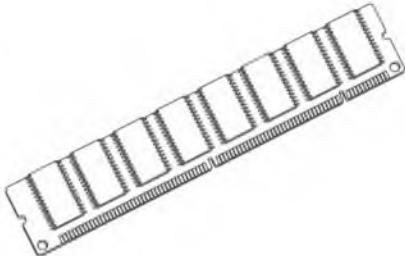


Fig. 9.12. Esquema de un módulo.

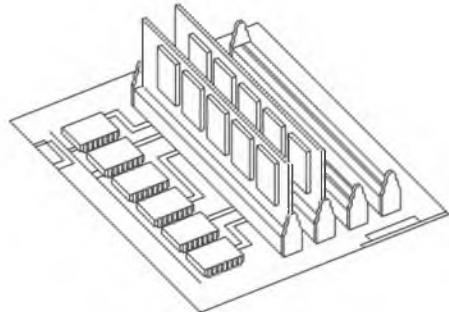


Fig. 9.13. Inserción de los módulos en una placa base.

Tabla 9-4. Descripción de parte de un DRAM DIMM.

<i>Pin</i>	<i>Identificación</i>	<i>Descripción</i>	<i>Tipos de línea</i>
1	VSS	<i>Ground</i>	<i>Conexión a tierra</i>
2	DQ0	<i>Data 0</i>	<i>Línea de dato</i>
3	DQ1	<i>Data 1</i>	<i>Línea de dato</i>
4	DQ2	<i>Data 2</i>	<i>Línea de dato</i>
5	DQ3	<i>Data 3</i>	<i>Línea de dato</i>
6	VCC	<i>+5 VDC o +3.3 VDC</i>	<i>Alimentación</i>
7	DQ4	<i>Data 4</i>	<i>Línea de dato</i>
8	DQ5	<i>Data 5</i>	<i>Línea de dato</i>
9			
10			
11			
12			
13			
14			
15			
16			
17			
18			
19			
20			
21			
22			
23			
24			
25			
26			
27			
28	/CAS0	<i>Column Address Strobe 0</i>	<i>Línea de control de dirección de columna 0</i>
29	/CAS1	<i>Column Address Strobe 1</i>	<i>Línea de control de dirección de columna 1</i>
30	/RAS0	<i>Row Address Strobe 0</i>	<i>Línea de control de dirección de fila 0</i>
31	/OEO	<i>Output Enable</i>	<i>Habilitación para salida de dato</i>
32	VSS	<i>Ground</i>	
33	A0	<i>Address 0</i>	<i>Línea de dirección</i>
34	A2	<i>Address 2</i>	<i>Línea de dirección</i>
35			
36			
37			
38			
39			
40			

9.7.2.3 Velocidad del bus de memoria

Para calcular el ancho de banda de las memorias se realiza el producto entre el ancho de bus expresado en bytes y la frecuencia efectiva de trabajo en MHz, también denominada velocidad física o real

Por ejemplo, una memoria identificada como *DDR200* se llama también *PC1600*, porque $(64 \text{ bits}/8) \text{ bytes} * 200 \text{ MHz} = 1600 \text{ MB/s} = 1,6 \text{ GB/seg}$, que es la velocidad de transferencia o ancho de banda del bus de memoria.

Cuanto más rápido sea un dispositivo, más difícil será de fabricar y, por lo tanto, más caro. Esta es la razón fundamental por la que la tecnología DDR SDRAM está instalada desde hace bastante tiempo en el mercado de memoria para computadoras.

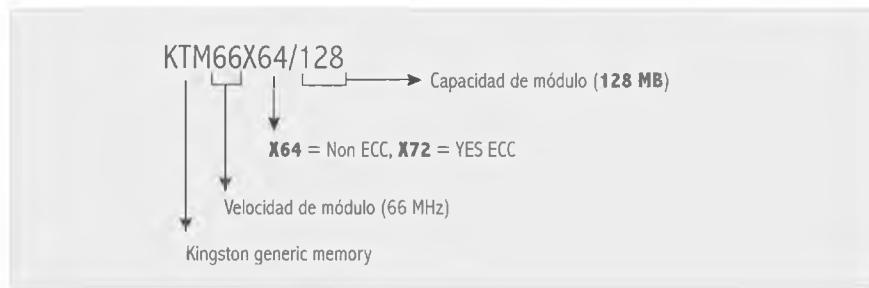
9.7.2.4 Caso ejemplo: DDR SDRAM (double data rate synchronous DRAM)

Éste es un ejemplo de tecnología. Esta memoria envía los datos dos veces (*Double Data Rate*) por cada ciclo de reloj, como consecuencia opera al doble de velocidad del bus del sistema o bus de memoria, sin necesidad de aumentar la frecuencia de reloj. De esta forma, una memoria con tecnología DDR que funcione con una señal de reloj "real", de, por ejemplo, 100 MHz, enviará tantos datos como otra sin tecnología DDR, que funcione a 200 MHz. Por ello, las velocidades de reloj de los DDR se suelen dar en lo que podríamos llamar "MHz efectivos o equivalentes" (en nuestro ejemplo, 200 MHz, "100 MHz x 2").

Esto significa que, a pesar de utilizar la misma frecuencia que sus antecesoras, la velocidad viene indicada con un valor duplicado: *DDR200* (100 MHz). La conocida denominación PCxxx no indica megahertz, sino la tasa de transferencia máxima en MB/s: *PC1600*.

DDR2-800 trabaja a 800 MHz con un bus de memoria de 400 MHz y ofrece un ancho de banda de 6,4 GHz, que se calcula como 64 bits del bus dividido 8 para obtener la cantidad de bytes (o sea, 8 bytes) por 800 MHz (6400 MHz = 6,4 GHz); de ahí la denominación *PC6400*.

9.7.2.5 Caso ejemplo: identificación de un chip SDRAM



9.7.2.6 Detección y corrección por ECC

El ECC 743 (*Error Checking And Correction*) es un código de bits que permite la detección y la corrección de errores, basado en un algoritmo complejo, que se utiliza fundamentalmente en computadoras de alta gama, como los servidores de red. El sistema trabaja en conjunción con el controlador de memoria y anexa a los bits de datos los denominados ECC, que se almacenan junto con los de datos. Estos bits extras o redundantes, junto con la decodificación correspondiente, sirven para realizar una comprobación en el momento de la lectura.

Su diferencia principal con la paridad es que puede detectar el error de un bit y **corregirlo**, con lo que, en general, el usuario no detecta que el error se produjo. Según el controlador de memoria utilizado, el sistema ECC también puede detectar errores de 2, 3 y 4 bits (muy raros), aunque en este caso no puede corregirlos; en estas situaciones el propio sistema devuelve un mensaje identificándolo como "un error de paridad".

En ambos casos, paridad o ECC, cuando se detecta un error se produce una interrupción no mascarable.

Ejercicio 4:

Pruebe ahora interpretar la información técnica siguiente:

<i>Capacidad de almacenamiento</i>	<i>1 GB</i>
Tipo	DRAM
Tecnología	DDR II SDRAM
Factor de forma	DIMM de 240 pines
Velocidad de memoria	800 MHz (PC2-6400)
Tiempos de latencia	CL5
Comprobación de integridad de datos	ECC
Características de la RAM	8K refresh
Configuración de módulos	128 · 64
Organización de los chips	64 · 8
Voltaje de alimentación	1.8V
Ranuras compatibles	1 x memoria - DIMM de 240 pines

9. 8 La memoria como en un espacio lógico

Para el procesador la memoria es un hardware, con forma de matriz de $m \times n$, que contiene las instrucciones y los datos para procesar, con muchos megabytes o gigabytes de almacenamiento, identificables por una dirección física (*physical address*). Ésta se puede considerar el número ordinal del byte dentro del mapa de direcciones a las que se puede acceder.

El procesador no hace diferencia entre las distintas tecnologías que pueden constituir una memoria principal. Simplemente envía la dirección física del byte vía bus de direcciones.

La manera en que una memoria se administra y se gestionan sus accesos depende fundamentalmente del sistema operativo. La evolución de los sistemas operativos en cuanto a administración de memoria y las nuevas plataformas de los procesadores, que le permiten administrarla de manera más eficiente, constituyen el núcleo central de este apartado.

Además, puede suceder que la computadora tenga menos memoria real que su posible capacidad de direccionamiento. Por ejemplo, una computadora puede tener un procesador cuyo posible espacio de direccionamiento es de *4 Gbytes* y sólo tener instalados *2 Gbytes*.

También puede suceder que la cantidad de bits de una referencia a memoria no alcance para acceder a toda la memoria. Por ejemplo, en los microprocesadores *Intel 80286* se utilizaban números de 16 bits para direccionar a 1 M direcciones. En ambos casos, la capacidad de direccionamiento lógico no es congruente con la capacidad física real de la memoria. Esto conduce a la aplicación de técnicas de direccionamiento a memoria. En estos casos, las direcciones en los programas no son direcciones físicas sino lógicas y para lograr el acceso deberán traducirse (mapearse). El procedimiento de "mapeo" consiste en aplicar un algoritmo para establecer la correspondencia entre direcciones lógicas y físicas, que a veces incluyen una o más tablas que contienen "una parte" de la dirección.



El procesador no hace diferencia entre las distintas tecnologías que pueden constituir una memoria principal. Simplemente envía la dirección física del byte vía bus de direcciones.

A continuación, se pretende detallar cómo el procesador y los programas que ejecuta "ven" la memoria. Llamaremos modelo a la vista lógica de la memoria que es totalmente independiente de la estructura interna de cada chip y de su organización en la placa.

9.8.1 Almacenamiento de bytes en memoria. *Big-Endian* y *Little-Endian*

El almacenamiento en la memoria, ya sea de datos o código de programa, sigue un orden específico. Como se podrá visualizar más adelante, el byte **menos significativo** se almacena en la dirección numéricamente más **baja** y el más significativo en la más alta. Esta forma de almacenamiento se denomina *Big-Endian* y se aplica tanto al código como a datos de tipo no numérico. Si el dato es numérico y su tamaño es de 2 o 4 u 8 bytes (palabra, doble palabra, cuádruple palabra), cada octeto se almacena en memoria en forma invertida. Por ejemplo, si la representación hexadecimal de una palabra es *ABCD*, en memoria se almacena primero *CD* y luego *AB*. O sea que la palabra se lee *CDAB*. En esta convención denominada de almacenamiento inverso o *Little-Endian*, el byte **menos significativo** se almacena en la dirección más **baja**. Esto ocurre para cualquier entidad numérica, incluso para datos en representación de punto flotante o cuando se almacena una referencia a memoria. Por ejemplo, el desplazamiento hexadecimal *0300* de una instrucción *MOVAH [0300]* se almacena como *8A260003*, donde *8A26* es el código de operación almacenado *Big-endian* y la referencia a memoria, *0003* almacenado *Little-Endian*.



Fig. 9.14. Esquemas de almacenamiento en memoria y almacenamiento en registros para estructura de datos tipo palabra y doble palabra.

9.8.2 Gestión de memoria y modos de operación de los procesadores

Una de las formas de administrar la memoria es hacerlo sin protección. En los microprocesadores *80x86* al modo de trabajo desprotegido se lo llamó modo real. En los sistemas operativos que admiten multitarea uno de los recursos para compartir es la memoria, por lo tanto, se deberían prevenir conflictos entre las distintas áreas locales de cada una de ellas. Esto es responsabilidad en parte de la administración del sistema operativo y en parte del soporte hardware, que desde el procesador gestiona el mapeo de algunos procesadores. El modo de operación del procesador, denominado modo protegido, implica, entre otras cosas, protección de memoria. Por supuesto que en ambos modos la memoria no tiene la misma vista lógica, razón por la cual ambos modos de operación no pueden estar activados en forma simultánea; el procesador puede conmutar de un modo al otro. En la actualidad, y por citar un ejemplo, un microprocesador *AMD64* puede operar en cuatro modos distintos. O sea que hay una de-

pendencia entre la forma de administrar la memoria y el soporte que brinda el hardware del procesador para hacerlo posible. Éste es el motivo por el que se incluyen algunos conceptos de administración de memoria en relación con el soporte hardware que requiere.

La forma en la que se obtenía una dirección física en modo real se encuentra ampliamente detallada en el caso ejemplo. Si bien en el presente ha caído en desuso, algunos procesadores y sistemas operativos permiten commutar a él. Aquellos lectores que pueden acceder desde sus computadoras personales a este modo pueden realizar los ejercicios allí planteados con el propósito de comprender en la práctica cuestiones más abstractas relacionadas con el estudio de procesadores más evolucionados, pues el enfoque del laboratorio les permite comprender en cada ejercicio los fundamentos teóricos relacionados con el procesamiento de instrucciones, la gestión de memoria y la gestión de archivos.

Por dar un ejemplo, las plataformas de 32 bits en vigencia admiten varios modelos de gestión de memoria:

- Como un espacio de direcciones físicas, conocido como espacio lineal de direcciones o modelo plano.
- Como un espacio que alberga segmentos puros, que son bloques lógicos de longitud variable o modelo segmentado.
- Como un espacio que alberga páginas, que son bloques lógicos de tamaño fijo o modelo paginado.
- Como un espacio que alberga segmentos-paginados, que es un modelo híbrido.

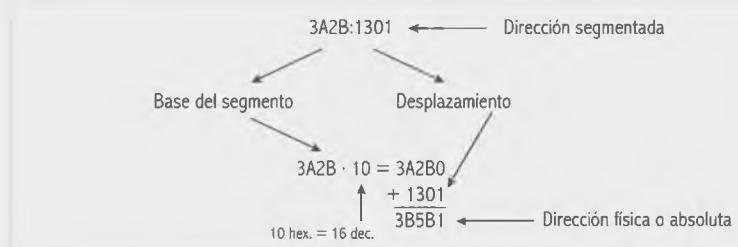
Caso ejemplo: cálculo de direcciones físicas en modo real

Por su simplicidad usaremos como primer ejemplo el modo real de los procesadores Intel x86.

Una dirección de memoria se representa con un número binario que permite identificar cada octeto dentro de ella. Cuando la cantidad de bits de la dirección es 16, el rango de direccionamiento queda limitado a $(2^{16} - 1) = 65535$, que es la dirección del último octeto direccionable, esto es, que de 0 a 65535 hay 65536 octetos direccionables. Se deduce que la capacidad de memoria total es, entonces, de 64 Kbytes (65536 octetos). Si se quiere acceder a más memoria, por ejemplo, a 1 mega, se puede segmentar la memoria en bloques de hasta 64 KB. Para ello se utilizan dos entidades de 16 bits y un algoritmo que las relacione. La primera entidad hace referencia a una zona de la memoria dentro del megabyte; la segunda indica el desplazamiento del octeto dentro de esa zona. Como el desplazamiento es de 16 bits, entonces la capacidad máxima de cada zona de memoria es de 64 KB. Si el total de la memoria es 1 MB y cada zona es de 64 KB, habrá 16 zonas (1024 KB/64 KB). Cada una de las zonas es un segmento. La primera entidad indica dónde comienza el segmento y la segunda, la posición del octeto dentro del segmento. Así, un octeto de memoria puede identificarse con una dirección lógica, por ejemplo, 3A2B:1301.

La pregunta ahora es: ¿Cómo se calcula la dirección física? Se sabe que 1 mega es $1 K \cdot 1 K = 2^{10} \cdot 2^{10} = 2^{20}$, esto implica que con 20 bits se calcula la dirección de la última palabra ($2^{20} - 1$). Se necesita un algoritmo que permita generar con dos referencias de 16 bits una dirección de 20 bits. Esto se logra así: A $3A2B_{(16)}$ se le agrega un $0_{(16)}$ o, lo que es lo mismo, se lo multiplica por $10_{(16)}$. A este valor se le suma $1301_{(16)}$, que es el desplazamiento dentro del segmento. Entonces, $3B5B1$ es la dirección física del octeto en memoria, que también se conoce como dirección absoluta.

Caso ejemplo: algoritmo de cálculo que realiza la MMU (Memory Manager Unit) del procesador en modo real



9.8.3 Modelo de memoria segmentada pura

Para acceder a la memoria física se puede utilizar una técnica denominada segmentación, que conceptualmente consiste en dividirla en territorios pequeños con límites determinados por el software. Un segmento es un bloque lógico en memoria, de tamaño variable. El control del tamaño de los segmentos es responsabilidad del sistema operativo que los administra. Los segmentos contienen un solo tipo de objeto, por ejemplo, en una memoria se puede organizar un segmento para el código ejecutable de un programa, otro para los datos y otro para una pila que él utilice. Segmentos distintos no necesitan estar en zonas contiguas, pero es necesario que un segmento como entidad lógica si lo esté. En el esquema se representan tres segmentos, cuyo acceso se referencia según el algoritmo presentado en el caso ejemplo anterior.

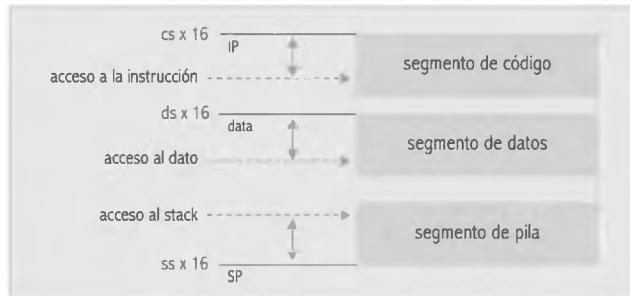


Fig. 9.15. Esquema que representa tres segmentos, cuyo acceso se referencia según el algoritmo de cálculo que realiza la MMU del procesador en modo real.

Denominaremos dirección lógica a la entidad binaria que identifica una locación de memoria a nivel software, que no se corresponde con la dirección física y, por lo tanto, necesita ser mapeada. Para direccionar una posición de memoria dentro de un segmento, cada dirección lógica se compone de dos entidades: una para identificar el segmento y otra para identificar un desplazamiento dentro del segmento. El cálculo lo hemos visto en el ejemplo anterior.

9.8.4 Modelo de memoria virtual

Ya en la época de las primeras computadoras comerciales, la memoria principal quedaba chica en relación con los requerimientos del programa, y el concepto de un programa almacenado en su totalidad en memoria principal parecía brillar en la teoría de los libros y hundirse en la práctica. Por aquél entonces, las memorias secundarias de tipo disco ya existían y los

programadores simulaban el concepto original de programa almacenado fragmentando el programa. De esta manera, un fragmento cabía en la memoria física y el programador provocaba la búsqueda del fragmento siguiente cuando el actual ya se había ejecutado. Con el tiempo se fue perfeccionando una técnica para que tanto la fragmentación como el intercambio de fragmentos entre el disco y la memoria se hicieran en forma automática; esto es, en la actualidad la gestión está a cargo del sistema operativo sin intervención del programador. Esta técnica se conoce como memoria virtual y posibilita la ilusión de contar con una memoria principal lo suficientemente amplia como para contener el programa y los datos, aunque esto no sea real.

La técnica de memoria virtual involucra por un lado memoria y discos y, por el otro, al procesador y al sistema operativo que la administra. Por supuesto que una dirección lógica perteneciente al espacio de direccionamiento virtual es mayor que una dirección física, y la llamaremos dirección virtual. Cuando se crea una tarea se asigna en disco un archivo temporal, que es copia del archivo ejecutable dividido en secciones, es un área de almacenamiento separada de la que ocupa el archivo ejecutable. Cada sección puede ser un segmento, una página, o un segmento paginado. La idea de "fraccionar" el ejecutable es que se puedan llevar a memoria secciones de código, o datos, más pequeñas para alojarlas en memoria principal en un menor espacio de almacenamiento. Como no se llevan todas las secciones, cada vez que se agote el uso de cada una se deberá demandar por otras alojadas en disco. De este modo los modelos de memoria que utilizan memoria virtual se conocen como segmentación por demanda y paginación por demanda, y se describirán a continuación.

Una dirección virtual debe traducirse a una dirección física, o sea que se mapea. En un modelo virtual el mapeo es una actividad que se intercala durante la ejecución del programa, por cada búsqueda de instrucción y por cada búsqueda de dato; asimismo, desde el punto de vista del hardware la gestiona una unidad específica, cuya función es traducir la dirección virtual a dirección física.



La memoria virtual es una forma de administrar la memoria para asignar más de ella a cada tarea, utilizando almacenamiento en disco, que suele ser el dispositivo de acceso directo más rápido; así, se simula que se dispone de una memoria mucho más amplia que la que el sistema tiene en realidad.



Un segmento es un bloque lógico de tamaño variable. Cada segmento contiene información de la misma clase (código, datos, pila).

¿Qué es la memoria virtual?

Aplicable a todas las ediciones de Windows Vista
¿Qué edición de Windows Vista estoy usando?

Si el equipo no tiene suficiente memoria de acceso aleatorio (RAM) para ejecutar un programa o una operación, Windows usa la memoria virtual para compensar la falta.

La memoria virtual combina la RAM del equipo con espacio temporal en el disco duro. Cuando queda poca RAM, la memoria virtual mueve datos de la RAM a un espacio llamado archivo de paginación. Al mover datos al archivo de paginación y desde él, se libera RAM para completar la tarea.

Cuanto mayor sea la RAM del equipo, más rápido tenderán a ejecutarse los programas. Si el equipo se ralentiza porque falta RAM, puede considerar la posibilidad de aumentar la memoria virtual para compensar. Sin embargo, el equipo puede leer los datos de la RAM mucho más rápido que de un disco duro, por lo que la mejor solución es agregar RAM.

Fig. 9.16. Definición de memoria virtual.

9.8.5 Modelo de memoria virtual paginada o paginación por demanda

9.8.5.1 Mapeo directo

Para administrar un archivo temporal bajo esta modalidad, se requiere una tabla de mapeo alojada en la memoria principal, con tantas entradas como páginas virtuales tenga el archivo. El método consiste en dividir la memoria principal en bloques de longitud fija, llamados páginas físicas, y el espacio de direccionamiento virtual en bloques de igual longitud, llamados páginas virtuales. Por ejemplo, si se considera como longitud fija de una página 1 Kbyte, una memoria principal organizada como una matriz de 8192×8 quedará dividida en 8 páginas físicas.

Si el espacio de direccionamiento virtual es de 64 Kbytes, quedará dividido en 64 páginas virtuales. Así, la dirección virtual referencia con 6 bits el número de página; 2^6 es igual a 64, y 2^{10} es igual a 1 K. Los 10 bits restantes hacen referencia a una de las 1024 (1 K) posiciones dentro de la página. En cambio, el identificador de página física es de 3 bits para referenciar a una de las 8 páginas físicas. Uno de los bits de atributo de página, alojado en la tabla de mapeo indica la presencia o no de una página virtual en la memoria principal, en la entrada de la tabla que le corresponda. Por ejemplo, en una página que contiene datos y está presente, los 3 bits identificadores de la página física concatenados con los bits del campo DATA de la instrucción logran la dirección física que recibe el MAR (dirección de $3 + 10 = 13$ bits, donde $2^{13} = 8192$ posiciones totales de la memoria principal).

Todo programa tiene asignado un espacio de memoria virtual (grande) mapeado en memoria física (más pequeña). Mientras el proceso de mapeo sea exitoso, o sea que una dirección virtual se transforme en una real de una página presente, no se producen errores; en cambio, si el programa intenta acceder a una posición perteneciente a una página no presente en la memoria principal, el procesador genera un error conocido como *page fault* (falta de página), que provoca una interrupción de tipo excepción. El error se advierte al sistema operativo, cuya función será buscar una nueva página en el disco.

Antes de esto libera una página y, si ésta se actualizó, primero la copia en el disco antes de reemplazarla (este procedimiento se denomina *swap out*). Recién entonces se utiliza esa página física con la página demandada por el programa. Cuando se recupera una página almacenada en disco, el procedimiento se denomina *swap in*. Al producirse continuas liberaciones y recuperaciones de páginas, la gestión de memoria virtual puede congestionarse y entrar en un estado conocido como *trashing*, término que denota el carácter de depósito que tiene la memoria de disco durante la gestión de memoria virtual (*trash* = basurero). Un término más adecuado para denotar un intercambio excesivo de páginas es hiperpaginación. La utilización eficiente de la memoria virtual depende básicamente del sistema operativo, esto es, de una técnica de software que determine el número de página virtual que debe retirarse de la memoria principal con un algoritmo de reemplazo eficiente. Éste selecciona la posición que tenga menos probabilidad de ser referenciada *a posteriori*, y así evita *page faults* futuros; también realiza la gestión de recuperación de una página desde disco y su alocación en la memoria principal.

De todas estas actividades la gestión del sistema operativo que más tiempo demora es la de una E/S (memoria-disco y disco-memoria), por lo tanto, no es deseable una sobrecarga de *swaps*, incluso puede provocar que el uso de memoria virtual, si no está bien administrada, traiga más problemas que soluciones.

9.8.5.2 Mapeo asociativo

La cantidad de páginas físicas en memoria real será siempre menor que la cantidad de páginas virtuales posibles. Así, para el ejemplo dado, $2^3 = 8$ páginas físicas y $2^7 = 128$ páginas virtuales, resulta una tabla de mapeo de 128 entradas, de las cuales sólo 8 contendrán informa-

ción de referencia a página física, o sea que 120 están siempre vacías; esto resulta inefficiente respecto del espacio de almacenamiento utilizado en memoria real para alojar la tabla.

Una mejor idea es crear una tabla que tenga igual cantidad de entradas como páginas físicas existan y que pueda implementarse en una pequeña memoria asociativa.

Cada entrada de la tabla contiene un campo que indica un número de página virtual actualmente presente en memoria principal y un campo que indica el número de página física correspondiente. Cuando se produce una consulta a la tabla, se compara el número de página virtual demandado con todos los campos "número de página virtual" de la tabla; si hay coincidencia y, por ejemplo, se está buscando un dato, entonces la dirección se logra concatenando los bits del número de página física con los del campo DATA del código de la instrucción que está en ejecución.

9.8.6 Memoria virtual segmentada o segmentación por demanda

En este modelo se organiza el espacio de direcciones virtuales en bloques de tamaño variable, los segmentos. El concepto de segmentación es el mismo, sólo que ahora los segmentos pueden residir tanto en la memoria principal como en el disco. Tomaremos como caso ejemplo el modelo de un procesador INTEL de 32 bits. En él la unidad de cálculo de dirección física se denomina unidad de gestión de memoria (MMU o *Memory Manager Unit*), pero esta vez el cálculo de la dirección física se realiza de otra manera, debido a que el modo del procesador es de 32 bits y administra la memoria virtual en modo protegido.

La MMU está constituida por dos unidades: **unidad de segmentación** y **unidad de paginación**, como se muestra en el esquema que sigue. Como la memoria principal es más pequeña que la virtual, la unidad de segmentación se encarga de detectar si un segmento se encuentra en el disco y no en la memoria física, y se lo comunica al sistema operativo para que posibilite el traslado. Cuando sólo se activa la unidad de segmentación, la traducción de una dirección virtual de 46 bits genera una dirección lineal que corresponde directamente a una dirección física de 32 bits, o sea que la unidad de paginación está inhabilitada.



La MMU está constituida por dos unidades: **unidad de segmentación** y **unidad de paginación**.

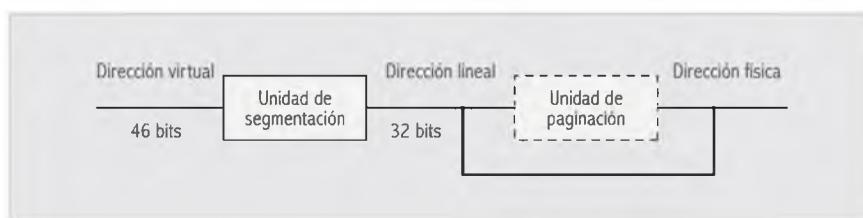


Fig. 9.17. Esquema de las unidades de la MMU.

Una tarea se asocia con su **tabla descriptora de segmentos**. Cada entrada de esta tabla contiene: un campo de 32 bits, identificador de la base del segmento (información que sólo tiene sentido si el segmento reside en memoria principal), un campo de 20 bits, que representa el tamaño máximo que el segmento puede alcanzar, y un campo de 12 bits, que permite determinar los atributos, por ejemplo, un bit de presencia, el bit de segmento actualizado, un bit de tipo de segmento (código, dato creciente, dato decreciente), etcétera.

Si el sistema operativo admite que se puedan cargar varias tareas distintas en la memoria, cada una estará asociada con su **tabla descriptora de segmentos**. La tabla se referencia con un registro base especial asociado al microprocesador, denominado **registro de tabla**



La **tabla descriptora de segmentos** se referencia con un registro base especial asociado al microprocesador, denominado **registro de tabla local activa** (Local Descriptor Table Register o LDTR).



La pila es una estructura de datos con un criterio de acceso LIFO (el último que entró es el primero en salir), que, asociada a un proceso, permite almacenar parámetros y direcciones de retorno.

local activa (LDTR). Este registro almacena la dirección de comienzo de la tabla asociada a la tarea que se ejecuta en la CPU. Para conmutar de una tarea a otra, el sistema operativo cambia el valor del LDTR y así cambia el puntero a la nueva tabla descriptora. La LDT de una tarea permanece en memoria mientras la tarea existe, como un recurso para administrar la gestión, y es creada, actualizada y eliminada por el sistema operativo. La MMU consulta esta tabla por cada nuevo cálculo de dirección física, ya sea de una instrucción en fase de búsqueda o de un dato en fase de ejecución. Los datos pueden ser del programa, conocidos con el atributo dato creciente, o de la pila, conocidos como "dato decreciente" (habiéndole cuenta del criterio LIFO de esta estructura).

Una dirección virtual está constituida por dos entidades: un campo selector y un campo desplazamiento. Para acceder a un segmento de código de la tarea en una *x86*, se consideran como selector los 14 bits de orden superior del registro de segmento denominado CS (*Code Segment*), y como desplazamiento el valor del EIP (*Instruction Pointer de 32 bits*). Estas dos entidades constituyen la dirección virtual que ingresa en la MMU, cuando se debe calcular la dirección física de una instrucción. El selector se considera el índice que apunta a una de las entradas de la **tabla de descriptores de segmentos** (o sea que es el desplazamiento respecto del valor del LDTR).



Fig. 9.18. Interpretación de una dirección virtual.

Caso ejemplo: dirección virtual IA-32

El campo selector tomado de un registro de segmento es "interpretado" por la unidad de segmentación, de modo que los 14 bits de orden superior constituyen el índice en la tabla de descriptores. TI indica si se está accediendo a la tabla de descriptores locales o globales y RPL marca el nivel de privilegio del segmento. Los 32 bits restantes se toman del campo EDATA de la instrucción, en el caso de un segmento de dato, o del EIP o del ESP, en el caso de segmentos de código o pila, respectivamente.

En el caso de un acceso a segmento de código, la dirección física se calcula sumando el valor de la base del segmento, ubicada en el descriptor, al valor del registro de desplazamiento EIP, y se conoce como dirección lineal que, en este caso, es la dirección física buscada.

Una ventaja de este modelo es el nivel de privilegio asociado a cada segmento, especificada en los 2 bits de orden inferior del registro de segmento (en este caso CS), que inhibe, por ejemplo, la operación de escritura si el segmento contiene código ejecutable, que no debe modificarse. La asignación de un segmento a distintas tareas, vía la tabla de segmentos, permite que varios procesos utilicen un segmento almacenado una sola vez en memoria sin necesidad de copias, con el consiguiente ahorro de memoria principal. En este caso el segmento se almacena en un área común determinada por el sistema operativo y conocida como área global. El área global cuenta con su propia **tabla descriptora de**

segmentos globales y con su propio registro base que apunta a la tabla GDTR (registro de tabla global activa o *Global Descriptor Table Register*).

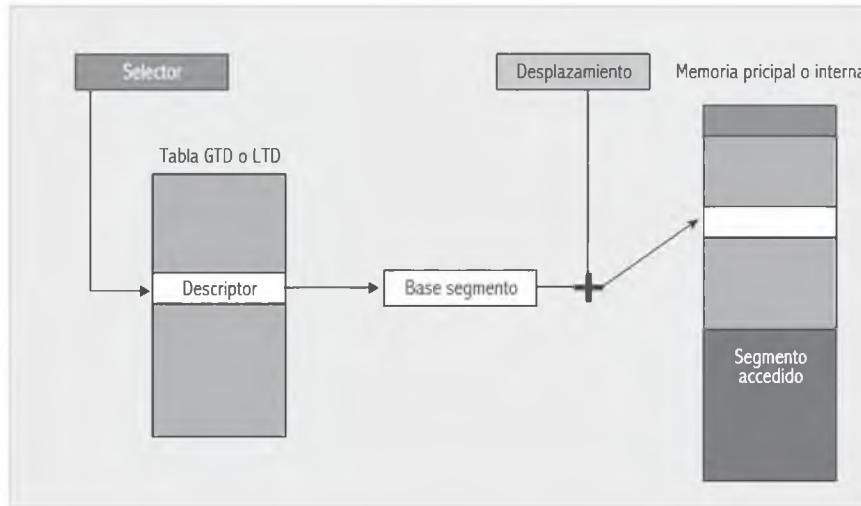


Fig. 9.19. Obtención de la dirección física.

9.8.6.1 Caso ejemplo de un descriptor de segmento en una IA-32 bits

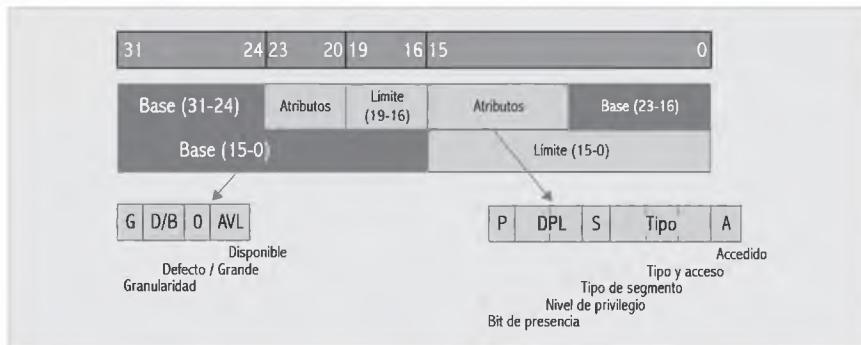


Fig. 9.20. Descriptor de segmento en una IA-32 bits.

Como se ve en el esquema, un bit en el campo atributo indica si el segmento está presente en la memoria principal. Si esto es así, entonces los campos BASE se unifican de modo de indicar la base del segmento presente en memoria principal; a este valor se sumará el campo desplazamiento de la dirección virtual. Por supuesto que la información escrita como base sólo tiene sentido para un segmento con bit de presencia seteado.

Descripción de algunos atributos

G bit de granularidad indica si el segmento está a su vez dividido en páginas. En este caso, se debe considerar el campo límite como cantidad de páginas.

P bit Present indica si el segmento está en memoria principal.

S System indica si el segmento es de sistema.

Modelo de memoria virtual con segmentos paginados



Fig. 9.21. MMU con la unidad de paginación habilitada.

En este caso cada segmento es una agrupación de páginas de longitud fija, o sea que la longitud del segmento varía en relación con las páginas asociadas a él. Se debe pensar que un segmento es una unidad de información físicamente contigua; si el tamaño del segmento de una tarea es mayor que los huecos libres que existen en la memoria principal, la tarea deberá esperar a que se produzca un espacio de tamaño suficiente como para contenerlo. La idea de dividir un segmento en páginas de tamaño fijo y reducido permite que el segmento mantenga su propiedad de entidad lógicamente indivisible, cuyo tamaño se adapta al objeto que contiene (código, datos o pila), pero que pueda estar físicamente distribuido en zonas de almacenamiento en memoria principal no contiguas. Además, no es necesario que se aloje en la memoria en su totalidad, basta con encontrar un espacio físicamente contiguo para una de sus páginas. Esto permite el aprovechamiento de huecos libres de menor tamaño.

En este nuevo modelo la **dirección lineal** ingresa en la unidad hardware, denominada **unidad de paginación**, que la traduce a una **dirección física** (una posición dentro de una página física). El funcionamiento de la unidad de paginación es optativo. La paginación en este procesador opera sólo en modo protegido. Teniendo en cuenta que con 32 bits de la dirección lineal se alcanza un espacio de direccionamiento de 4 gigabytes (2^{32}) y que el tamaño de cada página está definido en 4 KB (2^{12}), la cantidad de páginas virtuales posibles es un millón ($2^{32}/2^{12} = 2^{20}$); en consecuencia, para direccionar una página se necesitan 20 bits.

Cada vez que la unidad de paginación detecta ausencia de página en memoria genera la excepción *page fault*, que provoca un llamado al sistema operativo para que traiga la página del disco.

En el modelo presentado como mapeo directo, para saber si una página está en memoria principal o no, se debería mantener, para ésta nueva dimensión, una tabla de un millón de entradas que indicase su ausencia o presencia. Para no alojar una tabla tan grande en memoria, la traducción de la dirección se efectúa en dos niveles: directorio de tabla de página y tabla de páginas.

La unidad de paginación crea para cada tarea una tabla de 1 K entradas de 32 bits cada una, denominada **directorío de tablas de página**, que reside en memoria principal. El registro base que apunta al comienzo de esta tabla es un registro del microprocesador denominado *CR3*. El valor de los 10 bits de orden superior de la dirección lineal de 32 bits constituye el desplazamiento dentro del directorio de tablas, y los 32 bits del contenido de la entrada accedida en el directorio de páginas constituyen la base de una segunda tabla de páginas. Dentro de esta tabla los 10 bits centrales de la dirección lineal constituyen el nuevo desplazamiento. Por último, la entrada así accedida de la **tabla de páginas** de 32 bits se divide en 20 y 12. Los 20 de orden superior se concatenan con los 12 bits de orden inferior de la dirección lineal para formar la dirección física final. Los 12 bits de orden inferior de la entrada de la tabla de páginas son los bits del campo atributo de página, de los cuales no todos se utilizan. A continuación, se indican los atributos más importantes asociados con ellos.

D Bit dirty, indica si la página se escribió.

P Bit present, indica si la página está presente.

A Bit Access, indica si se accedió a la página, tanto para lectura como para escritura.

TLB (Translation Lookaside Buffer)

Es una memoria de tipo caché que almacena las últimas direcciones físicas traducidas por la unidad de paginación. Su inclusión en el diseño pretende solucionar el retardo producido por el acceso a las tablas de página almacenadas en la memoria principal, cuyas zonas de almacenamiento para lectura/escritura son de tecnología dinámica. Si la dirección está presente en la caché, se obtiene la dirección física en pocos nanosegundos; si no se encuentra, el tiempo de traducción sólo se penaliza en esos pocos nanosegundos.

9.9 Administración de memorias externas

Toda memoria alojada en una unidad de E/S se denomina memoria externa o secundaria; para algunos autores también se llama memoria de masa o masiva. Su inclusión en este apartado tiene como objetivo que se pueda establecer su relación, desde el punto de vista lógico, con los dos niveles jerárquicos superiores en la memoria principal.

Las características técnicas y la organización física de los dispositivos de almacenamiento se abordarán en el capítulo Dispositivos de entrada/salida.

La razón por la cual ubicamos las memorias secundarias en un último nivel de la jerarquía presentada responde a que uno de los parámetros elegidos para establecer esa jerarquía es, como ya se explicó, el tiempo de acceso a la información que contienen, ya que se consideran muy lentas con respecto a las memorias de semiconductores. En este nivel se utiliza como medida de tiempo el **μs (microsegundo = 10^{-6} seg)**. La mayoría de los parámetros de tiempo que permiten comparar dispositivos de distintos fabricantes se expresa en términos de **μs**.

Ejercicio:

Indique cuántos nanosegundos representan un microsegundo.

Las memorias externas se incluyen en toda computadora debido a su carácter no volátil, o sea que son imprescindibles para retener información cuando el sistema está desconectado de la fuente de alimentación. También se denominan memorias secundarias, nombre utilizado por los autores que llaman memoria principal o interna a la memoria de semiconductores fuera de la CPU y consideran el término interno como dentro de la computadora, aun cuando la memoria esté fuera de la CPU.

En un principio la CPU obtiene un programa y los datos necesarios desde el almacenamiento secundario. El programa se copia literalmente desde su ubicación en el almacenamiento secundario a la memoria de lectura/escritura RAM de la computadora, lo que le permite ejecutarse con mayor rapidez. Además, la CPU sólo tiene acceso directo a esta última por medio del bus de direcciones. De la misma forma, cualquier dato necesitado por el programa se copia desde el almacenamiento secundario hacia la memoria y esos datos retornarán al almacenamiento si se los modifica y se quiere guardar los cambios en forma permanente.

Los **dispositivos de E/S** más utilizados para “soportar” estas memorias son el disco rígido y los dispositivos de tecnología óptica, como la unidad de DVD.

Las memorias externas actúan de depósito y toda información que no se necesite para el proceso actual se almacena en ellas hasta que sea demandada. Los parámetros para tener en cuenta a la elección del **soporte** son:

- El costo por bit almacenado.
- El tiempo de acceso
- El modo de acceso.
- La capacidad de almacenamiento.
- El tipo de tarea que utiliza la información.

Se denomina **soporte** al elemento físico que retiene los bits de información. Los soportes utilizados con mayor frecuencia son los medios magnéticos. En estos soportes el costo por bit está dado por la densidad de grabación medida en bytes por pulgada (BPI o *Bytes Per Inch*). La densidad de grabación depende de la calidad del soporte y la capacidad del transductor para representar o detectar un bit en el medio.

El tiempo de acceso a la información se mide en microsegundos y depende del tipo de unidad accedida. Su lentitud respecto de las memorias de semiconductores se debe fundamentalmente al carácter electromecánico de la unidad o dispositivo de E/S. Para considerar dos parámetros que influyen en el tiempo de acceso, se puede considerar el tiempo de búsqueda en el soporte y el tiempo de transferencia entre el par memoria principal-memoria secundaria.

9.9.1 Archivos

Todo tipo de información se aloja en las memorias externas en una estructura denominada archivo (*file*).

Un código ejecutable es un archivo de programa y los datos de entrada o salida de un proceso constituyen un archivo de datos. Un archivo se diferencia de los demás por el **nombre simbólico** que se le asigna en el momento de su creación.

Por lo general, el nombre simbólico hace referencia al tipo de información que contiene. Sin embargo, la forma de nombrar archivos la determina el sistema de archivos del sistema operativo de la computadora; estas pautas están documentadas y se deben respetar, para que el archivo pueda ser reconocido por el sistema.

Todos los sistemas operativos admiten un nombre constituido por caracteres y es posible que se haga diferencia entre un nombre en minúscula y el mismo en mayúscula. Algunos sistemas operativos admiten una segunda parte para el nombre, como en el caso de Windows, que se denomina extensión.

En algunos casos esta extensión es una convención. Por ejemplo, todos los programas fuente de lenguaje C++ deben tener la extensión ".C++" para ser compilados. En otros la extensión sólo ayuda al usuario a recordar el tipo de archivo, sin que esto sea información específica para el sistema operativo.

9.9.1.1 Estructura de archivos

En principio, los archivos se pueden estructurar de las siguientes maneras:

- Una cadena de bytes, cuyo significado depende del proceso que lo consulta.
- Una agrupación de registros lógicos. Los archivos se dividen en unidades de acceso denominadas registros lógicos. Si se los compara con un fichero manual, cada ficha constituye una unidad de acceso y equivale, por lo tanto, a un registro lógico. El registro lógico está constituido por unidades elementales con sentido propio, por ejemplo,

el registro de un alumno en un archivo de datos personales contendrá el número de su DNI, el legajo y su apellido y nombre; estos datos "con sentido propio" se denominan campos. En cada operación de lectura o escritura (*read/write*) se accede a un registro lógico para recuperar o modificar un campo.

- Una agrupación de archivos relacionados entre sí constituye una base de datos.

9.9.1.2 Acceso a archivos

Un proceso que puede leer todos los bytes o registros del archivo en orden, comenzando desde el principio, sin saltar los registros fuera de ese orden, se dice que es un archivo al que se accede en forma secuencial.

Se dice que se accede al archivo en forma directa cuando un proceso puede leer bytes o registros de un archivo fuera de orden o puede acceder a los registros considerando uno de sus campos como una clave. Esta técnica involucra un acceso al azar (*random*).

9.9.1.3 Atributos de un archivo

El sistema de archivos asocia a cada uno cierta información adicional, por ejemplo, el día y la hora en que se creó y su tamaño. Esta información extra se conoce con el nombre de **atributos del archivo**. La lista de atributos varía en grado considerable de un sistema operativo a otro. Las siguientes son algunas de las posibilidades:

- Protección: determina quién y de qué manera se puede acceder al archivo.
- Sólo lectura: el archivo puede leerse, pero no modificarse.
- Escondido: el archivo existe pero no aparece en la lista del directorio o lista de archivos existentes.
- Sistema: es un archivo de uso exclusivo del sistema operativo.

9.9.1.4 Operaciones sobre archivos

Los diferentes sistemas operativos permiten realizar operaciones para almacenar y recuperar la información en los archivos. Estas operaciones se ejecutan mediante llamadas al sistema (*system calls*). En cada sistema operativo varía la especificación semántica del comando pero no su función:

- *Create* (crear): permite crear el archivo sin datos para asignarle alguno de sus atributos.
- *Delete* (borrar): permite borrar el archivo y liberar el espacio ocupado en el soporte.
- *Open* (abrir): todo archivo debe ser abierto por el proceso antes de ser leído o escrito. Esto permite que el sistema operativo busque los atributos y la lista de direcciones de disco. Es usual que durante la ejecución de una tarea esta información se almacene en la memoria principal en una tabla, para un acceso rápido hasta que el archivo se cierre.
- *Close* (cerrar): todo archivo que deja de utilizarse debe cerrarse para liberar su tabla de acceso en memoria principal.
- *Read* (leer): permite leer el archivo. La tarea que lee debe especificar cuánta información necesita, así como indicar un *buffer*, espacio en memoria principal, sobre el que desea colocar la información.
- *Write* (escribir): permite escribir el archivo. Se debe indicar la posición de comienzo de escritura, teniendo en cuenta que si esa posición es intermedia en el archivo, los datos allí presentes se sobreescreiben.

- *Append*: permite escribir el archivo sólo a partir del indicador de final de archivo, esto es, a partir del último byte grabado.
- *Rename* (renombrar): se utiliza para cambiar el nombre actual del archivo existente.



La densidad de grabación en una unidad de disco rígido depende de cuán pequeña es la distancia del campo magnético (*frame*) que representa un bit.

9.9.2 Sistema de archivos en discos de tecnología magnética

El sistema de archivos (*file system*) es un conjunto de programas del sistema operativo, cuyo objetivo principal es hacer transparente al usuario los detalles de la manera en que la información se almacena en archivos y se accede al soporte de información. La necesidad de almacenar información permanente en los soportes se debe a que:

- Todos los procesos en una computadora necesitan almacenar y recuperar información y es común que ella deba sobrevivir a la terminación del proceso.
- Es necesario almacenar grandes volúmenes de información, por lo tanto, se justifica la necesidad del uso de memorias externas.
- Muchos procesos pueden necesitar el acceso a la misma información en forma simultánea, por lo tanto, cuando un proceso modifica cierta información, ésta debe estar protegida del acceso de otro.

9.9.3 Disco magnético desde el punto de vista lógico

Las características técnicas de estos dispositivos de almacenamiento se abordarán en el capítulo Dispositivos de entrada/salida. Por el momento, sólo nos referiremos a las partes físicas necesarias para comprender su parte lógica, objetivo de este capítulo.

En la actualidad los discos magnéticos están constituidos por platos rígidos, de allí su denominación **disco rígido**, para diferenciarlos de los ya extinguidos discos flexibles. Están dispuestos en unidades de cabezas fijas o móviles, en grupos de uno o más platos. Más allá de todos estos detalles, debemos considerar que un soporte en un disco está constituido por un plato de aluminio, recubierto de material magnetizable con forma de disco, que gira alrededor de un eje dispuesto en la unidad, al que se accede por medio de una o más cabezas lectograbadoras.

La forma en que las cabezas recorren la superficie del **plato** describen círculos concéntricos denominados pistas (*tracks*).

Una **pista** es una división lógica y no física de la superficie del soporte, que se produce por efecto de la acción de rotación del soporte y de la posición fija de la cabeza al momento de grabación o lectura. La cantidad de pistas en la superficie depende del tipo de unidad a la que está asociado el soporte y del sistema de archivos que lo administre.

Si la unidad graba las dos caras de un plato o cuenta con más de uno, habrá muchas pistas de igual número (a la misma distancia respecto del eje) en las distintas caras. Podemos imaginarnos que por efecto de rotación se "visualizan" tantos **cilindros** concéntricos como pistas haya.

Por cada cara grabable habrá una cabeza lectograbadora. Todas ellas acceden de manera simultánea, como si fueran los dientes de un peine o **brazo actuador**, a un cilindro en particular, cuya identificación está dada por un número relativo que tiene que ver con la distancia del radio entre las cabezas y el eje. Por ejemplo, el cilindro 0 es el conjunto de pistas 0 de todas las caras grabables y es el cilindro más alejado del eje. Cuando la posición del brazo actuador está del lado externo del plato, éste es el cilindro al que más rápido se accede por estar más cerca de la posición inicial del brazo.

El concepto de cilindro (*cylinder*) es importante, dado que la información se graba en sentido vertical, o sea que se completa un cilindro y luego se posiciona el brazo en el siguiente. Considere que cada movimiento de cabeza es mecánico y, por lo tanto, resulta entonces óptimo grabar todas las pistas de igual número sin modificar la posición del brazo que sostiene las cabezas.

Cada pista está dividida en sectores que se pueden identificar con un número; el número de sectores por pista es fijo y, debido a que las pistas externas tienen una circunferencia mayor que las internas, estas últimas son grabadas a mayor velocidad.

El número de sectores que se pueden definir en la pista más interna limita el número de sectores por pista para todo el plato. Muchos discos usan una técnica llamada grabación de zona múltiple, para acomodar más datos en la superficie del disco. Esta técnica permite que el número de sectores por pista se ajuste para que se almacenen más sectores en las pistas externas que en las internas.

Si bien en la mayoría de los casos la unidad de referencia que hace el proceso para acceder a un archivo es el registro lógico, la unidad de acceso al soporte es una agrupación de ellos, denominada bloque. Ésta es la razón por la que los dispositivos de este tipo se conocen como dispositivos orientados al bloque, a diferencia de un teclado, que se denomina dispositivo orientado al carácter.

Como expresamos, en un disco por cada bloque se graban uno o varios sectores; cada sector es una porción de pista y, en términos del sistema operativo Windows, un bloque se denomina *cluster*.

Como todos los sectores de una pista almacenan la misma cantidad de bits, y a su vez los sectores de pistas distintas observan el mismo criterio, se deduce que la **densidad** de grabación aumenta en forma sucesiva respecto de la distancia del eje central (cuanto más cerca del eje, mayor densidad de grabación).

La cantidad de bits por pulgada depende de las cabezas lectograbadoras, la calidad del medio de grabación y las revoluciones por minuto a las que gira el disco. La cantidad de pistas por pulgada depende de las cabezas lectograbadoras, la precisión mecánica con la que se pueden posicionar las cabezas y la aptitud del disco de girar formando un círculo muy definido.

Si indicamos que la cantidad de bits por sector se mantiene constante, resulta, entonces, que la cantidad de bits de un bloque también. La determinación de la cantidad de información por sector o por bloque tiene relación con la capacidad total del volumen.

El sistema de archivos tiene la responsabilidad de aportar una estructura lógica, que permita un reconocimiento ágil de la ubicación de la información por acceder. Estos programas se dedican a organizar y administrar la gestión de almacenamiento, permitiendo la captura de bits desde el disco. Podemos indicar que dar formato a un volumen permite delimitar el territorio sobre el soporte. Originalmente, un disco es un territorio sin límites. Se dice que un programa que registra en el soporte una estructura productiva realiza un proceso de "dar forma" y es conocido como formateo o inicialización del soporte.

Los parámetros que determinan la ubicación de una entidad para su acceso son el número de pista o cilindro, el número de cara y el número de sector. El formateo (*format*) es el nombre con que se denomina al comando que ejecuta el programa de inicialización de un disco y que genera la grabación de marcas (*labels*), que identifican estos parámetros. De este modo, se puede localizar el lugar donde se graban los archivos. El proceso de formateo es, entonces, imprescindible para trabajar con el soporte.

Caso ejemplo: una forma de gestionar el acceso a un archivo

Cuando se ejecuta el comando que implica abrir un archivo existente, se determina dónde se encuentra éste en el disco. Para esto es necesaria la identificación de cilindro, cabeza y sector. Luego se transfiere esta información a la **controladora** de disco, que lleva al **motor del actuador** y, por lo tanto, al **brazo actuador**, a posicionar las cabezas sobre la pista adecuada. A medida que el disco rota, la cabeza apropiada lee la dirección de cada sector en la pista. Cuando aparece el sector deseado bajo la cabeza lectograbadora, se lee todo el contenido del sector en forma secuencial. Esta lectura vuelca los datos necesarios en una memoria ultrarrápida especial, denominada **caché de disco**. Entonces, el controlador que actúa de interfaz de la unidad de disco envía la información necesaria a la memoria principal de la computadora. Para almacenar datos en una unidad de disco rígido, el procedimiento es similar a recuperarlos, sólo que inverso.

9.9.3.1 Particiones

Un disco rígido es una unidad física y, por lo tanto, escapa al alcance de este capítulo. Sin embargo, los sistemas operativos las consideran unidades lógicas. En un mismo disco físico se pueden crear dos o más unidades lógicas, a las que se denomina **partición**. Desde el punto de vista del software, cada partición es un disco lógico o unidad lógica diferente. La partición se crea cuando se da formato al disco, su capacidad es fija y está constituida por un grupo de cilindros contiguos. Una de las razones por las cuales se crea más de una partición, es para que en cada una se instale un sistema operativo distinto; de esta manera, se crean dos entornos de trabajo diferentes para una misma computadora. De más está decir que el sistema se arranca desde una única partición por vez. En un mismo disco puede haber particiones primarias y secundarias. En una partición primaria se instala un sistema operativo, sólo una de ellas está activa por vez. Una partición secundaria se utiliza para almacenar archivos que, en algunos casos, pueden ser accedidos por distintos sistemas operativos, pero no contienen los programas de instalación.

Caso ejemplo: organización lógica de una partición

Consideremos la descripción de una estructura lógica en disco que nos permita visualizar los conceptos vigentes indicados hasta ahora. Sin embargo, como las computadoras pueden administrarse bajo distintos sistemas operativos, se mantuvieron algunas formas estándar comunes a todos ellos que serán descriptas.

Es común que en una partición primaria de un disco se organice un territorio denominado **área de sistema**, cuya información se utiliza para la administración del soporte, y un **área de datos** para los archivos.

En el área de sistema se puede encontrar:

- Un sector de arranque (*boot sector*).
- Una tabla índice que señala la locación inicial de cada archivo grabado, por ejemplo, una *File Allocation Table* o FAT.
- Una tabla de declaración de archivos, denominada directorio.

Cada sector se referencia internamente con la dirección formada por su referencia física: número de cilindro, cara y sector; no obstante, el sistema operativo identifica estos parámetros bajo un mismo número. Por ejemplo, cuando un *cluster* contiene un único sector, se puede decir que ese sector es el "cluster número N", que es lo mismo que identificar a un alumno por su número de documento o indicar que es el primero de la fila, o sea que se puede lograr una identificación con distintas referencias.

Un sector de arranque (*boot sector*) es una unidad lógica donde se almacena una rutina de carga, cuya ejecución produce la carga en memoria principal o interna de otros programas del sistema operativo.

En el momento de formateo se indica si una partición puede inicializar el sistema operativo o no.

Además de la rutina de arranque, la información que se puede encontrar en el sector de arranque es la que sigue:

- Número de bytes por sector
- Cantidad de sectores por *cluster* (la mayoría de las veces contiene más de uno).
- Cantidad de sectores reservados.
- Cantidad de FAT y de sectores que ocupa (para el caso de sistemas que organicen el espacio para el área de archivos de datos con esta técnica).
- Cantidad de entradas al nodo raíz, si se utiliza una organización de archivos tipo árbol.
- Cantidad total de sectores en la partición del disco.
- Técnica utilizada para la grabación de la información.
- Cantidad de sectores por pista.
- Cantidad de cabezas lectograbadoras.

Caso ejemplo: el sector de arranque de una computadora personal

Al encender la computadora, el sistema básico de E/S (programa alojado en memoria ROM) lee el “*master boot record*”, situado en el cilindro 0, cara 0, sector 1 del disco rígido que contiene la rutina de arranque y la descripción de las particiones (en este caso son dos, una primaria y otra secundaria).

Una vez que se detecta la partición activa, el sistema básico de E/S le transfiere el control a esa partición para que se ejecute la rutina, que, en este caso, carga los programas de Windows.

El sector maestro de arranque está constituido por dos partes básicas: el código de la rutina de arranque y cuatro entradas de 16 bytes para la descripción de las cuatro particiones que se pueden crear como máximo.

Cada entrada, de 16 bytes, describe características de una partición. El primer byte indica si la partición es activa o no, los 3 bytes siguientes se utilizan para indicar el comienzo de la partición, un byte indica el sistema operativo que está instalado en la partición, tres bytes indican el final de la partición, cuatro bytes, el número de sectores anteriores al comienzo de la partición y los últimos cuatro, el número de sectores totales.

Ejercicio:

Si los 3 bytes que indican comienzo o fin de partición se interpretan de la siguiente manera: 1 byte indica el número de cara; 2 bits del segundo byte, el número de cilindro que se completa con los 8 bits del último byte (en total diez bits) y los 6 bits restantes del segundo byte indican el número de sector. Asimismo, considerando que la numeración de caras y cilindros comienza en 0 y la de los sectores en 1, determine cómo se deduce dónde comienza la partición (cuántas caras, cilindros y sectores), si la combinación hexadecimal que obtiene es “3F BF B0”

9.9.3.2 FAT (File Allocation System)

Es una tabla de asignación de espacio para los archivos del área de datos. En cada entrada de la tabla se representa un *cluster* del área de datos. Así, la entrada 3 identifica al *cluster* 3 y esta entrada a su vez contiene un número que actúa de puntero a otro *cluster*. Si la entrada 3 contiene un 6, significa que el archivo ocupa el 3 y continúa en el 6. En este sistema un archivo está asociado a un *cluster* de comienzo, indicado en el directorio, como veremos más adelante.

Para mayor claridad, supongamos que en el directorio dice que el archivo comienza en el *cluster* 2; con ese número se accede a la entrada 2 de la FAT. La entrada señalada contiene el número de *cluster* donde continúa el archivo, o sea que el archivo continúa en el *cluster* 3. Consultando la entrada 3, se deduce que el archivo continúa en el *cluster* 6. Así, las sucesivas entradas se encadenan hasta que el *cluster* final del archivo se marca con una entrada de la FAT que contiene todos unos. Si un *cluster* de datos está vacío, su entrada correspondiente en la FAT contiene el número 0.

3	6	0000000	0000000	7	1111111	0000000
Entrada 2	Entrada 3	Entrada 4	Entrada 5	Entrada 6	Entrada 7	Entrada 8

En realidad, éste es un esquema muy simplificado de la organización de la FAT. La función de la FAT es mostrar la cadena de *clusters* en la que se aloja el archivo. Inicialmente los archivos se almacenan en *clusters* sucesivos, pero cuando el usuario elimina ciertos archivos quedan huecos que podrán ser utilizados por otro nuevo. Es común que el hueco no tenga el tamaño suficiente como para alojar el archivo nuevo y, por lo tanto, se produzca un salto al próximo *cluster* vacío, de manera que permita alojar el resto de los datos que se desea almacenar.

Este procedimiento se repite una y otra vez y los archivos quedan fraccionados en el soporte. El ejemplo de tabla muestra que el archivo que comienza en el 2, continúa en el 3, del 3 salta a 6 y finaliza en el 7; o sea que ocupó 4 *clusters* y, en el momento de su creación, los *clusters* 4 y 5 estarán ocupados, razón por la que del 3 salta al 6. En este momento ambas entradas figuran con 0, por lo que el archivo que las ocupaba dejó el espacio libre. Si el próximo archivo para grabar ocupa 3 *clusters* de dato, entonces la entrada 4 se completará con un 5 y la entrada 5 con un 8, luego la 8 con 1, que indican fin de archivo.

Si bien es lógico y deseable que los archivos aprovechen los huecos vacíos del soporte, recuperar un archivo fraccionado en *clusters* ubicados en cilindros diferentes insume más tiempo, porque provoca el movimiento mecánico del brazo, por lo que es mejor que un archivo se aloje en *clusters* sucesivos. En este tipo de organizaciones se produce una reestructuración de la información que se conoce con el nombre de desfragmentación (*defrag*).

9.9.3.3 El directorio raíz y los subdirectorios

El directorio es un archivo que se actualiza en relación con las operaciones que se realizan con los archivos. Es una tabla con información propia de los archivos y a cada uno se asigna una entrada que básicamente indica su nombre y atributos.

El área de archivos o área de datos está dividida en *clusters* de igual tamaño (que varía según el tamaño del disco).

Para acceder a un archivo guardado en un disco, se accede al directorio y se lo ubica por su nombre de archivo en el disco y un número del *cluster* de comienzo, que permita su ubicación en el soporte y el acceso a la primera entrada de la FAT, para conseguir su cadena de locaciones.

Además de archivos, un directorio puede contener otros directorios, que dependen del primero; así se establece una organización jerárquica que se puede identificar como un árbol invertido. De este modo, se pueden separar archivos utilizados en aplicaciones distintas en directorios diferentes.

La información que se puede encontrar en una entrada se resume así:

- Nombre del archivo.
- Byte de atributos.
- Hora de última modificación.
- Fecha de última modificación.
- Número de *cluster* donde comienza el archivo.
- Tamaño del archivo.

Es importante detenerse en los atributos de archivo. Los archivos admiten operaciones como abrir, cerrar, leer, escribir, crear, borrar (*open*, *close*, *read*, *write*, *create*, *delete*, respectivamente). Estas operaciones pueden ser reguladas por los atributos asociados al archivo. El estado de un atributo puede representarse con un bit, por ejemplo:

- *Read-Only*: el archivo puede abrirse, cerrarse y leerse, pero no puede borrarse ni escribirse.
- *Hidden*: el archivo existe pero no se muestra en las búsquedas normales dentro del directorio.
- *System*: el archivo pertenece al sistema operativo.
- *Volume*: indica que este archivo identifica el soporte, o sea que es una cadena de caracteres que identifica la unidad lógica con un nombre simbólico.

El procedimiento de acceso requiere ciertas operaciones. Citemos, por ejemplo, la creación, una lectura, o grabación, o su eliminación del soporte.

Creación

Se accede al directorio comparando el nombre del archivo con cada una de sus entradas. Si el archivo existe, no podrá crearse. Como consecuencia, deberá existir la opción de reemplazo de uno por el otro, previa consulta al usuario. Si el archivo no existe, se le asigna una entrada al directorio y se busca en la FAT. La primer entrada con contenido 0 se asigna como primer *cluster*, se graba y se busca otra entrada vacía; así, se arma una cadena que puede estar físicamente discontigua hasta la finalización del archivo.

Lectura/grabación

Se accede al directorio comparando el nombre del archivo por leer o grabar. Si se quiere leer y no existe, provoca un error archivo no encontrado (*file not found*); si existe, se accede a los *clusters* de datos, consultando alternativamente la FAT.



Los metadatos son datos estructurados que describen información. Esto es, son datos "acerca de otros datos". Un ejemplo puede ser: su identificación (área en la que está incluido), su calidad (nivel de precisión), su referencia (nivel de actualización).

Eliminación de un archivo

Se accede al directorio comparando el nombre del archivo con cada una de sus entradas. Si no se produce equiparamiento, se genera un mensaje del error; en caso contrario, se marca la entrada del directorio indicando que ésta queda libre; a su vez, se liberan las entradas de la FAT que le estaban asignadas. Es importante que le quede claro el concepto de que el archivo no se borra físicamente del soporte, sino que los territorios ocupados quedan liberados gracias a la actualización del directorio y las entradas de la FAT regrabadas con el número 0.

Lo más lógico en todos los soportes es que el área de sistema se aloje en el cilindro que permita el menor tiempo de acceso a las cabezas lectograbadoras, dado que éstas siempre son consultadas para acceder al soporte. Éste es el cilindro cuya distancia física de las cabezas es menor, mientras que los *clusters* de datos ocupan el resto de los cilindros.

Es importante recalcar que lo explicado hasta ahora es un ejemplo representativo de la organización lógica de un sistema de archivos específico. Otros soportes del mismo tipo se pueden organizar de manera distinta. El siguiente es otro ejemplo de organización del área de datos.

9.9.3.4 Sistema de archivos NTFS

Como su nombre lo indica (*New Technology File System*), es un sistema de organización de archivos que ordena la información almacenada en el soporte de manera distinta a los que utilizan FAT y es más eficiente para particiones de gran tamaño. Las características que los diferencian son:

- Compatibilidad mejorada con los metadatos
- Permite la estructura de datos denominada árboles-B
- Aprovecha mejor el espacio en disco
- Reduce la fragmentación de archivos típica del sistema FAT.

Los metadatos son archivos creados en forma automática por el sistema cuando formatea un volumen de NTFS. Se localizan utilizando una tabla de archivo maestra (*master file table*). Esta tabla es el punto de arranque de cualquier unidad bajo NTFS y se la puede analogar a la FAT, pero es mucho más que una lista de *clusters* vacíos.

Por cada archivo o directorio que se crea en el disco, se genera un registro en la MFT, que puede alcanzar un tamaño de 2 KB. Su función es almacenar información de los archivos o directorios con la forma de atributos, que dan la característica de "datos acerca de datos". Si el archivo es tan pequeño que los datos y los atributos de los datos entran en un registro, queda almacenado en la MFT y no requiere espacio adicional en el área de datos.

Cuando se agrega un registro más al MTF, que ocupa un lugar denominado zona de MTF y que se crea cuando se da formato a la partición, se calcula 10% más de la capacidad total del disco, a sólo efecto de agregar nuevos registros. Si este porcentaje no alcanza, se puede fragmentar la MTF y continuar su estructura en otra zona de disco.

La descripción de los registros de la tabla nos ayudará a comprender su funcionalidad.

Tabla 9-5. Descripción de los registros.

Nombre en inglés del archivo Metadata	Registro de la MFT	Descripción
<i>Master File Table</i> (MFT)	0	El primer registro contiene datos acerca de la propia MTF.
<i>Master File Table 2</i> (MFT2) or <i>Master File Table Mirror</i>	1	Es una copia de seguridad de la MTF que se almacena generalmente al final de la partición o en el medio.
<i>Log File</i>	2	Un registro de transacciones producidas en el volumen para que pueda ser recuperado en caso de error.
<i>Volume Descriptor</i>	3	Contiene la información propia de la partición.
<i>Attribute Definition Table</i>	4	Contiene el significado de cada atributo.
<i>Root Directory/Folder</i>	5	Éste es el puntero al directorio raíz del volumen.
<i>Cluster Allocation Bitmap</i>	6	Contiene una tabla de clusters ocupados y vacíos.
<i>Volume Boot Code</i>	7	Contiene una copia de la rutina de arranque.
<i>Bad Cluster File</i>	8	Contiene la tabla de clusters dañados para que no se utilicen.
<i>Quota Table</i>	9	Tabla de cuotas asignadas. Una cuota es una cantidad de espacio en disco que se le puede asignar a un peticonario.
<i>Upper Case Table</i>	10	Tabla que contiene la información para convertir los nombres de archivos a código UNICODE para que sean compatibles a nivel internacional.

9.9.4 Buffers y cachés de disco

Se denomina *buffer* de disco a una memoria de almacenamiento temporal en la que se aloja un bloque de información del disco. En un proceso de lectura, una vez que la cabeza recorre un sector, lo aloja en el *buffer* del controlador y luego lo envía a un *buffer* dispuesto en la memoria principal, donde concluye la tarea del controlador. En un proceso de escritura la información se envía de memoria principal al *buffer* del controlador, para luego grabarlo en disco. En la memoria principal se pueden alojar varios sectores, por lo tanto, es factible mantener tantos *buffers* de memoria principal como hayan sido definidos en la configuración del sistema. Esto agiliza los tiempos de búsqueda, ya que es más rápido buscar en memoria principal que acceder a disco de manera continua. Para optimizar aún más los accesos a disco, se puede asociar al sistema un caché de disco. Esto es, un *buffer* especial dotado de un algoritmo que permite determinar qué sectores son los solicitados con mayor frecuencia, para retenerlos y agilizar los tiempos de búsqueda. El caché puede estar alojado en el controlador de la unidad de disco (caché hardware), o bien, controlado por un programa de gestión del sistema operativo en memoria principal (caché software).

Ejercicio:

En su explorador puede encontrar la información relacionada con "guardar" lo más recientemente utilizado. Búsqelo e indique qué dice cuando consulta la ayuda por el contenido de "caché de página Web".

9.9.5 Discos virtuales

Ram disk o *virtual disk* son términos equivalentes que pueden llegar a confundirlo con otros conceptos que incluyan las palabras RAM o virtual. Esta facilidad no tiene la envergadura suficiente como para tratar el tema con mucho detalle y es, más bien, del ámbito de estudio de algunas extensiones de memoria en las computadoras personales. Si se considera que un acceso a disco puede tardar algunos microsegundos y que, a veces, un proceso depende de él para continuar su ejecución, sería interesante crear un disco ficticio en una parte de memoria RAM, de manera que se emule su actividad con la consecuente ganancia de tiempo. Se debe proveer un programa que haga posible la creación y el mantenimiento de este disco virtual. El programa debe generar el área de arranque y el directorio, así como la organización lógica que asumen los discos; o sea que se debe simular su funcionamiento. Se puede simular un disco duro usando como medio de almacenamiento la memoria interna; éste debe residir en memoria para atender los requerimientos que una aplicación pretende de este supuesto disco, interpretando los comandos típicos que se utilizan en la memoria externa. También se puede simular un disco duro empleando un servicio de almacenamiento *online* que ofrece varios GB como si se tratara de un disco duro local.

9.9.6 Sistema de archivos en discos de tecnología óptica

El primer sistema de archivos creado para medios ópticos denominados CD-ROM, esto es, que no aceptaban reescritura, fue especificado en la norma ISO 9660. Se trataba del primero que soportaba archivos para diferentes sistemas operativos. En tecnología óptica ésta es la característica más buscada, puesto que, en general, se utiliza el medio no sólo en distintas computadoras con sistemas operativos diferentes sino también en múltiples dispositivos digitales. En la actualidad, el formato más utilizado es el UDF (*Universal Disk Format*), que es la implementación de la norma ISO/IEC 13346, que mejora la anterior, ya que permite la grabación en medios ópticos. Sus siglas significan **formato de discos universal**, y permite el acceso a los soportes ópticos desde diferentes sistemas operativos. Es, pues, obligatorio para DVD y admitido por los CD. Como característica principal podemos indicar que admite la grabación por paquetes, aprovecha mejor el espacio de almacenamiento del disco compacto y mantiene sus atributos originales.

9.10 Resumen

En este capítulo hemos visto la memoria de una computadora desde el punto de vista de los componentes y desde el de la administración del sistema operativo. En cuanto a las distintas jerarquías en el subsistema de memoria, hemos pasado revista a las tecnologías disponibles para la implementación física, tanto de memorias volátiles como de memorias no volátiles, considerando como parámetros para establecer los distintos niveles, los tiempos de respuesta, la capacidad de almacenamiento y/o el costo relativo del almacenamiento de un bit. Hemos abordado las razones que justifican el uso generalizado de la memoria dinámica, como la actual tecnología de preferencia de tipo RAM en la memoria principal y la importancia de tener niveles de memoria *caché* para disminuir los accesos a ella. Esto sugiere que la velocidad con la que el procesador disponga de instrucciones y datos es de suma importancia, pues el factor disponibilidad incide

en su rendimiento en cuanto a velocidad de ejecución. Respecto del tiempo de acceso es razonable pensar que una memoria más veloz siempre mejora el rendimiento global, sin embargo, en un sistema real, el aumento o disminución de la prestación no siempre se da en la misma proporción que el aumento de velocidad de las memorias que lo componen. Las innumerables tablas de comparación de tiempos de acceso de los fabricantes de chips hacen referencia a la disponibilidad de información, desde que el componente recibe una orden de lectura hasta que se estabiliza su contenido en el bus de datos, y no a otros factores que inciden en su relación con los demás componentes del sistema. En el tiempo de acceso a un chip DRAM no se contempla el tráfico del bus, que depende de otras condiciones del sistema, y que se relacionan con el acceso a la memoria principal, pues influye en la dinámica de obtener bits. Considere que el procesador no es el único que está accediendo a memoria principal y sus accesos están limitados por el uso del bus en los ciclos de E/S. Estos ciclos se utilizan para transferir información entre la memoria principal y los dispositivos externos, por eso, el manejo de las interrupciones hardware, así como el detalle de cómo se lleva a cabo una transferencia influye en el tráfico del bus que asocia ambas unidades. Estas particularidades serán descriptas en el capítulo Dispositivos de Entrada/Salida, que considera sólo medios de almacenamiento masivo, y en el capítulo Transferencia de Información, en el cual no sólo se describen los distintos modos de transferencia sino también las interfaces hardware que se utilizan para la conexión de componentes.

9.11 Ejercicios propuestos

1. Indique qué significan las siglas DRAM, FPM, EDO y BEDO, correspondientes a tecnologías de memoria dinámica ya en desuso, y explique en cada caso cuál fue la mejora de una respecto de su antecesora.
2. Investigue qué significa el parámetro CL = Cas Latency (todas las memorias tienen una CL). ¿Cuál es su valor actual más común y qué significa CL2 CL2,5 CL3?
3. ¿Qué significan las siglas SIMM y DIMM y cuál es la diferencia entre ambos módulos de memoria?
4. Busque la ubicación de los pines asociados a la dirección física y a los datos en un módulo DIMM de 200 patillas o pines e incorpórelos en la práctica a modo de tabla.

Identificación del pin Descripción

5. Calcule la tasa de transferencia de una memoria DDR2-600 (memoria síncrona que transfiere datos dos veces por cada ciclo de reloj) si opera a 600 MHz y, por lo tanto, a 300 MHz de bus de memoria, e indique el valor de las "X" en la denominación PC-XXXX.

6. Para una memoria identificada comercialmente como PC-4800 o DDR2-600, indique el valor de operación expresado en MHz reales.
7. Investigue qué tasas de transferencia ofrece el mercado actual para su utilización en computadoras hogareñas, laptops, celulares u otros dispositivos a su elección. Indique su identificación y el nombre del fabricante.
8. ¿Qué diferencia hay entre una memoria DDR y una DDR2, desde el punto de vista del consumo, y por cuál optaría en el caso de una computadora portátil? Justifique su respuesta.
9. Averigüe el tamaño máximo del módulo de memoria que su PC acepta, el tipo de RAM y de conector y determine cuál de los cuatro tipos de RAM utiliza DRAM (EDO o FPM), SDRAM, DDR DRAM o RDRAM.
10. Los cuatro tipos se montan en uno de tres tipos de módulo: SIMM, DIMM o RIMM. Indique las características de cada uno de ellos.
11. "Los módulos SDRAM, DDR SDRAM y RDRAM nominalmente deben igualar o exceder la velocidad del bus frontal de la PC, que determina la velocidad del tráfico de datos entre la CPU y la RAM". Esta afirmación se encontró en una revista técnica conocida. Si es cierta, explique qué significa; en caso de que no lo sea, justifique su respuesta.



9.12 Contenido de la página Web de apoyo

El material marcado con asterisco (*) sólo está disponible para docentes.

Resumen gráfico del capítulo

Simulación

Ejercicios con memorias.

Animación

Conceptos generales sobre memorias.

Autoevaluación

Lecturas adicionales

Memoria de Martín Silva, es parte del libro "Sistemas Operativos" de Alfaomega Grupor Editor (48 páginas). Agradecemos a su autor por permitir que su escrito sea parte de las lecturas complementarias de esta obra.

Video explicativo (01:57 minutos aprox.)

Audio explicativo (01:57 minutos aprox.)

Evaluaciones Propuestas*

Presentaciones*

10

Instrucciones

Contenido

10.1 Introducción	258
10.2 Formato de instrucción.....	258
10.3 Modos de direccionamiento.....	262
10.4 Tipos válidos de instrucción.....	270
10.5 Resumen.....	271
10.6 Ejercicios propuestos.....	271
10.7 Contenido de la página Web de apoyo.....	275

Objetivos

- Comprender distintos tipos de instrucciones y las diversas modalidades de direccionamiento de operandos.



En la página Web de apoyo encontrará un breve comentario de la autora sobre este capítulo.

10.1 Introducción

En este capítulo se utilizarán formatos genéricos para representar los distintos tipos de instrucción, atendiendo a la modalidad utilizada en los manuales de procesadores reales. Como vimos en el capítulo Diseño de una computadora digital, un programa es una secuencia de operaciones o comandos denominados instrucciones, que la computadora ejecuta una a una. La ejecución de una instrucción se completa en cuatro pasos básicos: búsqueda de la instrucción en memoria, decodificación de la instrucción, ejecución de la instrucción y actualización necesaria del puntero de instrucción para proceder a la ejecución de la próxima. Cada instrucción en memoria se representa con una secuencia binaria, por ejemplo, `1011 1011 0000 0000 0000 0000`, donde cada grupo de bits es interpretado por la CPU como un campo. En la secuencia anterior `1011 1011` significa transferir una constante a un registro de cálculo y `0000 0000 0000 0000` es el campo que contiene el valor que se ha de transferir, en este caso, `0`. Cuando se presenta el set de instrucciones que puede ejecutar la CPU, se hace abstracción del contenido binario y se indican qué "campos" componen la instrucción y cómo se interpretan; esto se conoce como "formato" de instrucción y se puede representar como sigue:



Con la consecuente descripción del campo, donde:

- Cop es el código de operación de la instrucción.
- Data es el dato por operar.

Cuando el dato no es una constante, se puede obtener de memoria o de un registro.

Con la combinación de algunos modos de direccionamiento se obtienen otros; así, un Intel 80286 contaba con más de veinte modos, resultantes de los siete u ocho primarios descriptos en este capítulo.

10.2 Formato de instrucción

Cuando se utiliza el término "formato" de una instrucción, se hace referencia a la manera en que deben interpretarse los bits que constituyen el código de máquina de la instrucción.

El formato presentado para "X" es el más simple y obedece a un tipo de organización conocida como "de un solo acumulador"; como "X" es nuestra CPU "ábaco" o de estudio, la presentamos con un nivel de complejidad mínima. Todas las demás computadoras ejecutan instrucciones que no necesariamente afectan a datos y esto afecta el formato. En el caso de "X" el formato es el mismo, pero el campo de referencia al dato denominado Data se ignora durante la ejecución de esa instrucción.

Debemos estudiar formatos en las computadoras comerciales actuales. En los manuales de cada máquina se especifica el juego de instrucciones que su CPU puede comprender, el o los formatos en código binario de máquina que se utiliza y el formato de las instrucciones en simbólico de máquina.

10.2.1 Instrucciones sin dirección

La instrucción más simple está constituida por un grupo de bits que deben interpretarse como una unidad. Ésta representa únicamente al código de operación, esto es, cómo va a ser ejecutada después de haberse leído de memoria y colocada en la unidad de decodificación y secuenciamiento de la CPU.

Para dar un ejemplo, en el repertorio de instrucciones de Intel *80X86* podemos hacer referencia a la instrucción HLT (*halt*), que es de 8 bits y que indica a la CPU que debe detener la ejecución de las instrucciones del programa en curso y discontinuar su actividad. Como vemos, esta instrucción no afecta datos y, por lo tanto, su formato se representa así:



Otro ejemplo del mismo repertorio es la instrucción RET (*return*), cuyo código también es de un octeto y se representa en binario "1100 0011" o en hexadecimal "C3". Esta instrucción indica a la CPU que es el final de un procedimiento convocado por el programa con una instrucción CALL (*call*); de modo que es un final con "retorno" a la instrucción siguiente al llamado, o sea, la CPU retorna el procesamiento de las instrucciones del programa llamador.

Cuando una instrucción afecta un dato, éste puede estar implícito en el código de operación, por lo que su formato no cambia respecto del ejemplo anterior. Ése es el caso de la instrucción LAHF (*Load AH flags*), ejemplo del mismo repertorio, cuya función es cargar los *flags* (acarreo, paridad, acarreo BCD, cero y signo, denominados en esta CPU como CF, PF, AF, ZF y SF, respectivamente), en los 8 bits de orden superior del registro acumulador denominado AH (*A High*). Las banderas o *flags* constituyen el dato en cuestión y la acción está indicada en el código de operación. El octeto en binario es "1001 1111" y su representación en hexadecimal es "9F".

10.2.2 Instrucciones de una sola dirección

Éste es el formato utilizado por "X", en el que todas las instrucciones hacen referencia implícita, en el código de operación, al registro acumulador; por lo tanto, DATA1 hace referencia al dato afectado.



Ejemplo: ADA 100

En otros set de instrucciones también se utiliza este formato que exemplificaremos con instrucciones del set *80X86*, que utiliza el campo DATA1, como se describe a continuación.

10.2.2.1 Dato contenido en la instrucción

Cuando una instrucción afecta un dato y éste se encuentra incluido en el código de la instrucción, el formato cambia, o sea que debe incluir un grupo de bits que referencia al dato, que llamaremos en forma genérica campo DATA1. Para el repertorio Intel *80X86* exemplificaremos con dos instrucciones denominadas en "modo inmediato", a causa de que en el formato aparece el dato en la ubicación inmediata a código de operación. Ambas instrucciones ocupan tres octetos, uno para el código de operación y dos para el campo de dato; el simbólico de máquina es:

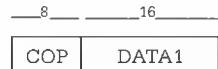
MOV AX,0000h

que indica una transferencia del dato cuyo valor en hexadecimal (h) es 0000 al registro acumulador AX. Otra instrucción del mismo tipo es

MOV BX,0000h

que indica la misma transferencia pero considerando como registro destino a BX.

El código de instrucción en hexadecimal para cada una de ellas es "B80000" y "BB0000", respectivamente. Para ambas, el formato de instrucción se representa así:

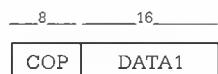


10.2.2.2 Dato referido por la instrucción

Continuando con instrucciones que afectan datos, podemos analizar una instrucción en la que el dato debe buscarse en la memoria principal. En este caso exemplificamos con otra instrucción del repertorio Intel 80X86, cuyo simbólico de máquina es:

MOV AX,[FFFF]

Esta instrucción también es de transferencia al registro acumulador, pero esta vez el dato se encuentra en la memoria y el código de la instrucción hace referencia a su posición en ella. Los corchetes indican "contenido de" y *FFFF* en hexadecimal es una información relativa al cálculo de la dirección efectiva donde se aloja el dato. Desde el punto de vista de la ejecución, la instrucción del ejemplo anterior (*MOV AX,0000*) y ésta (*MOV AX,[FFFF]*) generan eventos diferentes. En la primera el dato se aloja en forma inmediata al código de operación, por lo que no requiere un acceso a memoria, y se mueve desde la instrucción al registro X. En la segunda se debe calcular la dirección efectiva del dato en memoria y luego ordenar su lectura. Una vez transferido el dato vía Data bus, se lo puede "mover al registro AX". Esto implica que el código de operación de la instrucción es distinto; sin embargo, el formato de la instrucción no cambia, es el valor del código de operación el que indica que el contenido del campo DATA1 es una referencia al dato y no el dato en sí. El código de esta instrucción en hexadecimal es "A1FFFF" y en binario 1010 0001 1111 1111 1111 1111, o sea que utiliza el mismo formato:



Volvamos a recordar que éste es el formato utilizado por "X", en el que todas las instrucciones hacen referencia implícita, en el código de operación, al registro acumulador, por lo tanto, DATA1 hace referencia al dato afectado.

A continuación se representan otros formatos genéricos que se pueden encontrar en los repertorios de distintos procesadores.

10.2.3 Instrucciones de dos direcciones

Para reducir aún más el código del programa, algunos diseñadores optaron por agregar el campo DATA2 en algunas instrucciones, de modo tal que se utilice para referenciar el segundo operando. A su vez, referencia el lugar donde se almacena el resultado; como se puede observar, una instrucción reemplaza a tres del formato anterior. El formato resulta así:



Ejemplo:

SUME 100 A02

Equivale en "X" a:

LDA 100

ADA A02

STA A02

10.2.4 Instrucciones de tres direcciones

COP	DATA1	DATA2	DATA3
-----	-------	-------	-------

En otro formato se agrega el campo DATA3 para referenciar el lugar donde se almacena el resultado.

10.2.5 Instrucciones de cuatro direcciones

COP	DATA1	DATA2	DATA3	DATA4
-----	-------	-------	-------	-------

Este formato utiliza el campo DATA1 y el campo DATA2 para referenciar los datos afectados por el código de operación. El campo DATA3 referencia el resultado y DATA4 referencia la próxima instrucción por efectuarse. La mayor desventaja de este formato es la cantidad de bits que lo constituyen.

Un ejemplo de una instrucción ficticia puede ser:

SUME 100 A02 B2A 020

Que equivale en "X" a:

LDA 100

ADA A02

STA B2A

JMP 020

10.2.5.1 Instrucciones de formato fijo en sílabas

El formato de instrucciones para las nuevas tecnologías denominadas de 64 bits es exclusivo de estas plataformas. Se denomina sílaba a una agrupación de 128 bits que el procesador puede capturar por vez. Cada una de estas estructuras contiene tres instrucciones de 41 bits, lo que da un total de 123 bits, y un campo de 5 bits denominado plantilla o *template*, cuya información la generó el compilador. Si bien con 5 bits hay 32 combinaciones posibles, la realidad demuestra que sólo algunas de ellas es válida. Estas combinaciones válidas establecen en qué unidades de ejecución se pueden practicar las instrucciones en paralelo; esto significa que las tres instrucciones podrían ejecutarse en forma simultánea, utilizando una unidad de ejecución diferente; por ejemplo, una puede hacer referencia a la carga de registros desde memoria, otra, a una operación entre enteros en la unidad de enteros y la última, a una operación de coma flotante en la unidad de coma flotante. Esta característica de ejecución de las instrucciones se denomina EPIC (*Explicitly Parallel Instruction Computing*).

El formato de cada instrucción individual es

7 bits	7 bits	7 bits	6 bits	14 bits
DATA1	DATA 2	DATA 3	SALTO	COP

10.2.5.2 Interpretación de un código de máquina

Las instrucciones se implementan en lenguaje de máquina identificando con "campos de bits" los siguientes parámetros:

- Dónde se ubica la identificación del registro fuente y el registro destino.
- El tamaño de los datos.
- El tamaño del desplazamiento.
- Los registros de base e índice que se utilizan.

Por ejemplo, para las instrucciones del set *80X86*, que tienen una longitud de 1 o 2 octetos o bytes, el primer byte del código de máquina indica el código de operación, el tamaño y la ubicación de los operandos y guarda la forma siguiente:

00ccc0dw

donde *ccc* indica con 3 bits la instrucción que se va a ejecutar; *d* indica con 1 bit la dirección (si *d* = 0, la fuente es un registro y el destino es la memoria; si *d* = 1, la fuente está en la memoria y el destino es un registro); *w*: indica con 1 bit si el dato es de 1 byte (*w* = 0) o de 1 palabra (*w* = 1).

El segundo byte del código de máquina indica el modo de direccionamiento, de la manera siguiente:

mm sss aaa

donde *mm* indica con 2 bits parte del modo de direccionamiento, que se completa con *aaa*; *sss* indica con 3 bits un registro que será el registro fuente, si *d* = 0, o el destino, si *d* = 1; *aaa* completa con 3 bits el resto del modo de direccionamiento o indica el registro destino.

Por ejemplo, *01D1* en binario es *0000 0001 1101 0001* y corresponde a la instrucción *ADD DX,CX*, que es una instrucción del set más utilizado en los últimos 15 años del *80X86*

<i>00 000 0 0 1</i>	<i>11 010 001</i>
<i>ccc d w mm sss aaa</i>	
<i>ADD</i>	<i>DX,CX</i>

Para comprobarlo sólo variamos el registro fuente de *ADD CX,CX* en hexadecimal =*01C9*, en binario *0000 00011100 1001*

00 000 0 0 1 *1100 1001*

10.3 Modos de direccionamiento

Los datos afectados durante la ejecución de una instrucción están alojados en memoria principal, de modo que la instrucción hace referencia durante su ejecución a uno o más bytes que constituyen el dato u operando. Una instrucción que hace referencia a un dato lo involucra en una operación, que puede ser la lectura o la aplicación de un algoritmo que lo modifique.

En la computadora analizada, hasta ahora las instrucciones emplean un solo método que permite la obtención o la escritura del dato con un solo acceso a la memoria. Sin embargo, la variación en la técnica para acceder al dato permite una mayor flexibilidad para involucrar estructuras de datos más simples (el caso de asignación de constantes) o más complejas (el caso de sumar elementos de dos vectores).

Esto tiene como primera consecuencia una variación en el formato del código de instrucción, de modo que se pueda identificar de alguna manera la técnica empleada; el set de instrucciones de "X" es muy limitado. Entonces es imperativo conocer las técnicas básicas, que en el presente se utilizan combinadas entre sí y generan multiplicidad de modos de direccionamiento. Es el código de operación de la instrucción el que definirá con bits predeterminados el modo de direccionamiento; el set de instrucciones de una computadora con varios modos se amplía y da lugar a nuevos formatos.

Ejemplo considerado del set de instrucciones 80X86:

Si tomamos el simbólico *MOV AX,[0000]*, cuyo código de instrucción es *A10000* (que significa mover un dato desde la memoria hacia el registro acumulador), y lo comparamos con *MOV AX,CX*, cuyo código de instrucción es simplemente *89C8* (que significa mover un dato desde un registro llamado CX al acumulador) vemos con claridad que el código de la instrucción cambió numéricamente.

En realidad, lo que cambia es el código de operación; A1 en el primer MOV y 89C8 en el segundo. El campo DATA 0000 existe en el primero únicamente. La forma en que la CPU interpreta que esto ocurre así se debe a que el análisis de los bits que determinan el modo de direccionamiento también cambió, indicando una nueva técnica de búsqueda de operando durante la etapa de decodificación. Aunque no es de interés saber cuáles son esos bits, es importante notar que para un mismo verbo "MOV" (mover o transferir) ahora hay dos "instrucciones distintas". En la primera el dato se obtiene usando como referencia el valor entre corchetes 0000, y provoca una microoperación de lectura a memoria; en la segunda el dato se obtiene directamente de un registro interno de CPU que se identificó como CX (*counter*). El primer modo se denomina directo de memoria y el segundo, directo de registro.



El campo de referencia a dato, que hasta ahora hemos llamado DATA, no va a ser considerado de la misma forma, sino que su interpretación depende del valor en bits que determine una de varias técnicas para calcular la dirección efectiva (o física) del dato. A cada una de estas técnicas se la conoce como modos de direccionamiento.

10.3.1 Direccionamiento directo de memoria

Cuando el modo de direccionamiento es directo, la referencia al dato que especifica el campo DATA queda sin alteraciones y el tiempo de captación del dato depende sólo del tiempo de un acceso a memoria.

Ejemplo considerado del set de instrucciones 80X86:

En la instrucción *MOV AH,[0200]* los corchetes indican que el dato se encuentra "contenido" en la referencia 0200 de memoria y debe transferirse al registro AH (*A High* o parte alta del registro AX), que es un registro de CPU (para que algunas de las próximas instrucciones del programa lo involucren en algún tipo de cálculo).

Antes de la ejecución

CPU	lectura	Memoria	
		Referencia	Contenido
AH		0200	32
		0201	5F

Después de la ejecución

CPU	lectura	Memoria	
		Referencia	Contenido
AH	32	0200	32
		0201	5F

Sin embargo, hay dos modos que determinan que no se acceda a memoria para obtener el dato. El primero se llama **direccionamiento implícito** y el segundo, **direccionamiento inmediato**.

10.3.2 Direccionamiento implícito

En este modo el dato queda determinado por el mismo verbo y, en consecuencia, en el código de instrucción, en el campo código de operación (p. ej., *clear accumulator* o CLA, que significa poner a "0" el acumulador; en este caso el "0" es el dato implícito en el verbo *clear*). Todas las instrucciones que asignen un valor por medio del COP pertenecen a esta categoría.

En "X" este modo de direccionamiento no se usa en realidad, dado que todas las instrucciones comprenden al campo DATA, aun cuando lo ignoraran en algunos casos. La ventaja en este tipo de instrucciones es que la cantidad de bits que la definen representan el campo, el código de operación, no requieren fase de búsqueda del operando y, por lo tanto, su ciclo de instrucción es más corto.

10.3.3 Direcciónamiento inmediato

Las instrucciones que se adapten a este modo de direccionamiento involucran el dato en la instrucción en sí, pero ahora no en el campo código de operación sino en el campo DATA. Este tipo de instrucciones permite asignar un valor constante a un registro o una posición en memoria. La ventaja es que no se requiere el acceso a memoria para obtener el dato, la desventaja es que no se puede modificar el valor del operando durante el transcurso de la ejecución del programa ya que es una constante.

Ejemplo considerado del set de instrucciones 80X86;

En la modalidad inmediata, el operando se incluye como parte de la instrucción. Por ejemplo, *MOV AH,05*. Aquí, el número 5 forma parte de la instrucción en su totalidad. En otras palabras, el número se especifica como una constante numérica en la misma instrucción y no hay necesidad de acceder a memoria. La instrucción anterior mueve el valor 5 al registro AH.

Antes de la ejecución

MOV AH, 05

AH

Después de la ejecución

MOV AH, 05

AH 05

10.3.4 Direcciónamiento indirecto

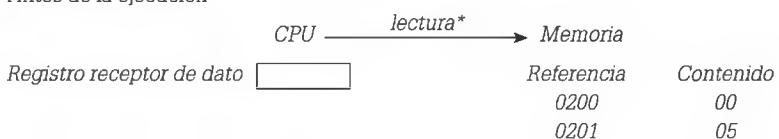
En este modo el campo DATA contiene una dirección de una posición de memoria que, a su vez, contiene la referencia al dato. Para acceder al dato se requieren dos accesos a memoria.

Ejemplo hipotético *mover reg, ((0200))*

Microoperaciones de lectura de un dato para ser transferido a la CPU:

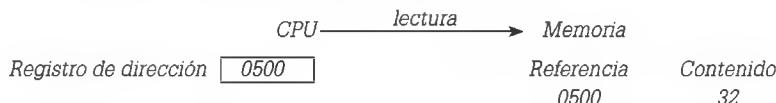
1. Transferir referencia al dato al registro de Direcciones Memoria.
2. Dar orden de lectura.
3. Transferir contenido leído al registro de Direcciones de Memoria.
4. Dar una nueva orden de lectura.
5. Transferir el contenido leído a la CPU.

Antes de la ejecución



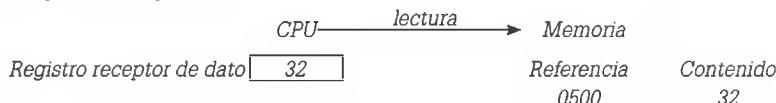
*CPU solicita lectura y obtiene dos bytes.

Como la referencia es de dos bytes, la CPU considera el contenido de *0200* y del siguiente byte *0201*; para "construir" la nueva referencia, lee el primer byte y lo acomoda a la derecha del registro de dirección y luego el segundo byte a su izquierda.



Con esta nueva referencia ordena una nueva lectura y obtiene el dato, que es el valor *32*.

Después de la ejecución



Existen computadoras que tienen un nivel de indirección múltiple, esto es, se accede a una posición en memoria que indica con un bit que su contenido se usa como señalador o "puntero" de otra; a la vez, la nueva posición también es un *pointer*, y así sucesivamente hasta llegar a aquella cuyo bit indicativo de *pointer* sea cero. En esta última se halla entonces el dato buscado.

10.3.5 Direccionamiento de la CPU asociado a registros

Las microoperaciones generadas por la unidad de control para la búsqueda de una instrucción o su ejecución siempre determinan acciones sobre registros asociados a la CPU o a la memoria. Si consideramos que la operación que requiere más tiempo de CPU es la de acceso a memoria (lectura o escritura), deducimos que cuantos menos accesos a ella se necesiten mayor será la velocidad de procesamiento. Por este motivo, la CPU de las computadoras disponibles en el mercado cuenta con varios registros internos. Éstos se pueden inicializar o modificar y cargar con datos de memoria. Las instrucciones pueden hacer referencia a estas memorias locales que son los registros y optimizar así el tiempo de ejecución de los programas. Este grupo de registros se denomina registros generales. La instrucción hace referencia al número de un registro en lugar de hacerlo a una dirección de memoria.

En estos casos, el código de la instrucción no sólo especifica con determinados bits el modo de direccionamiento, sino, con otros bits, el número que identifica al registro que no ha sido cargado con el dato previamente.

Ejemplo con dos instrucciones del set *80X86*:

1. *MOV CX, DX*
2. *MOV DX,CX*

La instrucción 1 mueve el valor de *DX* a *CX* y su código de operación es *89D1*.

La instrucción 2 mueve el valor de *CX* a *DX* y su código de operación es *89CA*.

Sin hacer un análisis muy profundo de los bits del código, convertiremos ambos a binario y observaremos algunas igualdades y diferencias.

```
89D1 1000 1001 11 01 0001
89CA 1000 1001 11 00 1010
```

Podemos suponer que verbo y modo de direccionamiento están definidos por los primeros 10 bits, ya que son iguales en ambas instrucciones, y podríamos expresarlos como "mover de el registro origen hacia el registro destino en modo directo por registro".

A continuación, el registro destino está marcado en negrita:

89D1 1000 1001 11 010 001

89CA 1000 1001 11 001 010

Se observa que los números de los registro que corresponden a *CX* y *DX* son **010** y **001**. Por último, vemos la identificación de los registros origen marcados en negrita con los mismos valores pero al revés para cada instrucción:

89D1 1000 1001 11 010 001

89CA 1000 1001 11 001 010

Los registros de *CPU* se pueden utilizar para almacenar resultados parciales o direcciones que permiten establecer nuevas modalidades de direccionamiento a memoria. Los registros generales pueden estar conectados entre sí, por medio de buses, formando una pequeña memoria local, o a través de conexiones directas con compuertas de habilitación, en cuyo caso se consideran registros independientes.

10.3.6 Direcciónamiento directo por registro

La referencia a un registro siempre es más rápida, puesto que no hay acceso a memoria y, por lo tanto, se observa una ganancia de tiempo en el acceso al dato, pues en un acceso a memoria no sólo se debe tomar en cuenta su velocidad de respuesta, sino también la disponibilidad y la velocidad del bus de sistema.

Ejemplos en el set de instrucciones *80X86*:

MOV CX, DX (ambos son registros que almacenan dos bytes)

Antes de la ejecución

<i>CPU</i>
CX 0000
DX 4F7A

Después de la ejecución

<i>CPU</i>
CX 4F7A
DX 4F7A

10.3.7 Direcciónamiento indexado

En un programa se utilizan algoritmos que involucran índices, por ejemplo, para sumar datos en posiciones contiguas de memoria.

Imagine que en "X" el procedimiento para sumar 10 valores alojados en referencias consecutivas implica el mantenimiento de una variable que actúe de contador de la cantidad de sumas y un ciclo que se repita hasta que esa suma se efectúe 10 veces. Además, no sólo se deben contar valores sino también utilizar el único acumulador de "X" para incrementar la dirección en una unidad, lo que supone el almacenamiento previo de cada suma parcial en memoria. Si se aloja en CPU un registro que lleve el control del índice, la tarea se simplifica y se gana en velocidad, ya que el tiempo de actualizar un registro interno es mucho menor que el de actualizar una variable en memoria. Este registro se denomina índice (*Index Register*).



En programación un vector es un conjunto o una agrupación de variables del mismo tipo, cuyo acceso se realiza por índices. La forma de acceder a los elementos del array es directa, o sea que el valor del dato deseado se obtiene a partir de su índice y no hay que ir buscándolo elemento por elemento.

El set de instrucciones debe proveer una instrucción de carga del registro índice que actúa como "señalador de elementos". Es común que una CPU contenga más de un registro índice, por lo que esto se debe reflejar en el código de la instrucción.

Ejemplo considerado del set de instrucciones *80X86*:

Antes de la ejecución

Registros de CPU

05	AH Registro de 8 bits
0000	SI Registro índice de 16 bits

Vector

3B	A1	03	01	7A	4F	00	Contenido
0	1	2	3	4	5	6	Número de elemento

MOV SI,0003; transfiere el valor 3 al registro índice SI.

ADDAH, [SI]; suma al registro AH el contenido del elemento 3 del vector.

Después de la ejecución

Registros de CPU

06	AH Registro de 8 bits
0003	SI Registro índice de 16 bits

Vector

3B	A1	03	01	7A	4F	00	Contenido
0	1	2	3	4	5	6	Número de elemento

10.3.8 Direcciónamiento relativo a la base

Supongamos que un dato de un byte se aloja siempre en la quinta posición del vector anterior y que este vector se carga en memoria en distintos momentos, cada vez en un área de memoria distinta. Para acceder a este dato, antes de acceder al vector será necesario conocer la referencia de comienzo en la que se aloja y almacenar dicha referencia en un registro de CPU que se denomina base; luego se puede acceder al quinto elemento sumándole a la base la constante 4.

Ejemplo considerado del set de instrucciones *80X86*:

En esta modalidad se usan dos registros, el BX (base), que también puede ser utilizado como registro de cálculo, y el BP (base pointer).

Antes de la ejecución

AH Registro de 8 bits	AH 00
BX Registro base de 16 bits	BX 00 00

Vector

0200	0201	0202	0203	0204	0205	0206	Referencia
3B	A1	03	01	7A	4F	00	Contenido
0	1	2	3	4	5	6	Número de elemento

*MOV BX,0200; transfiere el valor 0200 al registro Base BX.
MOVAH,[BX]+4; transfiere al registro AH el contenido del elemento de la quinta posición.*

Después de la ejecución

Registros de CPU

7A	AH	0200	BX
3B A1 03 01 7A 4F 00			Contenido
0 1 2 3 4 5 6			Número de elemento

Ejemplo de combinaciones de más de un modo de direccionamiento basado en el set de instrucciones *80X86*:

Para la instrucción *MOV CX, [BX+SI+5]*, cuyo código de instrucción es *8B4805*, durante la fase de cálculo de dirección del dato se suma el contenido de BX al de SI (*Source Index*), y a esto la constante *5* (todos entre corchetes).

10.3.9 Direccionamiento a una pila (*stack*)

La CPU puede contar entre sus registros internos con un puntero a una pila de datos en memoria. Una pila es una estructura de datos a los que se accede según el criterio LIFO (*Last In First Out*). Este criterio supone que los datos van ocupando posiciones sucesivas y cuando se quiere leer el contenido de una ellas sólo se puede acceder a la última que se escribió en dicha estructura.

La operación de la pila puede compararse con una pila de platos, el último en colocarse es el que puede sacarse primero, sin separar la pila. El que lleva el control de las direcciones en la pila es el puntero (*stack pointer*). Su contenido dirige la primera posición vacía de la pila y se actualiza por cada dato agregado y cada dato extraído. Los datos extraídos no desaparecen físicamente de la pila sino que sus direcciones se liberan, ya que no son referenciadas por el puntero. La instrucción del programa sólo hace referencia al puntero cuyo contenido es información para el cálculo de la dirección efectiva del dato.

La pila es un lugar utilizado para almacenar el contenido de los registros de la CPU en forma temporal. Esto permite su reutilización como área de trabajo y la restauración de sus valores originales una vez concluida la operación.

Además, cuando un programa convoca a una subrutina (procedimiento), la misma instrucción de llamado (*call*) almacena en la pila la dirección de retorno que se utilizará al terminar la ejecución de la subrutina. La propia instrucción de retorno (*return*) "retira" la dirección de retorno y actualiza el valor del puntero de instrucción para que la próxima instrucción por ejecutarse sea la siguiente al *call*.

Cuando se ejecuta la subrutina, en la pila se almacenan sus variables locales; cuando la subrutina concluye, estas variables son descartadas o "retiradas" de ella.

Es aconsejable utilizar el BP (puntero de base) para acceder a datos en la pila. Por ejemplo, si se quiere armar una pila de datos en memoria, es aconsejable copiar el SP al BP y luego para direccionar la pila usar el BP más un desplazamiento para acceder a cualquier palabra de la pila. De esta manera, el SP siempre estará apuntando a la última palabra utilizada del segmento de pila, sin modificar su criterio LIFO.

A manera de ejemplo, se exponen algunos modos de direccionamiento del set de instrucciones *80X86* que pueden verse como pertenecientes a dos casos típicos:

- **Referencia a registro**, donde el operando está en la dirección del registro especificado en la instrucción.
- **Referencia a memoria**, donde para acceder al operando se deben sumar "hipotéticamente" las siguientes cantidades:

$$\boxed{\text{direcc. de segmento} + \text{direcc. base} + \text{índice} + \text{desplazamiento}}$$

donde:

1. La dirección del segmento se encuentra en el registro de segmento (CS, DS, ES o SS) y se multiplica por 16 antes de ser sumada.
2. La dirección base almacena el registro base (BX o BP).
3. El índice se encuentra en el registro índice (SI o DI).
4. El desplazamiento puede ser de 16 bits, 8 bits o 0 bit (inexistente).

La base y el índice son datos que se pueden modificar durante el procesamiento, ya que se encuentran en registros de propósito general de la CPU; no obstante, el desplazamiento es una cantidad fija que se define en el momento del ensamblado del programa y no puede cambiarse durante la ejecución.

mm y **aaa** hacen referencia al byte de direccionamiento

Tabla 10-1 - Modos de direccionamiento

Ejemplos de transferencia	Código de máquina	mm y aaa	Desplazamiento según el direccionamiento
MOV AX, [BX+SI]	8B00	00 000	(BX) + (SI)
MOV AX, [BX+DI]	8B01	00 001	(BX) + (DI)
MOV AX, [BP+SI]	8B02	00 010	(BP) + (SI)
MOV AX, [BP+DI]	8B03	00 011	(BP) + (DI)
MOV AX, [SI]	8B04	00 100	(SI)
MOV AX, [DI]	8B05	00 101	(DI)
MOV AX, [0000]	A10000	00 110	Dirección directa
MOV AX, [BX]	8B07	00 111	(BX)
MOV AX, [BX+SI+08]	8B4008	01 000	(BX) + (SI) + número 8 bits
MOV AX, [BX+DI+08]	8B4108	01 001	(BX) + (DI) + número 8 bits
MOV AX, [BP+SI+08]	8B4208	01 010	(BP) + (SI) + número 8 bits
MOV AX, [BP+DI+08]	8B4308	01 011	(BP) + (DI) + número 8 bits
MOV AX, [SI+08]	8B4408	01 100	(SI) + número 8 bits
MOV AX, [DI+08]	8B4508	01 101	(DI) + número 8 bits
MOV AX, [BP+08]	8B4608	01 110	(BP) + número 8 bits
MOV AX, [BX+08]	8B4708	01 111	(BX) + número 8 bits
MOV AX, [BX+SI+8000]	8B800080	10 000	(BX) + (SI) + número de 16 bits
MOV AX, [BX+DI+8000]	8B810080	10 001	(BX) + (DI) + número de 16 bits
MOV AX, [BP+SI+8000]	8B820080	10 010	(BP) + (SI) + número de 16 bits

Tabla 10-1 - Modos de direccionamiento (continuación)

10.4 Tipos válidos de instrucción

Como vimos, la instrucción *MOV* copiaba datos desde una "fuente" a un "destino"; el formato de la instrucción en "simbólico de máquina" (no debe confundirse con el formato de instrucción en código de máquina) es:

Los tipos de transferencias válidas son:

- Constante a registro *MOV AH, 1234h*
 - Constante a memoria *MOV [0200], 6789h*
 - Registro a registro *MOV AX, BX*
 - Registro a memoria *MOV [012A], AX*
 - Memoria a registro *MOV AX, [0203]*
 - Los operandos fuente y destino deben ser del mismo tamaño.
 - Las constantes y los registros de segmento no pueden moverse a registros de segmento.
 - Los operandos fuente y destino no pueden ser ambos posiciones de memoria.

Es importante que se busquen los tipos válidos de instrucción antes de programar *Assembler* en el "Manual de desarrollo de aplicaciones" del procesador en el que se va a ejecutar la aplicación.

10.5 Resumen

En teoría, el formato de la instrucción puede incluir, además del código de operación, una, dos, tres, cuatro o ninguna referencia a datos, ya sea que se alojen en memoria, en los registros de la CPU o sean simplemente los valores que se han de operar. La cantidad de campos de dirección que acompañan al código de operación también dependen de la organización interna de los registros de la CPU.

El juego de instrucciones de un procesador se puede dividir en los grupos siguientes: transferencia de datos, aritmética entera binaria, operaciones lógicas, desplazamientos y rotaciones, gestión de bits, aritmética codificada en binario (BCD), gestión de cadenas, aritmética de coma flotante, control del programa, control del sistema y multimedia.

Esta división en grupos se encuentra en los manuales referenciada con el término "clasi ficación de instrucciones" y se pueden utilizar denominaciones de grupo alternativas. Por ejemplo, las de control de programa también se conocen como instrucciones de salto o de bifurcación, las multimedia son instrucciones implementadas para el uso de aplicaciones de este tipo y en el Manual de Intel aparecen como instrucciones MMX, SSE y SSEII.

Por último, una modalidad de direccionamiento es una técnica que permite localizar un dato, o sea, definir la posición física donde se encuentra el dato que deberá ser operado por una instrucción.

10.6 Ejercicios propuestos

INSTRUCCIONES PRIVILEGIADAS Y DEL SISTEMA OPERATIVO:

Actividades pedagógicas en las que se utilizan las instrucciones referenciadas en el manual: "IA-32 Intel Architecture Software Developer's Manual Volumen 1 Basic Architecture", o en el link Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Reference Manual.

El objetivo de esta actividad es reconocer la diferencia entre las instrucciones del programador de aplicaciones y las instrucciones del sistema operativo. A continuación se describen las más utilizadas:

HLT - LGDT - SGDT - SYSEXIT - LLDT - LTR - STR - UD_T - SIDT - LMSW - SMSW - CLTS - ARPL - LAR - LSL - VERR - INVD - WBINVD - INVLPG - LOCK (prefijo) - RSM - RDMSR - WRMSR - RDPMC - RDTSC - SYSENTER

1. La instrucción LGDT carga una estructura de dato de 48 bits, en dos campos (2-field). Los primeros 16 son el campo de límite y los últimos 32 son el campo de base. Nombre una estructura que utilice LGDT. Entre las instrucciones enunciadas antes, mencione dos que estén directamente relacionadas con ella e indique para qué se utiliza, por ejemplo, UD_T.
2. La instrucción INVD invalida la información en las memorias caché internas sin escribir las actualizaciones que hayan sufrido; se utiliza cuando no interesa mantener la coherencia entre la información de la caché y la que se encuentra en la memoria principal. Indique cuál de las enunciadas es la instrucción ne-

cesaria para invalidar la caché con una previa actualización de memoria.

3. Exprese verbalmente el significado de cada título de campo de la tabla, para qué se utiliza la instrucción y cómo se interpreta cada elemento de la segunda fila:

Código de operación	Instrucción	Descripción
OF 02 /r	LAR r16,r/m16	r16 r/m16 masked by FPOOH

4. Luego de realizar estos ejercicios, investigue para qué se utilizan tres instrucciones elegidas al azar y exprese en pocas palabras qué diferencia encuentra entre este tipo de instrucciones y las que emplea una aplicación.

INSTRUCCIONES DE CONTROL DE SEÑALIZADORES:

Esta actividad tiene como objetivo mostrar que un programa de aplicación puede actualizar ciertas banderas del registro de estado según su conveniencia. Para responder las preguntas utilice las instrucciones presentadas a continuación:

STC – CLC – CMC – CLD – STD – LAHF – SAHF – PUSHF – PUSHFD – POPF – POPFD – STI – CLI

1. ¿Cuál es la instrucción que almacena el registro de señalizadores o registro de estado en la pila y cuándo es necesario que esto ocurra?
2. ¿Cuál es la instrucción que activa o inactiva la bandera de interrupciones?
3. ¿Cuál es la instrucción que activa o inactiva la bandera de acarreo?

INSTRUCCIONES ARITMÉTICAS BINARIAS

ADD - ADC SUB - SBB - IMUL — MUL - IDIV - DIV - INC - DEC - NEG - CMP

1. ¿Cuál es la diferencia entre una instrucción DIV y una IDIV respecto de su ejecución y de los registros de cálculo que utilice?
2. ¿Cuál es la instrucción que permite sumar dos operandos considerando un acarreo anterior?
3. ¿Qué tipo de complemento calcula la instrucción NEG aplicada a un operando?
4. ¿Qué diferencia hay entre la instrucción SUB y la CMP?

INSTRUCCIONES LÓGICAS

AND – OR – XOR – NOT

Para dos operandos “1F” y “65”, expresados en hexadecimal, explicar las operaciones en binario si se aplican sobre ellas las cuatro operaciones lógicas mencionadas.

INSTRUCCIONES DE BIT Y BYTE

BT – BTS – BTR – BTC – BCF – BCR

SETE/SETZ – SETNE/SETNZ – SETA/SETNBE – SETAE/SETNB/SETNC – SETB/SETNA/SETC – SETBE/SETNA – SETG/SETNLE – SETGE/SETNL – SETL/SETNGE – SETLE/SETNG – SETS
SETNS – SETO – SETNO – SETPE/SETP
SETPO/SETNP – TEST

1. Indique una aplicación que justifique el uso de la instrucción SETB.
2. Indique como mínimo cuatro ejemplos en Assembler que utilicen las variantes de la instrucción TEST.

INSTRUCCIONES DE CADENA

LODS/LODSB - LODS/LODSW - LODS/LODSD - STOS/STOSB - STOS/STOSW - STOS/STOSD - REP - SCAS/SCASW - SCAS/SCASD REPE/REPZ

REPNE/REPNEZ - INS/INSB - INS/INSW - INS/INSD - OUTS/OUTSB - SCAS/SCASB - MOVS/MOVSD - CMPS/CMPSB - MOVS/MOVSW - MOVS/MOVSB - OUTS/OUTSW - OUTS/OUTSD

Indique a qué instrucción pertenece cada descripción:

1. Copia los datos del operando fuente (segundo operando) al puerto de la entrada-salida, especificados con el operando destino (primer operando).
2. Después de cada secuencia CX se autodecremente y se comprueba la bandera indicadora de cero. Si no está “seteada”, se repite la ejecución de las instrucciones de la secuencia hasta que CX llegue a cero y se pueda salir del bucle.
3. Busca un byte en la dirección indicada por el registro ESI, lo almacena en la dirección indicada por el registro EDI, incrementa y decremente ambos en una unidad. Comprueba el valor de la bandera cero si la bandera “no está a uno” repite el procedimiento; caso contrario finaliza la copia del string.

INSTRUCCIONES “SHIFT AND ROTATE”

SAR – SHR – SAL/SHL – SHRD – SHLD – ROR – ROL – RCR – RCL

Representar gráficamente las instrucciones SAR SAL/SHL y ROR para la secuencia 11011001 en un registro de 8 bits.

INSTRUCCIONES ARITMÉTICAS DECIMALES

DAA – DAS – AAA – AAS – AAM – AAD

1. Indique un ejemplo de una suma BCD y aplique un ajuste a ASCII.
2. ¿En qué consiste el ajuste decimal luego de la suma?
3. De las instrucciones mencionadas, ¿qué diferencia hay entre la instrucción DAA y la AAA?

INSTRUCCIONES DE TRANSFERENCIA DE CONTROL

1. Suponga que el registro de estado tiene las banderas Signo, Cero, Sobreflugo y Acarreo en cero, suponga que la referencia a memoria que acompaña a cada uno de los mnemónicos descritos a continuación es 0100, que el valor actual de puntero de instrucción es 0202 y que el formato de cada una de las instrucciones en código de máquina es de 3 bytes. Para cada

una de las instrucciones descriptas. ¿Cuál es el valor del puntero luego de su ejecución si el valor de salto que acompaña a cada mnemónico es 0400?

JMP - JE/JZ - JNE/JNZ - JA/JNBE - JAE/JNB - JB/JNAE - JBE/JNA - JG/JNLE - JGE/JNL - JL/JNGE - JLE/JNG - JC - JNC - JO - JNO - JS - JNS - JPO/JNP - JPE/JP - JCXZ/JECXZ

2. Traduzca los siguientes mnemónicos:

LOOP - LOOPZ/LOOPE - LOOPNZ/LOOPNE -
CALL - RET - IRET - INT - INTO - BOUND -
ENTER - LEAVE

INSTRUCCIONES DE TRANSFERENCIA DE DATOS

MOV -CMOVE/CMOVZ - CMOVC - CMOVNO - CMOVNP/CMOVPE - CMOVNP/CMOVPO - XCHG - SWAP - XADD - CMPXCHG - PUSH - POP - PUSHF/PUSHAD - POPA/POPAD - IN/OUT - CWD/CDO CBW/CWDE - MOVSX MOVZX

1. Indique un ejemplo para cada una de las diferentes transferencias de la instrucción MOV y exprese el modo de direccionamiento apropiado.

Sintaxis: MOV Destino, Fuente

Destino	Fuente	Ejemplo	Modo de direccionamiento
Mem	Acum		
Acum	Mem		
Reg de Segm	Mem/Reg		
Mem/Reg	Reg de Segm		
Reg	Reg		
Reg	Mem		
Mem	Reg		
Reg	Dato inm		
Mem	Dato inm		

2. Para el siguiente set de instrucciones Assembler anote para cada una un comentario luego del punto y coma. Por último exprese verbalmente por qué ambos segmentos de programa son lógicamente iguales.

```
XOR EDX,EDX ;
ADD ECX, 4 ;
JNC sigo ;
MOV ECX,EDX ;

sigo
XOR EDX,EDX ;
ECX, 4 ;
CMOVC ECX,EDX ;
```

INSTRUCCIONES DEL SEGMENT REGISTER

LDS – LES – LFS – LGS – LSS

1. Traduzca cada uno de los mnemónicos, e indique en qué caso se utiliza LSS.

INSTRUCCIONES EN PUNTO FLOTANTE

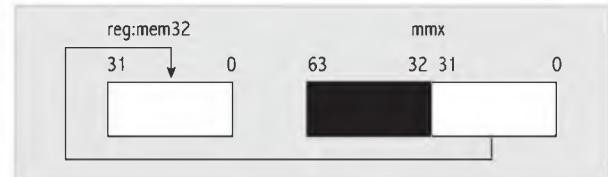
1. Averigüe los códigos de operación para cada una de las siguientes instrucciones.

Código de operación	Instrucción	Descripción: carga de constantes en el registro de pila de la FPU
	FLD1	Carga +1.0
	FLDL2T	Carga $\log_2 10$
	FLDL2E	Carga $\log_2 e$
	FLDPI	Carga π
	FLDLG2	Carga $\log_{10} 2$
	FLDLN2	Carga $\log_e 2$
	FLDZ	Carga +0.0

INSTRUCCIONES MMX

Instrucciones de transferencia de datos: MOVD – MOVO.

Explique el significado de la siguiente imagen, correspondiente a la ejecución de la instrucción MOVD.



Instrucciones de conversión: PACKSSWB - PACKSSDW - PACKUSWB - PUNPCKHWB - PUNPCKHWD - PUNPCKHDQ - PUNPCKLBW - PUNPCKLWD - PUNPCKLDD

Describa y grafe la transferencia de datos realizada por la instrucción PUNPCKHWB.

Instrucciones aritméticas packed: PADDB - PADDW - PADDD - PADDSD - PADDSW - PADDUSB - PADDUSW - PSUBB - PSUBW - PSUBSB - PSUBSW - PSUBUSB - PSUBUSW - PMULHW - PMULLW - PMADDWD

Describa y grafique la transferencia de datos realizada por la instrucción PSUBD.

Instrucciones de comparación: PCMPEQB – PCMPEQW – PCMPED – PCMPGTB – PCMPGTW – PCMPGTD

Describa y grafique la transferencia de datos realizada por la instrucción PCMPED.

Instrucciones lógicas: PAND – PANDN – POR – PXOR

Describa y grafique la transferencia de datos realizada por la instrucción PXOR.

Instrucciones de desplazamiento y rotación: PSLLW – PSLLD – PSLLQ – PSRLW – PSRLD – PSRLO – PSRAW – PSRAD

Indique qué registro/s puede/n intervenir en la ejecución de una instrucción PSLLW.

La instrucción de EMMS limpia el contexto MMX. Se debe utilizar en el final de una rutina MMX antes de llamar otras rutinas que puedan ejecutar instrucciones de punto flotante. Si no se hace, puede producirse un desbordamiento de la pila de punto flotante, lo que da lugar a una excepción o a un resultado incorrecto.

INSTRUCCIONES EN PUNTO FLOTANTE

Instrucciones de transferencia de datos: FLD – FST – FSTP – FILD – FIST – FISTP – FBLD – FBSTP – FXCH – FCMOVE – FCMOVNE – FCMOVB – FCMOVBE – FCMOVNB – FCMOVNBE – FCMOVU – FCMOVNU

Instrucciones de aritmética básica: FADD – FADDP – FIADD – FSUB – FSUBP – FISUB – FSUBR – FSUBRP – FISUBR – FMUL – FMULP – FIMUL – FDIV – FDIVP – FIDIV – FDIVR – FDIVRP – FIDIVR – FPREM – FPREM1 – FABS – FCBS

FRNDINT- FSCALE – FSQRT – FXTRACT

Instrucciones de comparación: FICOMP – FCOMI – FUCOMI – FCOMIP – FUCOMIP – FTST – FUCOMPP – FICOM – FCOM – FCOMP – FCOMPP – FUCOM – FUCOMP – FXAM

I. Realizar un programa Assembler que permita la suma de dos números en punto flotante en formato IEEE P754 de precisión simple, utilizando el algoritmo de Booth e instrucciones que sólo operen enteros. Luego representar el mismo programa con las instrucciones de coma flotante descriptas a continuación y clasificadas como instrucciones de transferencia de datos, de aritmética básica, de comparación, trascendentales y de asignación de constantes.

TRANSCENDENTAL INSTRUCTIONS

2. Indicar cuál de estas instrucciones calcula el seno, el coseno y la tangente, y expresar en cada caso su sintaxis.

FSIN – FCOS – FSINCOS – FPTAN – FPATAN – F2XM1 – FYL2X – FYL2XP1

Load constant instructions: FLD1 – FLDZ – FLDPI – FLDL2E – FLDLN2 – FLDL2T – FLDLG2

MODOS DE DIRECCIONAMIENTO

1. Complete la columna "Descripción".

Modo	Descripción
Inmediato	
Directo a Registro	
Directo a Memoria	
Indirecto a Registro	
Relativo a la Base	
Relativo a la Base indexado	

2. Para cada uno de los siguientes modos de direccionamiento dé un "ejemplo del lector" concreto, diferente al de la primera columna.

Ejemplo	Ejemplo del lector
mov ax,09h	
mov ax,bx	
mov ax,var	
mov ax,[bx]	
mov ax,[bx+4]	
mov ax,[bx+si+4]	

3. Para cada uno de los modos enunciados antes represente un esquema que los identifique, por ejemplo:

MOV AX,BX

BX 0111001100001111

AX 0111001100001111

10.7 Contenido de la página Web de apoyo

El material marcado con asterisco (*) sólo está disponible para docentes.

Resumen gráfico del capítulo

Autoevaluación

Video explicativo (02:57 minutos aprox.). Capítulos 10 y 11

Audio explicativo (02:57 minutos aprox.). Capítulos 10 y 11

Evaluaciones Propuestas*

Presentaciones*

11

Software del sistema

Contenido

11.1 Introducción	278
11.2 Clasificación del software de sistema	278
11.3 Sistema operativo	278
11.4 Traductores de lenguaje	282
11.5 Resumen	285
11.6 Ejercicios propuestos	286
11.7 Contenido de la página Web de apoyo	288

Objetivos

- Conocer los conceptos básicos relacionados con el software de base.
- Reconocer la relación entre la instrucción software y la capacidad del hardware para decodificarla y lograr su ejecución
- Reconocer, por medio de un lenguaje de programación, la relación lenguaje simbólico y lenguaje de máquina.



En la página Web de apoyo encontrará un breve comentario de la autora sobre este capítulo.

11.1 Introducción

En este capítulo se introducen los primeros conceptos relacionados con el software de sistema. Con el único fin de que se interprete su mención en esta obra, se comentan las actividades comunes de los sistemas operativos organizadas por nivel de administración. En el caso de los procesos, interesa conocer sus cinco estados básicos y la conmutación de uno a otro, para comprender, por ejemplo, qué ocurre cuando un proceso recibe una interrupción o para justificar el uso de la pila.

Además, la administración de memoria, tanto segmentada como paginada, en modo virtual o no, requiere hardware de sustentación, por ejemplo, la unidad de traducción de una dirección virtual a una dirección física. Estos componentes funcionales diseñados en hardware justifican la breve descripción que haremos a continuación acerca del software de sistema. Se destacan los problemas propuestos que permiten relacionar el código de máquina con el lenguaje *Assembler* y los lenguajes de alto nivel.

11.2 Clasificación del software de sistema

El software del sistema es el nexo entre las necesidades del usuario y las capacidades del hardware. Se considera que está integrado por los siguientes componentes:

- Software de base
- Software de comunicaciones
- Software de administración de bases de datos

El software de base controla y respalda en cierto modo el software de las otras categorías, y todas ellas están íntimamente relacionadas en mayor o menor grado con el diseño del hardware, lo que las hace aptas para una computadora y sus compatibles y no para otros. El núcleo del software de base se denomina **sistema operativo**; sus componentes supervisan y controlan la actividad de los recursos físicos (hardware) y los recursos lógicos (usuarios, procesos, archivos, actividades de entrada/salida).

También forma parte del software de base la interfaz gráfica de usuario (GUI o *Graphical User Interface*), cuyo objetivo principal es crear un entorno organizado para el usuario y los utilitarios o utilidades.

Cuando un programador desarrolla un algoritmo para la resolución de un problema con una computadora, éste se debe expresar en términos de las reglas sintácticas y semánticas de un lenguaje de programación. El programador debe atenerse a las reglas de codificación del lenguaje; luego se requiere un procesamiento explícito o no de un programa traductor, por lo general un compilador.

Cada uno de los pasos elementales del algoritmo se representa por medio de una sentencia. Un conjunto de sentencias constituye un programa fuente, que al traducirlo genera un programa ejecutable para el procesador de esa computadora. El conjunto de estos programas conforma el software de aplicación, que utiliza como soporte el entorno que crea el software de base para ejecutarse.

11.3 Sistema operativo

En el diccionario de la Real Academia Española se encontraron varias opciones para definir el verbo administrar. Todas son compatibles con la función del sistema operativo:

- Suministrar, proporcionar o distribuir.



El sistema operativo administra los recursos, siempre limitados, del sistema.

- Graduar o dosificar el uso de algo, para obtener mayor rendimiento de ello o para que produzca mejor efecto.
- Ordenar, disponer, organizar.
- Gobernar, ejercer la autoridad o el mando.

Un sistema operativo es una colección de programas que administran la operación de una (o varias) computadoras, con el fin de obtener un comportamiento eficiente. Es una plataforma software que asigna recursos y supervisa al resto de los programas que se ejecutan en la computadora. El sistema operativo no realiza por sí solo ninguna función que genere resultados para las aplicaciones del usuario, sino que es un medio para que estos programas los obtengan. La asignación de recursos, de hardware o software, que se requieren durante la ejecución puede diferenciarse a partir de las siguientes funciones del sistema operativo:

- Control de tiempos de ejecución en el procesador y sincronización de los procesos.
- Asignación de espacio de memoria requerido para cada proceso.
- Asignación de espacio de almacenamiento y recuperación de archivos (en memorias auxiliares).
- Interfaz con los manejadores de dispositivos de E/S.
- Contabilidad del tiempo de uso
- Control y recuperación de errores.
- Protección en el uso de recursos.
- Comunicación con el usuario.
- Prevención de errores en el mal uso del sistema.

Su objetivo principal, al ejecutar aplicativos, es crearles un entorno organizado, abastecer sus requerimientos y solucionar los problemas que surjan durante la ejecución.

Los programas que constituyen el sistema operativo son desarrollados por fabricantes de la industria del software y proporcionados a los compradores. Están diseñados para hacer el mejor uso de los componentes de cada sistema de computación. Pueden existir diferentes sistemas operativos pertenecientes al mismo fabricante. En cuanto al usuario, le ofrece comodidad, ya que lo libera de programar tareas rutinarias, y mejora la eficiencia del sistema, sobre todo en lo que se refiere a tratar de minimizar el tiempo ocioso de la CPU.

Se puede afirmar que hay dos grandes componentes software del sistema operativo:

Residentes: también llamados supervisores. Son componentes que residen de manera permanente en la memoria principal durante todo el procesamiento.

Transitorios: residen sólo cuando se los necesita y están almacenados en memorias secundarias cuando no están en la memoria principal.

11.3.1 Niveles de administración del sistema operativo

11.3.1.1 Administración del procesador y los procesos

Un proceso es un programa, o parte de él, en estado de ejecución, o sea que está en alguna de las etapas de su ciclo de vida. Está constituido por el código ejecutable, los datos con los que va a operar, el estado de los registros en la CPU y su propio estado respecto de otros procesos.



Se denomina software de aplicación al conjunto de programas que atienden los problemas específicos del usuario. Es concebido o creado por el programador en una empresa o por un fabricante; por ejemplo, el procesador de texto que se utiliza para escribir este capítulo fue diseñado por una empresa que vende productos de software denominados aplicaciones o aplicativos y que sirven para atender trabajos específicos del usuario.



Se espera que el diseño de un sistema operativo maximice la eficiencia global del procesamiento en la computadora. Debe supervisar todas las actividades, ejecutar programas especiales de sistema cuando sea necesario, asignar nuevos recursos y planificar los trabajos para la actividad continua del sistema.

El despachador es el módulo de sistema operativo que asigna el proceso al procesador, pertenece al nivel de administración más cercano al proceso. Es el que permite la creación de un nuevo proceso aceptando la solicitud de otro o de un usuario. La creación de un proceso implica una serie de actividades que arranca con el reconocimiento del proceso por parte del sistema operativo, que verifica que haya recursos suficientes para su ejecución; por ejemplo, el proceso requiere memoria principal para sus instrucciones y datos. En esta etapa le asigna una identificación, un estado, una prioridad y demás elementos para realizar su seguimiento. En consecuencia, el despachador es el que administra el contexto asociado a un proceso para que pase por los distintos estados de su ciclo de vida. A continuación describiremos cinco etapas o estados en los que evoluciona hasta finalizar y salir del sistema.

- Estado nuevo. Se puede decir que un programa ejecutable "adquiere vida" cuando se le crea un entorno adecuado para comenzar su ciclo de vida. Ciclo que finaliza cuando termina su ejecución de manera normal o anormal.
- Estado listo o preparado. Es cuando un proceso tiene recursos asignados suficientes para que la CPU comience o continúe su ejecución, pero no tiene la asignación del procesador.
- Estado de ejecución de un proceso. Se denomina así al estado en el que se está haciendo uso de la CPU.
- Estado en espera. Es aquel en el que el proceso no puede continuar su ejecución porque le falta algún recurso que se le proveerá luego.
- Estado parado. Es aquel en el que el proceso terminó su ejecución y se requiere un tiempo en el que el sistema operativo libere los recursos que le había asignado y destruya toda la información de contexto del proceso para eliminarlo del sistema.

Cuando el proceso P1 en ejecución queda en espera porque, por ejemplo, necesita una operación de E/S, el sistema operativo posibilita que otro proceso P2, que está en estado listo pase a estado de ejecución. Cuando P1 se pone en lista de espera y P2 se ejecuta, se produce la intervención del *dispatcher*. Al finalizar la operación de E/S, P1 podrá pasar de nuevo a estado listo.

11.3.1.2 Administración de memoria

El sistema operativo asigna la memoria que un proceso requiere para su ejecución. También administra la memoria virtual en los sistemas operativos que cuenten con esta facilidad. El concepto de memoria virtual se sigue utilizando en sistemas operativos actuales. Recuerde que si un programa, debido a su tamaño, no puede acomodarse en forma completa en la memoria principal, el sistema operativo puede fragmentarlo y facilitar el intercambio de fragmentos entre memoria principal y memoria de disco. Esto le permite al sistema ampliar la memoria principal utilizando el disco como una extensión de ella.

El control de programas en estado de ejecución fija límites y restricciones entre procesos como, por ejemplo, a qué partes de la memoria puede acceder un programa en ejecución y cuáles no.

11.3.1.3 Administración de dispositivos de entrada/salida

Un dispositivo de entrada/salida requiere comandos o instrucciones específicas. Por ejemplo, un disco debe comenzar a girar antes de que se acceda a él para su lectura o escritura. El lenguaje de comandos del disco es particular y distinto del de la impresora; además, un dis-

positivo genera información de su propio estado. Todos estos detalles deben quedar aislados de las aplicaciones, de modo que el acceso a cada uno de ellos "sea visto" como una simple operación de lectura o escritura.

Encapsular los detalles propios de cada dispositivo es tarea de los manejadores de dispositivo o *drivers*. De este modo, un fabricante de dispositivo escribe su propio *driver* para que sirva de software para el conjunto de sistemas operativos más usados, por lo que el diseñador no requiere desarrollar una nueva versión de cada sistema operativo en cuestión.

La programación de funciones estandarizadas agrupando dispositivos por clases generales corresponde al diseño de un sistema operativo; así se crea una capa de software que actúa de interfaz entre el proceso y la clase de dispositivo. Cuando un proceso requiere una entrada/salida produce una "llamada al sistema" (*system call*) que convoca a la función estándar. Dentro de los servicios brindados por el sistema operativo, debemos mencionar las llamadas al sistema (*system calls*), en este caso de "entrada/salida". Un proceso en curso suspende su ejecución para solicitar un servicio del sistema operativo por medio de estas instrucciones "especiales", que tienen un nivel de privilegio mayor que las restantes y se conocen como primitivas.

Por ejemplo, cuando un proceso de usuario quiere grabar un registro en un archivo en disco, la primitiva invoca la función estándar creada. Esta rutina requiere que el proceso "le pase" parámetros. Si en el disco se detecta una falla que provoca que la grabación no sea posible, la función recibe como parámetro un indicativo del error. Si el error no se produce después de que el sistema operativo ejecutó la llamada al sistema, el proceso puede continuar su ejecución. Por ejemplo, en *GNU/LINUX* la función *ioctl()* permite que el núcleo del sistema operativo se comunique con un *driver*. Esta función requiere un primer argumento que identifique el *driver*, un segundo argumento que indique cuál es la operación y un tercer argumento que establezca dónde queda el resultado de la operación.

11.3.1.4 Administración de archivos

Supervisa la gestión de archivos para su creación, acceso y eliminación. Define la política que determina de qué forma serán almacenados físicamente. Determina la asignación de un archivo a una aplicación, y si corresponde de acuerdo con los derechos de acceso. Informa de su disponibilidad cuando el sistema permite que varias aplicaciones puedan acceder a archivos de manera alternada.

11.3.2 Tipos de sistemas operativos

11.3.2.1 Multitarea y tiempo compartido

Los sistemas operativos de tiempo compartido tratan de administrar los recursos repartiéndolos de manera equitativa. Los sistemas de multitarea son capaces de administrar procesos concurrentes y permiten que tanto las instrucciones como los datos procedentes de varios procesos residan al mismo tiempo en la memoria principal y eventualmente en el disco. Los *task* o tareas activas compiten de manera simultánea por los recursos del sistema en forma alternada.

11.3.2.2 Multiusuario

Los sistemas operativos multiusuario permiten el acceso de varios usuarios desde distintas terminales administradas por el mismo sistema operativo.

11.3.2.3 Tiempo real

Estos sistemas tienen como objetivo proporcionar tiempos más rápidos de respuesta, pues se utilizan para brindar soporte a sistemas como el control de transportes, el control de pro-

cesos industriales, los cajeros automáticos, por dar algunos ejemplos. La característica más importante de estos sistemas es que sus acciones se deben ejecutar en intervalos de tiempo determinados por la dinámica de los sistemas físicos que supervisan o controlan; además, deben ser sistemas rigurosos en cuanto a la integridad de la información, asegurar un servicio sin interrupciones y que soporte políticas de seguridad eficientes.

11.4 Traductores de lenguaje

Son programas cuya función es convertir los programas escritos por el usuario en lenguaje simbólico a lenguaje de máquina. Un programa escrito en lenguaje simbólico se denomina fuente, mientras que un programa en lenguaje de máquina se denomina ejecutable. Las instrucciones del programa fuente constituyen la entrada al traductor de lenguaje, la salida serán las instrucciones del programa ejecutable. La traducción comprende el análisis del léxico y la sintaxis de cada instrucción o sentencia, así como las referencias lógicas a las que apunta cada instrucción. Si del análisis surgieran errores, el traductor generará un informe donde indicará el lugar donde se produjo y cuál es el tipo de error cometido. Para realizar el análisis sintáctico, el programa traductor controla cada sentencia del programa fuente (nombres de datos, signos de puntuación y palabras propias del lenguaje para que tengan una construcción aceptada por el traductor). Se necesita un traductor para cada lenguaje simbólico que se utilice. Se destacan tres tipos de traductores de lenguaje: ensambladores, intérpretes y compiladores.

11.4.1 Ensambladores

El término ensamblador (*assembler*) se refiere a un tipo de software traductor que se encarga de traducir un archivo fuente escrito en un lenguaje Assembler a un archivo cuyas instrucciones estén en código máquina, que es ejecutable directamente por el procesador para el que se creó.

La razón por la que se creó este tipo de traductor es "humanizar", por medio de un lenguaje simbólico, la escritura de programas a nivel código de máquina, que es el código binario "decodificable" por el procesador pero imposible de escribir en la práctica. Así, el código *8E26* se representa en Assembler *x86* como *MOV BH* en el Assembler, donde *MOV* es mover y *BH* es el nombre de un registro de CPU.

Este tipo de traductor evolucionó como todos los lenguajes de programación y en la actualidad dejó de llamarse ensamblador para identificarse como compilador *Assembler*.

Existe la creencia errónea de que el Assembler es un lenguaje "antiguo"; esto se debe a que la mayoría de los programadores trabaja con lenguajes más evolucionados, como aquellos muy utilizados en el presente que son los visuales. Sin embargo, a la hora de programar a nivel hardware, la única posibilidad la ofrece el set de instrucciones Assembler del procesador que se esté utilizando. Todo procesador, grande o pequeño, desde el de una computadora personal hasta el de una supercomputadora, posee su lenguaje máquina y, por lo tanto, su simbólico de máquina.

Los traductores se dividen en dos grupos en función de la relación entre lenguaje fuente y lenguaje de máquina. El primer grupo traduce una instrucción de un lenguaje fuente y genera una única instrucción de máquina; ese lenguaje fuente es su Assembler. El segundo grupo lo constituyen los lenguajes de alto nivel en los que una sentencia se traduce a varias instrucciones en código de máquina.

Como en todo Assembler hay en mayor medida una correspondencia 1 a 1 entre instrucciones simbólicas e instrucciones de máquina y, por ello, se puede realizar la traducción inversa, denominada *desensamblaje*.

Asimismo, todo lenguaje tiene un conjunto de reglas que el programador debe respetar. Un programa Assembler está organizado con una secuencia lógica de líneas de programa; para dar un ejemplo, observemos el significado de cada campo en una línea Assembler 80x86 así:

SUM ADD AX,[0203] / En esta línea se suma el dato de memoria al contenido de AX.

Esta instrucción se divide básicamente en tres campos. El primero es el rótulo y es opcional; especifica una referencia simbólica a la línea. Está formado por una secuencia de caracteres alfanuméricos, en general corta, en la que el primer carácter siempre es una letra. El ensamblador reconoce los primeros caracteres como rótulo hasta que encuentre un blanco (espacio) o coma. En el segundo campo se simboliza una instrucción de máquina o una seudoinstrucción (una seudoinstrucción no representa una instrucción de máquina sino que son órdenes al ensamblador). El verbo se define con un mnemónico de 2 o 3 letras, un blanco o espacio, un operando destino, luego una coma y el operando origen; en este ejemplo la referencia al dato implicado se representa en hexadecimal y entre corchetes. También puede usarse el nombre simbólico del dato, por ejemplo, *VAR1*. El tercer campo está separado por una barra oblicua y el ensamblador no lo traduce porque se lo considera campo de comentario. La traducción de esta instrucción a código de máquina es *A10302*.

Cuando el programa es complejo suele dividirse en módulos que se programan en forma independiente, y al momento de traducción pueden estar alojados en zonas diferentes de memoria, que se ensamblan y se ponen a punto de manera independiente del proceso de traducción. Es necesario juntar los módulos, de manera de organizar un solo programa ejecutable, donde las instrucciones se encuentren una detrás de otra. Una solución a este problema es ensamblar los módulos como si cada uno tuviese un origen *0* y generar una tabla con las direcciones absolutas, de manera que, una vez conocido el origen real del programa, sólo se tenga que desplazar el valor del origen en las posiciones indicadas en la tabla de direcciones absolutas.

El ensamblador permite generar tanto el código absoluto como el que permite reubicar los módulos. Asimismo, lee dos veces el programa fuente, o sea, hace dos pasadas. En la primera pasada guarda todos los nombres simbólicos en una tabla de nombres simbólicos y la completa con las referencias a memoria correspondientes para cada uno; así, en la segunda pasada ya se conocen los valores de todos los nombres simbólicos, con lo que soluciona el problema de las referencias adelantadas.

La seudoinstrucción *ORG* permite iniciar el programa generando el código absoluto a partir de la referencia indicada en ella, por ejemplo, *ORG 100*. Si ésta no se especifica, el ensamblador asume que el origen es cero y genera la tabla de direcciones absolutas. Además del problema de reubicación de módulos a cargo del ensamblador, existe otro que se produce cuando un módulo necesita acceder a datos o instrucciones contenidas en otro módulo, que seguramente no serán localizados debido a que no se completó la tabla de nombres simbólicos. Para realizarlo hay dos seudoinstrucciones: *export* e *import*. *Export* contiene todos los nombres simbólicos de un módulo que serán referenciados por otro; por ejemplo, título *1* (subrutina de título) pertenece al módulo *1*. Por lo tanto, en éste deberá aparecer una seudoinstrucción que indique "*export título 1*", que permite que otros módulos la utilicen. *Import* debe indicar la referencia externa dentro del módulo que la quiere utilizar, por lo tanto, si un módulo *3* pide la ejecución de la subrutina de título, en este último módulo deberá aparecer: "*import título 1*". Para el manejo de estas referencias externas el ensamblador confeccionará dos tablas, la tabla de nombres simbólicos importados y la tabla de nombres simbólicos exportados. Éstas contendrán el nombre simbólico y la dirección relativa en el módulo en caso de importación, o el valor de la dirección o parámetro en el caso de exportación.

El programa enlazador (*linker*) es el encargado de relacionar (en inglés, *link* es unir) los distintos módulos en un solo programa. Primero produce la reubicación de módulos, a partir de determinar la dirección de comienzo e incrementarla con la TDA o con la TNSE, y la determinación de las referencias externas, para lo que requiere del usuario los nombres de los módulos y las posiciones absolutas que éste desea. En definitiva, la función del enlazador es unir los módulos que ya fueron traducidos por el ensamblador para que constituyan una unidad. Por último, lo producido en esta etapa se almacena como archivo ejecutable en una memoria no volátil, como el disco.

Una macro Assembler es un grupo de instrucciones definidas con anterioridad que se puede incorporar en un nuevo programa durante su proceso de ensamblado. En el programa se la referencia con una seudoinstrucción que le avisa al ensamblador dónde debe ubicar la macro y reemplazar la seudoinstrucción por ella.

La diferencia entre una macro y una subrutina es que cuando se utiliza una macro el ensamblador repite la secuencia de instrucciones en el programa tantas veces como sea llamada la macro; en cambio, cuando se llama a una subrutina se provoca una ruptura de secuencia en el programa principal, se ejecuta la subrutina y luego se retorna al programa principal a partir de la instrucción siguiente al llamado.

11.4.2 Intérpretes

Es un traductor de lenguaje que traduce una instrucción en lenguaje de alto nivel a lenguaje de máquina y, de ser correcta, la ejecuta inmediatamente. Si encuentra un error de sintaxis, lo señala e interrumpe la ejecución. La ventaja de la traducción con intérpretes es que el programa se va probando a medida que se confecciona, o sea que permite una programación "interactiva". Su desventaja es que debe traducirse cada vez que se quiere ejecutar, aun cuando no haya sufrido modificaciones. La interpretación es una variante, que en primer lugar requiere que el lenguaje se adapte a esta forma de traducción.

11.4.3 Compiladores

Es un traductor de lenguaje que traduce un programa escrito en lenguaje de alto nivel a lenguaje de máquina, pero tiene algunas diferencias significativas respecto del intérprete. Fundamentalmente, separa la traducción de la ejecución del programa y agiliza tanto una como otra. La ejecución del programa sólo se realiza cuando la compilación terminó de manera satisfactoria. Permite obtener el código de máquina del programa compilado y hacer un resguardo del mismo en una memoria externa. Luego se puede convocar al programa ejecutable tantas veces como sea necesario, con la única condición de que se lo transfiera desde la memoria externa hacia la memoria principal. La compilación también proporciona un informe del programa fuente con los errores. La relación entre las instrucciones de alto nivel y las de máquina son 1 a n (siendo $n > 1$), motivo por el cual las instrucciones en lenguaje de alto nivel suelen denominarse sentencias.

Ejemplo:

Para la siguiente sentencia Pascal $C := A + B$ el compilador generará el código de máquina que en Assembler de nuestra computadora "X" equivale a

LDA, A

ADA, B

STA, C

El proceso de compilación de un programa puede interpretarse en tres etapas:

- Análisis del léxico: se separa la cadena de caracteres en símbolos elementales, como se especificó antes en nombres de datos, signos de puntuación y palabras especiales, también llamadas palabras reservadas.
- Análisis sintáctico: se determina la estructura sintáctica de acuerdo con un patrón de reglas gramaticales correspondiente al lenguaje que se utilice, y se analiza la secuencia de señales generadas en el proceso de análisis de léxico.
- Generación de código: se genera el código de las instrucciones para cada elemento sintáctico del programa.

Una cuarta etapa puede ser la de optimización, cuyo objetivo es reducir el programa o hacerlo más veloz, utilizando técnicas como detección y eliminación de instrucciones redundantes y uso de registros asociados a la CPU en vez de palabras de memoria, siempre que sea posible.

11.5 Resumen

En un sistema de procesamiento de datos, en el que se utiliza una computadora para realizar el procesamiento, hay dos extremos en ámbitos incompatibles por completo: en uno se encuentra el usuario del sistema, que no quiere ni necesita conocer los detalles del otro extremo, el hardware del sistema. En el medio, y actuando como interfaz, se encuentra el software. En cuanto a éste, toda tarea específica que necesite ser descripta por el usuario al sector de desarrollo se denomina software de aplicación; en cambio, toda tarea que sea necesaria para la administración del sistema y la gestión organizada de actividades corresponde al software de sistema. En la mayoría de los libros de teoría de sistemas operativos se lo define como un administrador de recursos físicos (como la memoria o el procesador) y un administrador de recursos lógicos (como los usuarios, los archivos o los procesos).

Sin embargo, en lo que a esta obra se refiere, nos centramos en las herramientas que permiten la traducción de los lenguajes de alto nivel al lenguaje de máquina y es por eso que los problemas propuestos sólo se relacionan con estas herramientas. Las etapas en la creación de un programa son básicamente tres: edición del programa fuente en lenguaje de alto nivel o de bajo nivel, creación del módulo ejecutable o módulo objeto y enlace, que inserta rutinas que se intercalan en el módulo objeto para así generar el módulo ejecutable (p. ej., en lenguaje "C" estas rutinas son las librerías, y, por nombrar alguna de ellas, podemos mencionar "stdio.h", que significa *Standard Input Output Header*).

11.6 Ejercicios propuestos

- I) El programa siguiente muestra en pantalla la hora con el formato HH:MM:SS que sigue, centrado y sin cursor
- Realice todas las etapas hasta llegar a su ejecución.
 - Editelo en un archivo de texto.
 - Ejecute el ensamblador.
 - Ejecute el "linkeditor".
 - Ejecute el archivo para ver cómo funciona.

```

DATOS SEGMENT
    JOYA DB 'ES LA HORA='
    HORA DW ?
    DB ?
    MINUTO DW ?
    DB ?
    SEGUNDO DW ?
    DB '$'
DATOS ENDS

PILA SEGMENT STACK
    DB 100 DUP (?)
PILA ENDS

PRO SEGMENT
    ASSUME CS:PRO,DS:DATOS,SS:PILA

INICIO: MOV AX,DATOS
        MOV DS,AX
        MOV AX,PILA
        MOV SS,AX
        MOV AX,00H
        INT 10H ;VIDEO MODE
        MOV AH,02
        MOV DH,12 ;FILA
        MOV DL,26 ;COLUMN
        INT 10H ;SET CURSOR POSITION
        MOV AH,01H
        MOV CH,4
        MOV CL,3 ;BORRA EL CURSOR
        INT 10H
        MOVAH,2CH
        INT 21H ;CH=HORA CL=MIN DH=SEG
        DL=CENTECIMAS

```

```

MOV BL,0AH ;CTE PARA % 10
MOV AH,00H
MOVAL,CH ;CH=HORA EN HEXA
DIV BL ;% 10 AL=RESULTADO AH=RESTO
ADD AX,3030H ;SE CONVIERTEN ASCII
MOV[HORA],AX ;GUARDO

```

```

MOV BL,0AH
MOV AH,00H
MOVAL,CL
DIV BL
ADD AX,3030H
MOV [MINUTO],AX

```

```

MOV BL,0AH
MOV AH,00H
MOVAL,DH
DIV BL
ADD AX,3030H
MOV [SEGUNDO],AX

```

```

MOV DX,OFFSET JOYA ;DX APUNTA AL
STRING JOYA
MOV AH,09H
INT 21H

```

```

MOV AH,00H
INT 1AH
ADD DX,100H
MOV BX,DX
AHI: INT 1AH
CMP DX,BX
JNE AHI

```

```

MOV AH,01H
MOV CH,06H
MOV CL,07H
INT 10H

```

```

MOV AX,4C00H ;FIN
INT 21H
PRO ENDS
END INICIO

```

2) Indique para qué sirven las seudoinstrucciones:

- a. DATOS segment y DATOS ends
- b. PILA segment y PILA ende
- c. PRO segment y PRO ende

LEA SI,BUFFER

*SIGUE: MOVAH,01H;
INT 21H*

3) Indique el significado de las palabras siguientes reservadas en Assembler

- a. DB, DW, DD
- b. ASSUME CS:PRO, DS:DATOS,SS:PILA

*MOV BYTE PTR DS:[SI],AL
INC SI
DEC CX
JNZ SIGUE*

c. INICIO: y END INICIO

*MOVAH,01H
MOV DX,00H
INT 17H*

d. Indique cuál es la función de cada párrafo separado por una línea en blanco del código descripto desde INICIO hasta END INICIO

LEA SI,BUFFER

*MOV CX,10
MOVAH,0H*

MOV DX,0H

4) Realice el mismo programa en un lenguaje de alto nivel que usted conozca, cubriendo las etapas de edición, compilación y ejecución.

OTRO: MOVAL,DS:[SI]

*INT 17H
INC SI
LOOP OTRO
MOVAH,4CH
INT 21H
EJEMPLO ENDP
CODIGO ENDS
END EJEMPLO*

5) El programa siguiente despliega las teclas presionadas por el usuario; es probable que si lo ejecuta lo despliegue en su monitor.

- a. Indique en qué párrafo se completa la carga del buffer de teclas ingresadas.
- b. Indique en qué párrafo se inicializa la impresora.
- c. Indique en qué párrafo se completa el despliegue o la impresión de las teclas ingresadas.

DATOS SEGMENT

*ERROR DB 'LA IMPRESORA NO FUE
SELECCIONADA, NO TIENE PAPEL O ESTA
APAGADA '\$'*

BUFFER DB 10 DUP (0)

DATOS ENDS

PILA SEGMENT STACK

DB 128 DUP (?)

PILA ENDS

CODIGO SEGMENT

*ASSUME CS:CODIGO,DS:DATOS,ES:DATOS,SS
:PILA*

EJEMPLO PROC

MOV AX,DATOS

MOV DS,AX

MOV CX,10H



11.7 Contenido de la página Web de apoyo

El material marcado con asterisco (*) sólo está disponible para docentes.

Resumen gráfico del capítulo

Autoevaluación

Video explicativo (02:57 minutos aprox.). Capítulos 10 y 11

Audio explicativo (02:57 minutos aprox.). Capítulos 10 y 11

Lecturas adicionales:

El proceso de compilación de Gustavo López, Ismael Jeder y Augusto Vega, es parte del libro "Análisis y Diseño de Algoritmos", de Alfaomega Grupo Editor (16 páginas). Agradecemos a sus autores por permitir que su escrito sea parte de las lecturas complementarias de esta obra.

Evaluaciones Propuestas*

Presentaciones*

12

Dispositivos de entrada/salida

Contenido

12.1 Introducción	290
12.2 Discos rígidos.....	290
12.3 Dispositivos de almacenamiento removibles	294
12.4 Resumen.....	297
12.5 Ejercicios propuestos	298
12.6 Contenido de la página Web de apoyo.....	298

Objetivos

- Conocer distintos componentes físicos y especificaciones técnicas para dispositivos de almacenamiento masivo.



En la página Web de apoyo encontrará un breve comentario de la autora sobre este capítulo.

12.1 Introducción

Este capítulo permite conocer las prestaciones de las memorias de almacenamiento secundario, las que por su característica de no volatilidad y bajo costo por bit (respecto de las memorias de semiconductores), se utilizan como medios de almacenamiento masivo. Si bien las memorias de semiconductores son las más rápidas para el acceso desde la CPU, su capacidad de almacenamiento es relativamente pequeña respecto de, por ejemplo, la capacidad de un disco magnético. Si consideramos que una *notebook* almacena 120 veces más información en memoria de disco que en memoria principal, la diferencia es apreciable.

Las memorias secundarias almacenan programas ejecutables y no ejecutables, base de datos, archivos de imágenes, de video y toda información digital que se quiera almacenar en forma permanente. La CPU intercambia instrucciones y datos con distintos periféricos, como la unidad de disco rígido, una tarjeta de memoria o una unidad de DVD. Estos periféricos se denominan dispositivos de entrada/salida y están conectados a la CPU en la mayoría de las computadoras, con un bus individual denominado bus de entrada/salida (*input/output bus*), que actúa de colector auxiliar desde el bus de sistema y hacia el mismo. La CPU, extrae instrucciones para ejecutar y datos para operar de la memoria de semiconductores, cuyas celdas de bits constituyen el soporte de información binaria a corto plazo; esto implica que la versión original de un programa ejecutable o de los datos fueron trasladados allí desde un almacén masivo.

De la misma manera, los resultados obtenidos se almacenan primero, y en forma temporal, en la memoria principal (memoria de semiconductores). Para almacenar los resultados obtenidos por la CPU en forma permanente se actualizan archivos, que se graban en medios magnéticos u ópticos. Existe una variedad de opciones de almacenamiento masivo, cada una con características que las hacen más apropiadas a un uso particular; por ejemplo, nadie duda de la gran ventaja que presentan las tarjetas de memoria cuando se habla de almacenamiento portable y la gran ventaja que presentan los discos rígidos a la hora de grandes volúmenes de información de acceso rápido. Las computadoras permiten el acceso a muchos de estos dispositivos y es por eso que los describiremos a continuación; por supuesto, comenzaremos por los de mayor prestación en el uso habitual, los discos rígidos.

12.2 Discos rígidos

Son los dispositivos de entrada/salida de mayor uso en nuestros días. En el capítulo Memorias ya describimos algunas características físicas y lógicas de ellos, debido a la necesidad de utilizar esos conceptos para la comprensión de la organización de archivos almacenados allí. Es por eso que aquello que mencionamos y no forma parte de lo físico, pero que sea necesario para la comprensión de este capítulo, se enunciará brevemente a los efectos de no tener que cambiar de sección en el libro. A continuación, se desarrolla un breve resumen.

Los discos magnéticos pueden ser rígidos, dispuestos en unidades de cabezas fijas o móviles, en grupos de uno o más platos. Más allá de todos estos detalles, debemos considerar que un disco está constituido por una base, recubierta de material magnetizable con forma de circunferencia, que gira alrededor de un eje dispuesto en la unidad, y que es accedido por una o más cabezas lectograbadoras. Una cabeza registra los bits en círculos concéntricos denominados pistas. Una pista es una división lógica y no física de la superficie, producto de la acción de rotación del soporte y de la posición fija de la cabeza al momento de grabación o lectura. La cantidad de pistas en la superficie depende del tipo de unidad a la que está asociado el soporte y del sistema operativo que lo gestione. Si la unidad graba las dos caras de un plato o cuenta con más de uno, o se presentan ambas situaciones, habrá muchas pistas de igual número (e igual distancia respecto del eje) en las distintas caras. Podemos imaginarnos que por efecto de rotación se describen tantos cilindros concéntricos como pistas haya. Por cada cara grabable

habrá una cabeza lectograbadora. Todas ellas acceden en forma simultánea como si fueran los dientes de un peine a un cilindro (cuya dirección está dada por un número relativo a la distancia del radio entre las cabezas y el eje). El concepto es importante, dado que la información se graba por cilindro, esto es, verticalmente. Considere que cada movimiento de cabeza es mecánico y, por lo tanto, resulta óptimo grabar todas las pistas de igual número sin modificar su posición. Cuando se completa un cilindro, se pasa al siguiente.

12.2.1 Controladora de disco

La electrónica del microprocesador y la memoria asociada de un disco es compleja y delicada; el conjunto de platos está incluido en una caja soldada de aluminio que provee un entorno libre de contaminación para la operación de las cabezas lectograbadoras.

El vínculo entre la unidad de disco rígido y el bus del sistema se realiza a través del conector de la interfaz del bus, que toma las señales de la parte electrónica que oficia de interfaz de la unidad de disco y la pasa al bus de entrada/salida de la computadora. La controladora del disco está constituida por:

- Controladora del motor del eje y brazo actuador, para asegurar que los platos giran a la velocidad correcta y que el brazo actuador posiciona las cabezas lectograbadoras en el lugar preciso sobre el plato.
- Controladora de interfaz, para comunicarse con la CPU del sistema de transferencia adecuado.
- Un microprocesador que "ejecuta" los "comandos" propios del disco, como "estacionar cabezas" o "detener motor", y una memoria asociada para almacenar el bloque que se leyó o que se va a transferir.

La cobertura magnética de una unidad de disco está compuesta por áreas "dominios" (en inglés se utiliza el término *frame*). Cada dominio es magnetizable. Antes de grabar los datos, la unidad usa las cabezas lectograbadoras para orientar los dominios en una región pequeña, de forma que todos los polos magnéticos apunten en la misma dirección.

Los 0 y 1 que conforman la información en un disco se almacenan como patrones magnéticos en la cubierta magnética de éste. Las cabezas lectograbadoras generan estos patrones al escribir los datos en los platos del disco. Luego, al leer del disco, las cabezas lectograbadoras convierten los patrones magnéticos almacenados en señales eléctricas que representan los datos almacenados. Las unidades de disco rígido suelen tener una cabeza lectograbadora a cada lado del plato.

12.2.2 Especificaciones técnicas de un disco

Cuando se debe seleccionar una unidad de disco apropiada para una computadora se deben considerar fundamentalmente las "especificaciones de desempeño" y hacer un estudio comparativo de precios y prestaciones. Hay especificaciones que permiten medir la funcionalidad mecánica de la unidad, otras, sus características electrónicas y otras, sus requerimientos eléctricos. Además, se consideran sus dimensiones físicas y requerimientos ambientales.

A continuación, se desarrolla un caso ejemplo tomado textualmente de un aviso publicado referido a una unidad de disco del mercado actual, por lo tanto, debe considerarse que en poco tiempo estas especificaciones cambiarán, porque serán mejoradas por tecnologías nuevas. Luego se describen las que consideramos más importantes de todas ellas.

Especificaciones de desempeño

- Velocidad de rotación 5400 RPM (nominal)
- Tamaño del buffer 8 MB
- Latencia promedio 4.20 ms (nominal)

Tiempos de búsqueda

Tiempo de búsqueda en lectura *12.0 ms*

Tiempo de búsqueda de pista a pista *2.0 ms* (promedio)

Tasas de transferencia

Buffer al host (Serie ATA) *1.5 Gb/s* (Máx)

Velocidad de transferencia (*buffer a disco*) *600 Mbits/s* (Máx)

Especificaciones físicas

Capacidad formateada *250.059 MB*

Capacidad *250 GB*

Interfaz SATA *1.5 Gb/s*

Dimensiones físicas

Altura *9.5 ± 0.20 mm*

Longitud *100.2 ± 0.25 mm*

Ancho *69.85 ± 0.25 mm*

Peso *0.117 kg*

Especificaciones ambientales*Acústica*

Modo inactivo *24 dBA* (promedio)

Modo de búsqueda *26 dBA* (promedio)

Temperatura

Operativo De *5 °C a 60 °C*

No operativo De *-40 °C a 65 °C*

Humedad

Operativo *8-90%* humedad relativa no condensante

No operativo *5-95%* humedad relativa no condensante

Altitud

Operativo De *-305 M a 3050 M*

No operativo De *-305 M a 12200 M*

Vibración

Operativo *0.00459 g²/Hz (10 to 500 Hz)*

No operativo *0.05102 g²/Hz (10 to 500 Hz)*

Especificaciones eléctricas. Requerimientos actuales*5 VCC*

Lectura/escritura *500 mA*

Inactivo *400 mA*

En espera *50 mA*

Apagado automático *20 mA*

Disipación de energía

Lectura/escritura *2.50 Watts*

Inactivo *2.00 Watts*

En espera *0.25 Watts*

Apagado automático *0.10 Watts*

Las especificaciones de partes mecánicas controladas electrónicamente son:

Tiempo de búsqueda

Tiempo de cambio de cabezas (*switch*)

Latencia rotacional

Tiempo de acceso

Las velocidades de transferencia del disco miden la velocidad de transferencia entre el *buffer* del disco y el *host*, que es una función puramente electrónica.

La velocidad de transferencia de datos determina la relación efectiva de transferencia de datos entre el *buffer* de disco y el soporte en disco.

12.2.3 Tiempos de acceso a disco

12.2.3.1. Tiempo de búsqueda

Se denomina tiempo de búsqueda al período que tarda el brazo en mover las cabezas lectograbadoras entre las pistas en milisegundos (ms , $1\ ms = 0,001). Los tiempos de búsqueda se computan considerando el tiempo de posicionamiento entre pistas adyacentes, el tiempo de posicionamiento entre la pista más interna y la más externa y el tiempo de búsqueda promedio que se determina que toma posicionar las cabezas lectograbadoras de la unidad de disco para un pedido de posición aleatoria.$

En general, es una especificación valiosa y el tiempo que se toma como parámetro es el período de búsqueda promedio.

12.2.3.2 Tiempo de cambio de cabezas o tiempo de switch

El brazo mueve todas las cabezas lectograbadoras sobre los platos de forma sincrónica. Sin embargo, sólo una de las cabezas puede estar leyendo o grabando datos a la vez. El tiempo de cambio de cabezas mide el período medio que le lleva a la unidad de disco cambiar entre dos de las cabezas cuando está leyendo o escribiendo datos. Este tiempo también se mide en milisegundos.

12.2.3.3 Latencia rotacional

Una vez que la cabeza lectograbadora se posiciona sobre la pista adecuada, debe esperar que la unidad de disco gire el plato al sector correcto. Este período de espera, llamado latencia rotacional, se mide en milisegundos y depende de la velocidad de giro de los discos (medida en RPM). A mayor RPM, menor latencia rotacional. En el peor caso, la cabeza llega al cilindro justo después de que el sector deseado pasó por debajo de ésta. La cabeza debe esperar que el plato dé un giro completo antes de posicionarse en forma correcta sobre el sector que se ha de leer o escribir.

En promedio, el disco necesita girar sólo media vuelta antes de que el próximo sector para leer o escribir esté debajo de la cabeza.

12.2.4 Tiempo de acceso a los datos

El tiempo de acceso a los datos (o tiempo de acceso) es una medida de lo que se tarda en posicionar una cabeza lectograbadora sobre una pista particular y encontrar el o los sectores de interés dentro de esa pista para leer o escribir. Por lo tanto, el tiempo de acceso es una combinación del tiempo de búsqueda, el tiempo de cambio de cabezas y la latencia rotacional y se mide en milisegundos.

Como ya se indicó, el *buffer* caché de una unidad de disco se usa tanto en las transferencias de datos del disco al *host* (lectura) como del *host* al disco (escritura). Se utilizan tecnologías de caché, como DisCache y WriteCache.

Con el caché de lectura, una vez que la CPU genera un pedido de datos, la unidad de disco accede a los sectores apropiados, los lee y los almacena en la memoria caché. No obstante, no sólo lee lo pedido sino que continúa leyendo los datos secuenciales hasta que el *buffer* caché esté lleno. La lectura de estos datos adicionales no pedidos se llama prebúsqueda o *caché look ahead*.

Una vez que los datos adicionales están en el caché, pueden enviarse directamente desde allí a la memoria interna mucho más rápido de lo que tomaría su obtención del disco en una operación de lectura subsiguiente. Por lo tanto, cuando la unidad de disco transfiere los datos, éstos se envían a una velocidad de transferencia denominada velocidad de ráfaga sostenida máxima.

La utilización de una técnica de prebúsqueda se apoya en el supuesto de que se accede a los datos de manera secuencial. Cuando la unidad de disco recibe un pedido de lectura, recuperá los sectores pedidos y prebusca tantos sectores secuenciales como pueda. Si esos datos son pedidos por el siguiente comando de lectura, pueden transferirse desde el caché en microsegundos (10^{-6} seg) en vez de milisegundos (10^3 seg). En general, la técnica mejora el tiempo de transferencia.

Durante las operaciones de escritura de datos, el caché de escritura permite que las transferencias del sistema al *buffer* y del *buffer* al disco se produzcan en paralelo. Esto elimina las latencias rotacionales durante los accesos secuenciales y solapa la latencia rotacional y el tiempo de búsqueda con procesamiento del sistema durante los accesos aleatorios. Esto permite que mejoren las velocidades de transferencia de datos sostenidas que se incrementan cuando las escrituras son secuenciales.



12.2.4.1 Velocidades de transferencia

Luego de posicionar la cabeza lectograbadora, la unidad de disco está lista para leer/escribir datos desde/hacia el disco. En cualquier caso, esto conlleva una transferencia de datos entre el disco y la memoria interna. Cuanto más rápida sea esa transferencia, menor será el tiempo que el proceso tenga sus datos disponibles.

La velocidad de transferencia de datos es sumamente dependiente de dos medidas: la velocidad de transferencia del disco –o la rapidez con que pasan los datos desde el disco hacia el *buffer* o memoria controladora– y la velocidad de transferencia del *host* –o la rapidez con que la controladora pasa los datos a la memoria interna a la que accede la CPU-. La velocidad de transferencia de datos se mide en megabits por segundo (Mb/s) o gigabits por segundo (Gb/s).

12.3 Dispositivos de almacenamiento removibles

Aunque las unidades de disco rígido son el medio de almacenamiento más utilizado para las computadoras, hay tecnologías alternativas que también satisfacen las necesidades de almacenamiento voluminosas de las aplicaciones actuales. Por otro lado, a medida que el procesamiento de datos sensibles se mueve de los *mainframes* a las computadoras personales, las soluciones de almacenamiento portables se volvieron muy útiles. Los discos fueron eliminados del mercado y, aunque fueron los dispositivos de almacenamiento removible más comunes de los últimos veinte años, resultan inadecuados por su capacidad limitada, por lo tanto, fueron desplazados por los CD-ROM y otros discos ópticos tales como los discos magneto-ópticos (MO), las tarjetas de memoria, los *memory sticks* y los DVD.

12.3.1 Discos ópticos

Los discos ópticos almacenan cantidades de datos grandes y se utilizan cabezales que emiten rayos láser para leer/escribir desde/en el medio. Al escribir en un disco óptico, el láser crea agujeros microscópicos denominados *pits*. Las áreas que no son quemadas por el láser se

llaman *lands*. El láser de lectura, de menor potencia que el de escritura, lee los datos en el disco óptico buscando *pits* y *lands*. Si bien ofrecen gran capacidad de almacenamiento, no son tan rápidos como las unidades de disco rígido.

Hay tres tecnologías de discos ópticos primarias disponibles para almacenamiento: unidades de CD-ROM (*Compact Disk-Read Only Memory*) y sus derivadas CD-R (*Compact Disk-Recordable* o disco compacto grabable), CD-RW (*Compact Disk-Rewriteable* o disco compacto regrabable), DVD-ROM (*Digital Video Disk-Read Only Memory* o disco de video digital solo de lectura) y el nuevo DVD-RAM (*Digital Video Disk-Random Access Memory* o disco de video digital regrabable); las unidades WORM (*Write Once Read Many* o escribibles una vez y legibles muchas veces –parecido al concepto de CD-R–) y las unidades de discos ópticos regrabables.

Los CD-ROM son, por lejos, la forma más popular de almacenamiento en disco óptico. Sin embargo, sólo pueden leerse. En la fábrica se usan láseres para crear un CD-ROM maestro y luego se hace un molde a partir de ese CD. Se inyecta plástico dentro del molde y se imprime el patrón de datos en el disco, como en una grabación fonográfica. La utilización típica de los CD-ROM es la distribución software, imágenes o música. En el presente casi todas las aplicaciones se distribuyen en CD-ROM; las estrellas de este formato son los juegos y las enciclopedias interactivas, debido a las posibilidades multimedia que brinda la capacidad de los CD-ROM.

Las unidades WORM se usan casi con exclusividad para almacenamiento en archivo de datos, en los que es importante que los datos no puedan cambiarse ni borrarse luego de haberse escrito, por ejemplo, para almacenamiento de registros financieros.

12.3.2 Discos magneto-ópticos (MO)

Los sistemas de discos magneto-ópticos (MO) combinan la tecnología de los medios magnéticos tradicionales, como las unidades de disco rígido, con la tecnología de los discos ópticos. Las unidades MO también son conocidas como floptical (disquete óptico). La tecnología MO permite a los usuarios empacar cientos de megabytes de datos en un disco que luce similar a un disquete de 3,5 pulgadas y viene, por lo general, en factores de forma de 3,5 y 5,25 pulgadas. Un disco MO está fabricado con materiales altamente resistentes a los campos magnéticos, o fuerza coercitiva, a temperatura ambiente.

Una unidad MO escribe en el disco usando una cabeza lectograbadora asistida por un láser. Un láser calienta la superficie del disco hasta su punto de Curie –esto es, la temperatura que permite que las partículas magnéticas de la superficie del disco sean alineadas por el campo magnético creado en la cabeza-. Luego, la cabeza pasa sobre la superficie del disco, polarizando aquellas áreas que fueron calentadas por el láser. Debido a que un láser puede enfocarse en un campo mucho más angosto que un campo magnético tradicional, los datos escritos en un disco MO con la asistencia del láser dan como resultado una densidad mayor que la disponible con tecnología de unidad de disco rígido convencional.

Durante la operación de lectura, la unidad MO usa el mismo láser para censar los datos almacenados en el disco. A medida que el láser recorre la superficie del disco, la unidad detecta una reflexión de luz por los bits de datos orientados en un sentido y no detecta reflexión alguna por los bits de datos orientados en sentido contrario. De esta forma, la unidad MO distingue entre los *0* y *1* almacenados en el disco.

Los discos MO tienen varias ventajas:

- Proveen densidades de datos muy altas, disponibles gracias a la asistencia del láser.
- Los datos almacenados en ellos pueden cambiarse a gusto –agregados, modificados o borrados– como si estuviesen almacenados en una unidad de disco rígido.

- Son resistentes a los campos magnéticos. Al contrario que los disquetes o los discos rígidos, un campo magnético por sí solo no puede alterar los datos sin el calor adicional provisto por un láser.
- Debido al uso del láser para asistir las operaciones de lectura y escritura, las cabezas lectograbadoras no necesitan estar tan cerca de la superficie del disco como en una unidad de disco rígido. En vez de volar, la cabeza de una unidad MO recorre la pista, lo que implica una posibilidad de choque menor.

La desventaja de la tecnología MO es que, debido a la intensidad alta del campo magnético creado con el uso combinado de cabeza y láser, la unidad no puede cambiar la polaridad del campo con gran rapidez. Por lo tanto, la unidad debe pasar dos veces sobre el disco para escribirlo. En la primera rotación todos los bits se orientan de la misma forma, borrando los datos de manera efectiva; en la segunda rotación, algunos de esos bits se reorientan en el sentido opuesto para distinguir los bits en 0 de los bits en 1.

Aun cuando algunas unidades MO giran, en promedio, a revoluciones por minuto comparables a las unidades de disco rígido, las dos rotaciones necesarias para escribir los datos las hacen dos veces más lentas que las unidades de disco rígido durante las operaciones de escritura. Por supuesto, los fabricantes de unidades MO están trabajando para tratar de comprimir el proceso de escritura a una sola rotación del disco. En el presente las unidades MO no son muy populares.

12.3.3 Tarjetas de memoria

El estándar PCMCIA, que define las *PC cards*, está aceptado internacionalmente y significa *Personal Computer Memory Card International Association*. Son tarjetas de memoria usadas para expandir las capacidades de la computadora con más medios de almacenamiento. Pueden ser tarjetas de E/S o de almacenamiento. Las *PC cards* consumen poca energía y son compactas, muy confiables y livianas, lo que las hace ideales para las computadoras portátiles, los PDA (asistentes personales digitales) y otros dispositivos de mano y de comunicaciones. Debido a su tamaño diminuto, las *PC cards* usadas para almacenamiento, llamadas comúnmente tarjetas de memoria, facilitan el transporte de datos; pueden usarse para almacenamiento de programas o intercambio de datos entre sistemas.

En la actualidad la tarjeta de memoria aún es más cara que el almacenamiento en medio magnético. Dado que los chips de memoria de las tarjetas son los componentes más caros, el costo de duplicar la capacidad de una tarjeta de memoria duplica el costo de producción. Por el contrario, los avances en la densidad magnética en las unidades de disco tienen como consecuencia incrementos mínimos en el costo.

Hay varios tipos diferentes de chips que se utilizan en las tarjetas de memoria de estado sólido y las tarjetas se nombran de acuerdo con ellos. En las secciones que siguen se detallan las diferencias y las similitudes entre los distintos tipos de tarjetas de memoria de estado sólido y se provee información acerca de los mejores usos para ellas.

12.3.4 Tarjetas ROM y OTP

Para un usuario final, una tarjeta ROM (*Read Only Memory* o memoria sólo de lectura) y una tarjeta OTP (*One Time Programmable* o programable una sola vez) lucen y se comportan igual. La diferencia está en la manera en que se programa la información almacenada en la tarjeta. Cuando se fabrica una tarjeta ROM, el programa o los datos, o ambos, van incorporados en los chips ROM. Debido a que la información almacenada en una tarjeta ROM es integral e inseparable del hardware, por lo general una tarjeta ROM es dependiente del dispositivo y no funcionará

en un sistema de computadora que no sea aquel para el que fue fabricada de manera específica (a menos que la arquitectura del sistema sea virtualmente idéntica). Por el contrario, las tarjetas OTP (al igual que otros tipos de *PC cards* de almacenamiento) se fabrican borradas, esto es, sin información alguna. Luego de fabricadas, sus chips pueden escribirse con un dispositivo especial. Una vez escritos, la información almacenada en una tarjeta OTP no puede borrarse o alterarse.

Debido al desplazamiento de las tecnologías nuevas, el uso actual de las tarjetas ROM y OTP es muy limitado, si no es nulo. Aunque tienen la ventaja de brindar una forma de almacenamiento no volátil –o sea que las tarjetas no necesitan una fuente de alimentación para mantener los datos intactos– hay nuevas tecnologías de almacenamiento no volátiles disponibles. La mayor desventaja en el uso de las tarjetas ROM y OTP es que, una vez que la información se almacena en la tarjeta, no puede alterarse en forma alguna. Además, el proceso de fabricación personalizado de los chips ROM hace que las tarjetas ROM sean muy caras, a menos que se produzcan en cantidades grandes. Antes, las compañías de software usaban tarjetas ROM y OTP para distribuir programas de aplicación a los usuarios de sistemas de computadoras muy pequeños, para incorporar unidades de disco rígido o disqueteras. Sin embargo, apenas lanzaban una versión nueva del software, cualquier tarjeta que quedaba en *stock* era obsoleta.

12.3.5 Tarjetas SRAM

Las tarjetas SRAM tienen una ventaja respecto de las ROM y las OTP: los usuarios pueden escribir, modificar o borrar la información almacenada en ellas. Sin embargo, los chips SRAM son volátiles, lo que hace que las tarjetas SRAM necesiten que se las alimente con algo de energía todo el tiempo. Si se interrumpe la energía, la información de la tarjeta se pierde. Mientras está insertada en el zócalo PCMCIA, una tarjeta SRAM toma energía de su *host* (una computadora u otro dispositivo electrónico). Cuando se quita del zócalo PCMCIA, se alimenta con su propia pila interna, una pila de reloj pequeña. Una sola de estas pequeñas pilas provee alrededor de un año de alimentación a la tarjeta, lo que indica la poca energía que consumen. Sin embargo, la desventaja de las SRAM es obvia: una falla en la pila significa la pérdida de los datos.

Las tarjetas SRAM se usan para volúmenes de almacenamiento relativamente bajos, debido a su costo prohibitivo. Las tarjetas SRAM nunca fueron muy populares entre los fabricantes de software ya que, aun cuando podían actualizarse con las versiones nuevas, las pilas que estaban en el estante podían descargarse en cualquier momento.

12.3.6 Tarjetas Flash

Uno de los últimos adelantos en las tarjetas de memoria es el uso de la tecnología de las memorias Flash. Las tarjetas Flash combinan las mejores características tanto de las SRAM como de las ROM/OTP. Al igual que los chips ROM u OTP, los componentes Flash no son volátiles, retienen los datos aun sin fuente de energía. Asimismo, al igual que los chips SRAM, los usuarios pueden escribir, modificar y borrar la información almacenada en ellas.

12.4 Resumen

En este capítulo se desarrollaron las distintas opciones de almacenamiento en memorias masivas o secundarias, ya sea en medios magnéticos u ópticos. El disco sigue siendo la opción preferida desde su aparición en el mercado (desde el ya en desuso disco flexible hasta los veloces discos rígidos actuales, cuyas prestaciones siguen siendo las de menor costo de

almacenamiento por bit, aumentando la velocidad de acceso a los datos en forma excepcional). Es recomendable el estudio profundo de lo expuesto en este capítulo y que conserve una mirada atenta a las nuevas tecnologías; las demás memorias que se han descripto se destacan por su portabilidad y por sus múltiples aplicaciones en el mercado de la computación portátil y en dispositivos como las cámaras fotográficas y de video.

12.5 Ejercicios propuestos

- 1) ¿A qué se denomina transferencia elemental?
- 2) ¿Qué es un ciclo de bus?
- 3) ¿A qué se denomina Arquitectura de buses?
- 4) Describa con sus propias palabras o con un esquema la comunicación HAND SHAKING
- 5) ¿Qué significan las siglas EISA? Describa la tecnología
- 6) ¿Qué significan las siglas VL? Describa la tecnología
- 7) ¿Qué significan las siglas PCI? Describa la tecnología
- 8) ¿Qué significan las siglas SCSI? Describa la tecnología
- 9) Describa cuál es la función de un software identificado como "driver de periférico".
- 10) Describa las tres funciones de un adaptador de video.
- 11) Reconozca y describa la función de cada una de las siguientes señales: OWS ("Zero Wait State"), AEN ("Address Enabled"), ALE ("Address Latch Enabled"), CLK, I/O CH CHK ("Channel Check"), I/O CH RDY ("Channel Ready"), IRQ ("Interrupt request"), IOR ("I/O Read"), IOW ("I/O Write"), RESET, SBHE ("System Bus High Enable").

12.6 Contenido de la página Web de apoyo

El material marcado con asterisco (*) sólo está disponible para docentes.



Resumen gráfico del capítulo

Autoevaluación

Video explicativo (01:39 minutos aprox.). Capítulos 12 y 13

Audio explicativo (01:39 minutos aprox.). Capítulos 12 y 13

Evaluaciones Propuestas*

Presentaciones*

13

Transferencias de información

Contenido

13.1 Introducción	300
13.2 Buses	300
13.3 Dispositivos de entrada/salida	304
13.4 Modalidades de entrada/salida	311
13.5 Resumen	315
13.6 Ejercicios propuestos	315
13.7 Contenido de la página Web de apoyo	316

Objetivos

- Comprender las funciones y principios de operación de los dispositivos de adaptación y conexión.
- Distinguir las relaciones más significativas que se dan entre el subsistema constituido por la CPU y la Memoria con los elementos del entorno.
- Diferenciar las diversas modalidades de transferencia como alternativas de resolución de las operaciones de E/S.



En la página Web de apoyo encontrará un breve comentario de la autora sobre este capítulo.

13.1 Introducción

Debido a la necesidad de conectividad de los diversos componentes para formar un sistema (que va desde la comunicación de los registros internos del microprocesador hasta la transferencia a los dispositivos de almacenamiento más rápidos desarrollados en el capítulo anterior), se ha establecido una jerarquía por niveles de comunicación presentada en orden consecutivo, desde las transferencias elementales que ocurren en la CPU (buses internos al chip) hasta las transferencias completas de bloques de información que ocurren en los buses de entrada/salida.

Se han clasificado bajo el nombre de dispositivos de entrada/salida a los componentes hardware que se involucran en la interconexión, como puede ser un adaptador de red, una controladora de disco, un simple puerto paralelo –o algo más complejo como un puerto USB–, una interfaz SATA para disco o, incluso, un procesador dedicado al flujo de entrada/salida denominado canal de entrada/salida (por dar un ejemplo, el IOP332 de Intel es un procesador de entrada/salida para unidades de almacenamiento masivo).

Por último, se hace referencia al software de entrada/salida, en cuanto a la función de un *driver* de dispositivo, y se explica la tradicional clasificación de las distintas modalidades de transferencia por su actual vigencia y para describir la relación entre los programas en estado de ejecución en la CPU cuando se produce una necesidad de una comunicación con el entorno.



Un **bus** es un elemento de comunicación que relaciona cierto número de componentes o dispositivos. Se puede definir como un conjunto de conductores que transfieren señales eléctricas en forma pasiva, asociado con un hardware que regula su actividad, denominado **controlador de bus**.

13.2 Buses

La transmisión de información en un sistema digital puede utilizar como medio de enlace conductores por los que se transfieran (por ejemplo) tensiones que representen los valores lógicos 0 y 1 y que denominamos buses, utilizados para la conectividad de módulos de hardware.

Los tipos de señales que transfieren se pueden clasificar según su tipo de información en señales de dirección, control y dato.

Cuando se establece la conexión entre dos unidades, una actúa de emisora y la otra de receptora. En el caso de que ambas unidades puedan invertir sus papeles, el **sentido** de la transferencia cambia, por lo tanto, una de las cosas que un controlador de bus debe controlar es el sentido de la transferencia. Cada dato transferido por un bus se conoce como **transferencia elemental** y se produce en un tiempo determinado, regulado por el controlador, y denominado **ciclo de bus**.

Sin detenernos en profundizar, condiciones como la distancia entre las unidades comunicadas, la velocidad necesaria para llevar a cabo la transferencia o la posibilidad de error en un bit del mensaje permiten seleccionar el tipo de soporte físico que transmitirán los bits. Los soportes más comunes son las pistas de circuito impreso y los cables planos. Las señales que representan bits sobre el bus son digitales y pueden transferir el dato en **serie** o en **paralelo**.

El grado de paralelismo del bus unido a la velocidad que admite para lograr la transferencia se denomina **caudal** del bus. Cuando el ciclo de bus está controlado por el reloj del sistema, el bus realiza una transferencia sincrónica; en cambio, cuando su operación está controlada por los dispositivos conectados a él realiza una transferencia asincrónica. Sin embargo, para profundizar en sus características principales conviene que pensemos dónde los encontramos y cuáles son las condiciones que imponen los dispositivos que conecta. Establezcamos la siguiente jerarquía.



Consideraremos en este capítulo sólo aquellos medios de transmisión que conducen señales eléctricas para representar la transmisión de bits.

13.2.1 Jerarquía de buses

Si establecemos un orden por niveles de buses, desde los internos en un chip hacia afuera, podemos determinar características propias de cada nivel.

13.2.1.1 Buses internos al chip

Por ejemplo, se consideran buses las conexiones internas que comunican registros en un chip, como en el caso de las conexiones entre los registros internos de un microprocesador. Piense que un bus que relaciona dos registros dentro de un chip tiene el menor nivel de complejidad, o sea que es el tipo más simple, y podemos decir que es "un camino" entre registros.

13.2.1.2 Buses que conectan chips sobre una placa

Los distintos módulos que albergan chips se comunican entre sí conectándose a una placa o tarjeta. Estos módulos se interconectan entre sí también a través de buses. Estas conexiones son más evidentes al usuario que las ve, por ejemplo, sobre la superficie de circuito impreso configurando pistas o caminos metálicos. En general estos buses realizan transferencias sincrónicas. Las señales del mensaje que constituyen el bus de dato son bidireccionales. Las señales que acompañan a las señales de datos permiten el direccionamiento y el control de la transferencia.

A continuación, se describen todas las señales correspondientes a un **bus de sistema** que conecta un microprocesador con la memoria principal; el microprocesador no es actual pero se presenta así para verlo completo.

Este bus tiene diecisésis líneas de dirección A_0 hasta A_{15} y ocho líneas de dato D_0 a D_7 . Cada señal está asociada a un *pin* o una patilla que se referencia con un número, en este caso, de 1 a 39. Cada *pin* tiene una identificación estándar que se muestra en la Tabla 13-1.:

Tabla 13-1. Descripción de la identificación estándar de cada pin.

1	GND	Masa
2/8	A14/A8	Líneas del bus de direcciones no multiplexadas. Junto con la patilla 39 mantienen su valor durante todo el ciclo de funcionamiento del bus
9/16	AD7/AD0	Líneas compartidas (multiplexadas). Son las direcciones A_0 - A_7 durante los ciclos T_1 y los 8 bits de datos D_0 - D_7 , durante los ciclos T_2 , T_3 , T_w y T_4
17	NMI	Petición de interrupción relacionada con fallos de hardware.
18	INTR	Petición de interrupción enmascarable relacionada con solicitudes de hardware.
19	CLK	Entrada de señal de reloj (<i>clock</i>)
20	GND	Masa
21	RESET	Entrada de la señal de inicio del procesador.
22	READY	Esta señal sincroniza el procesador con los periféricos. Cuando un dispositivo necesita más tiempo pone a cero esta línea y el procesador inserta tiempo de espera entre los ciclos T_2 y T_3 .
23	TEST	Esperar para comprobación (<i>wait for test</i>). Después de cada instrucción el procesador setea el estado de esta señal. Si es bajo (0) es que la ejecución continúa normalmente; si es alto (1), es que permanece detenido en un estado de espera
24/25	QS1/QS0	Estas señales permiten que un dispositivo externo pueda identificar el estado de una cola de instrucciones del procesador. QS1 QS0 Significado 0 0 Sin operación 0 1 Primer byte del código de instrucción 1 0 Cola vacía 1 1 Siguiente byte del código de instrucción

1	GND	Masa																																				
		Permite determinar 8 estados del procesador:																																				
26/28	S2/S0	<table border="1"> <thead> <tr> <th>S2</th> <th>S1</th> <th>S0</th> <th>Significado</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> <td>Reconocimiento de interrupción enmascarable</td> </tr> <tr> <td>0</td> <td>0</td> <td>1</td> <td>Lectura en un puerto E/S</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> <td>Escrivura en un puerto E/S</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> <td>Detención (<i>halt</i>)</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> <td>Acceso a código</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> <td>Lectura de memoria</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> <td>Escrivura de memoria</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> <td>Pasivo</td> </tr> </tbody> </table>	S2	S1	S0	Significado	0	0	0	Reconocimiento de interrupción enmascarable	0	0	1	Lectura en un puerto E/S	0	1	0	Escrivura en un puerto E/S	0	1	1	Detención (<i>halt</i>)	1	0	0	Acceso a código	1	0	1	Lectura de memoria	1	1	0	Escrivura de memoria	1	1	1	Pasivo
S2	S1	S0	Significado																																			
0	0	0	Reconocimiento de interrupción enmascarable																																			
0	0	1	Lectura en un puerto E/S																																			
0	1	0	Escrivura en un puerto E/S																																			
0	1	1	Detención (<i>halt</i>)																																			
1	0	0	Acceso a código																																			
1	0	1	Lectura de memoria																																			
1	1	0	Escrivura de memoria																																			
1	1	1	Pasivo																																			
29	LOCK	La señal es activada por las instrucciones que tienen el prefijo <i>LOCK</i> (en ensamblador), hasta que se completa la instrucción siguiente. Durante este tiempo ningún otro dispositivo puede adquirir control del bus																																				
30/31	RQ/GT	<p>Señales de pedido o concesión de control del bus por cualquier dispositivo distinto del procesador capaz de asumir su control (<i>bus mastering</i>). El proceso es el siguiente:</p> <ul style="list-style-type: none"> • Un pulso de un ciclo de reloj desde un dispositivo externo solicita el control del bus de sistema desde otro <i>bus master</i>. Esta señal se conoce como <i>HRQ</i> (<i>Hold Request</i>) • Un pulso de un ciclo del procesador durante los tics <i>T4</i> o <i>T1</i>, es la señal al peticionario de que el procesador colocará la señal de reconocimiento de petición de control <i>HLDA</i> (<i>Hold Acknowledge</i>) en el próximo tic de reloj. • Por último, un pulso de un ciclo <i>CLK</i> del peticionario indica al procesador que el control solicitado va a terminar, de modo que el procesador puede recuperar el control del bus en el próximo ciclo de reloj. Es la señal <i>EOP</i> (<i>End Of Process</i>) 																																				
32	RD	Lectura. Esta señal indica que el procesador está realizando un ciclo de lectura de memoria o de puerto (según el estado de la señal S2).																																				
33	MN/MX	Esta señal se utiliza para activar uno de dos modos de operación del procesador.																																				
34	SSO	Línea de estado.																																				
35/38	A19/A16 S6/S3	Líneas compartidas (multiplexadas) de direcciones/estado. Durante el ciclo <i>T1</i> de las operaciones E/S con memoria, las líneas 35-38 contienen los 4 bits más significativos (A19-A16) del bus de direcciones (este bus tiene las líneas A0-A19).																																				
39	A15	Línea del bus de direcciones																																				
40	Vcc	Alimentación. + 5 V. c.c. +/- 10%																																				

13.2.1.3 Buses que conectan distintas placas

La conexión entre las distintas placas necesita buses que permitan su interconexión. Las placas se conectan entre sí por medio de buses pero esta vez el control de la transferencia es más complejo. En este nivel, los diversos diseños de placas que provienen de distintos fabricantes crean la necesidad de compatibilizar la forma de transferir un dato. Entonces ya no podemos pensar en un bus como un simple camino de bits, sino como un componente de complejidad superior estandarizado e independiente, que se acopla a las componentes del sistema hardware de la computadora. Este tipo de buses se denomina normalizado y tiene un "nombre" en el mercado.

Lo expuesto hasta el momento deja de lado la relación de los buses respecto de los dispositivos de E/S. Si bien en el último modelo se pueden acoplar placas que controlen dispositivos de entrada/salida, sus detalles se analizan con mayor profundidad en la sección siguiente, que trata de las transferencias de información entre el bus de sistema y el exterior, denominados buses de entrada/salida.

13.2.1.4 Buses de entrada/salida

Los buses afectados a la entrada y la salida de información (datos, direcciones, control y estado) determinan una multiplicidad de estructuras que se denominan **arquitectura de buses**. Una computadora puede variar desde una **arquitectura** de bus único hasta complejas estructuras de buses. La arquitectura de buses permite definir normas de comportamiento para la transferencia de datos desde o hacia los dispositivos de entrada/salida. Su diseño involucra, por ejemplo, la determinación de controles de estado durante y luego de la transferencia, las velocidades admitidas por los distintos componentes y la cantidad de bits de datos para enviar entre ellos. Para algunas computadoras el bus que relaciona la CPU con la memoria está separado del que conecta estas unidades con los dispositivos de entrada/salida. La cantidad de variantes que admiten los sistemas convierte al subsistema de entrada/salida en algo muy relevante que afecta el diseño del hardware y el software del sistema total.

Las señales más comunes de los buses de entrada/salida son:

- **0WS (zero wait state)**. Es utilizado por los dispositivos rápidos conectados al bus para prevenir al procesador que no inserte estados adicionales de espera.
- **AEN (address enabled)**. Se utiliza para indicar si es la CPU o el controlador de acceso directo a memoria el que tiene control sobre las líneas de datos y direcciones en ese momento. Esta señal activa la interpretación de otras relacionadas con el controlador DMA, que, se identifican con las siglas MEMR, MEMW, IOR e IOW.
- **ALE (address latch enabled)**. Cuando está alta, señala que la CPU colocó una dirección válida en el bus de direcciones.
- **CLK**. Señal del reloj que conecta directamente con un *pin* del procesador.
- **OSC**. Señal del oscilador; basada en la frecuencia de un cristal instalado en el sistema.
- **I/O CH CHK (channel check)**. Es una señal de detección de errores, que indica que algún dispositivo del bus ha detectado un error de paridad.
- **I/O CH RDY (channel ready)**. Sirve para avisar al procesador o al controlador de acceso directo a memoria que un dispositivo lento necesita tiempo extra para estar preparado.
- **IRQ (interrupt request)**. Señal de interrupción. Es una señal que se origina en un dispositivo externo para indicar al procesador que se requiere su atención inmediata (de ahí su nombre, literalmente petición de interrupción). Se solicita al procesador que suspenda lo que está haciendo para atender la petición.
- **IOR (I/O read)** u orden de lectura. Esta señal indica a un dispositivo E/S conectado al bus que debe colocar un dato en el bus de datos.
- **IOW (I/O write)** u orden de escritura. Esta señal indica a un dispositivo E/S que debe leer el dato situado en el bus de datos.
- **RESET**. Una señal en esta línea, que conecta con un *pin* especial del procesador, originando su reinicio.
- **SBHE (system bus high enable)**. Cuando está activado indica que se está realizando una transferencia de datos.

Evolución de las distintas tecnologías del bus: un poco de historia

El bus ha sido el camino de comunicación de alta velocidad tradicional entre la CPU y la memoria del sistema. Las velocidades de la CPU y los anchos del bus se incrementaron de 8 MHz y 16 bits en 1979, con sistemas basados en el procesador 8086 hasta a los sistemas avanzados con microprocesadores Pentium III; asimismo, las velocidades de transferencia de disco aumentaron. Los avances en la tecnología siguieron a esas mejoras con frecuencias de reloj más altas y anchos de canal mayores, por lo que la CPU y la memoria pueden operar con máximo rendimiento.

Los periféricos (como las unidades de disco rígido), fueron conectados en las PC por el bus de expansión, que se conoce como bus ISA (*Industry Standard Architecture*). Con el tiempo se introdujeron buses de expansión adicionales, como el EISA (*Enhanced ISA*) y el MicroChannel. Aunque estos buses nuevos mejoraban las limitaciones de ancho y velocidad del bus ISA, cada uno fue excedido en desempeño por los diseños modernos de las PC basadas en los procesadores Pentium, en las que las unidades de disco rígido aprovecharon las posibilidades provistas por el Bus Local. El uso del Bus Local para la transferencia de datos hizo posible que las velocidades de transferencia sean hasta cuatro veces superiores a las del bus ISA.

Se desarrollaron dos estándares para la implementación del Bus Local: la *Video Electronics Standards Association* (VESA) creó el VESA Local (VL) bus e Intel creó el *Peripheral Components Interface* (PCI) local bus. El bus local PCI también se usó en los sistemas 486 y compatibles. Más aún, se convirtió en el estándar para todos los diseños de computadoras personales basados en los procesadores Pentium, porque ofreció un conjunto de comandos para comunicación más completo que el bus VL, además de propiciar un mejor soporte para el estándar *Plug and Play*.

Ejercicio:

Obtenga información sobre los buses más utilizados en las placas actuales y compare las velocidades presentes respecto de las "históricas", enunciadas en la sección "Evolución de las distintas tecnologías de bus".

13.3 Dispositivos de entrada/salida

Se denominan dispositivos de E/S tanto las unidades periféricas en sí como aquellas "intermediarias", que se encargan de efectivizar una transferencia entre la memoria interna y la memoria externa en los periféricos. En esta sección abordaremos controladores, adaptadores, puertos e interfaces.

En toda transferencia se utilizan señales de control, dato y dirección, esta vez sobre los buses de E/S. Las señales de control y tiempo se utilizan para regular la transferencia elemental, indicando cómo y cuándo debe ocurrir. O sea que permiten contestar las siguientes preguntas:

¿Puede producirse una transferencia ahora?

¿Qué tipo de transferencia se va a realizar?

¿Cuándo comienza?

¿Cuándo termina?

¿Se realizó con éxito?

Estas preguntas nos permiten deducir que las funciones de la "lógica o inteligencia" de un bus de E/S son:

- Comunicarse con el periférico y el sistema CPU-memoria.
- Controlar la temporización durante la transferencia.
- Almacenar temporalmente bits para "paliar" la diferencia de velocidad entre emisor y receptor.
- Detectar si se produjeron errores durante la transferencia.

Cuando la actividad del bus de E/S es sincrónica se utilizan señales del *clock* que regulan la transferencia, por lo tanto, la frecuencia de *clock* es un parámetro para considerar respecto de la velocidad de la transferencia. Un bus sincrónico requiere que los dispositivos conectados a él estén "sintonizados" a esa frecuencia y que todas las actividades se produzcan en intervalos de tiempo fijos.

Cuando la actividad del bus es asincrónica, las señales del *clock* no regulan su operación. Por lo tanto, su velocidad depende de los tiempos de los dispositivos conectados a ellos.

El siguiente es un resumen de las señales diferentes comunes a la mayoría de los buses de E/S para "controlar" la transferencia:

- Señal de *clock*, que permite sincronizar el ciclo del bus con la operación de la CPU (CLK).
- Señal que puede habilitar un tiempo de espera.
- Señal de lectura o escritura a un dispositivo de E/S o a la memoria (IOR o IOW).
- Señales de interrupción a la CPU (IRQn).
- Señales de reconocimiento (aceptación o requerimiento), que permiten el diálogo de los dispositivos durante la transferencia (p. ej., la técnica de *hand shaking* o "apretón de manos").
- Señal de bus cedido u ocupado.

Las señales de dirección permiten representar la dirección del emisor y el receptor ($A_n = address\ sub\ n$).

Cada dispositivo de E/S tiene asociada una combinación binaria. Por ejemplo, cuando se envía una transferencia al receptor, éste "reconoce" su dirección tomando los bits A_n del bus y decodificándolos con un circuito lógico o "lógica circuital". O sea que las señales de dirección permiten contestar las preguntas

¿de quién? y ¿a quién?

Las señales de dato representan los bits del mensaje que se ha de transferir ($D_n = data\ sub\ n$). El "ancho de bus" o bien la cantidad de líneas afectadas a la transferencia elemental dan una medida de potencial de trabajo; esto es, a mayor cantidad de bits transferidos en paralelo mayor capacidad de transferencia.

Las unidades de comunicación con el bus son unidades hardware, que actúan de intermediarias en la comunicación CPU - memoria interna → controlador - periférico y resuelven, fundamentalmente, el problema de disparidad en los tiempos de operación entre las unidades que conectan. Se pueden agrupar en:

- Interfaces paralelo.
- Interfaces serie.
- Controlador DMA (o sólo DMA's).
- Canal o procesador de entrada-salida (*Input Output Processor* o IOP).

La elección del tipo de intermediario tiene que ver con las características de cada dispositivo que se ha de conectar. Así, en un mismo sistema pueden existir todos o algunos de ellos. A su vez, cada intermediario puede estar asociado con más de un dispositivo. Antes de involucrarnos en forma directa con ellos revisaremos algunos conceptos.

Los dispositivos de E/S permiten la comunicación de la CPU/memoria con el medio externo. O sea que son aquellas unidades que permiten la entrada (*Input* o I) o la salida (*Output* u O), o ambas.

Un teclado y un monitor son típicos dispositivos periféricos, de entrada el primero, de salida el segundo. La relación entre la CPU y los periféricos no puede ser directa sino que se necesitan nexos físicos, cuya función establezca los controles para lograr una transferencia. Éstos también forman parte del conjunto "dispositivos de entrada/salida" y se describirán en esta sección del capítulo. En el presente la variedad de dispositivos que se puede conectar en un sistema complica la relación entre ellos; entonces, se hace necesaria la implementación de un subsistema denominado "de entrada/salida", que permita representar el juego de señales que controlen las transferencias y contemplen las relaciones de tiempo entre los distintos dispositivos. Este subsistema debe regular el tráfico entre la CPU, la memoria interna y todos los dispositivos asociados con la entrada/salida de datos.

13.3.1 Controladores

El término **controlador** se utiliza en gran medida para definir cualquier unidad hardware "que gobierna" a otra.

En este caso nos referiremos, de manera específica, en primer lugar a controladores de periférico. Un **controlador de periférico** es un dispositivo asociado en forma directa al periférico, que puede estar físicamente integrado a él, o bien separado de éste, y está constituido por:

- Un *buffer* interno (memoria RAM), que permite el almacenamiento de la información que "viaja" desde o hacia el soporte.
- Una lógica de control, que interpreta comandos de periférico, genera señales para su ejecución y gobierna así la unidad.

Por ejemplo, un controlador de disco permite el acceso al soporte de información, que exige una serie de actividades que son muy distintas si comparamos dispositivos de distinto tipo. Para acceder a un sector del disco, el controlador acciona el motor que hace girar el soporte. Cuando éste alcanza una velocidad constante, el controlador gestiona el movimiento del brazo desde una posición inicial hasta otra calculada, según el número del cilindro que se desea acceder. Con esta descripción elemental del conjunto de actividades, necesarias para el acceso, podemos darnos cuenta de que el controlador de una impresora no realiza ninguna de estas actividades, sino que las suyas tendrán que ver con el avance del papel, el movimiento del cabezal de impresión, etcétera. Así, cada dispositivo distinto se asocia con su propio controlador. Su función se puede resumir en, por lo menos, dos operaciones fundamentales:

- Aislar el software de servicio de entrada/salida, que se "ocupa" de la transferencia de los detalles específicos del hardware del periférico y los convierte en invisibles.
- Compatibilizar la velocidad del periférico respecto de la del resto del sistema.

Otros controladores, cuya función permite gobernar actividades del sistema, liberan a la CPU de ciertas tareas que pueden llevarse a cabo de manera independiente. Algo así como cuando realizamos una tarea que implica un razonamiento y a la vez percibimos que nos estamos quemando un dedo y lo retiramos. Ejemplos de controladores de sistema pueden ser el controlador DMA –que permite una transferencia entre un dispositivo de entrada y salida y la

memoria interna, sin intervención de la CPU (sólo con su permiso)–, el controlador programable de interrupciones –que atiende solicitudes de distintos dispositivos que requieren atención de la CPU y arbitra sus demandas– y el controlador de memorias tipo DRAM –indicado en el capítulo de memorias–.

Cuando un controlador se involucra con una transferencia, la CPU recibe parámetros de un programa que ejecuta; a través de una interfaz, este programa forma parte de la administración de entrada/salida y se denomina **manejador de dispositivo** (*device driver*).

Los *drivers* tienen una relación directa con las particularidades físicas del dispositivo que manejan, o sea que son programas cuyos comandos o instrucciones dependen por completo del dispositivo; por ejemplo, para un disco son coherentes los comandos del tipo leer, recalibrar, escribir. Es decir que el *driver* es quien establece la secuencia lógica de comandos y el controlador los reconoce y activa los mecanismos necesarios para su ejecución.

13.3.2 Adaptadores

Un adaptador provee una función para conectar y lograr la operación de un componente conectado a un bus. Un adaptador puede residir en una tarjeta o en la placa del sistema (integrado en un chip). Por ejemplo, un adaptador gráfico de video se usa para controlar la operación de un video y su relación con el bus.

Las unidades que constituyen este adaptador son básicamente el controlador de video (p.ej., trazado de rayos), el procesador de video (p.ej., procesa imágenes según estén codificadas; como rotarlas) y el conversor digital-analógico (p.ej., convierte la imagen digital a señal analógica y regula la frecuencia de refresco de pantalla). O sea que todo dispositivo que no cumpla la función por sí solo necesita su propio adaptador; por ejemplo, un adaptador de red es un dispositivo que, instalado en una placa principal de una computadora personal, conecta ésta físicamente a una red que puede ser pública o privada.

13.3.3 Puertos de entrada/salida

Un puerto (*port*) es un área de almacenamiento alojada en una interface, que permite la comunicación de un periférico con la memoria para enviar o recibir una secuencia de bits.

El software de sistema la identifica con un nombre, por ejemplo COM1, de la interfaz RS232 (el protocolo de transferencia puede "dar nombre" a la interfaz) al que nos referimos por su extendido uso, pero actualmente reemplazado por puertos más evolucionados como el USB.

13.3.4 Interfaces

Una interfaz es un hardware que actúa de nexo entre un periférico o un adaptador y el bus. Sirve, en primer término, para adecuar las señales y preparar la transferencia elemental basada en un protocolo (p.ej., un byte). No tiene capacidad suficiente para tomar la responsabilidad de la transferencia completa (bloque), son la CPU o el canal los que asumen el control de una transferencia completa.

13.3.4.1 Interfaz paralela

Es un dispositivo hardware que permite el control de la transferencia en paralelo entre el bus de sistema y un periférico. Está asociada a una lógica de direccionamiento que permite establecer que esa interfaz ha sido seleccionada. La interfaz cuenta con registros denominados *ports*. Un *port* está dividido en partes, por ejemplo, registro de dato (o *port* de dato) y

registro de control (o *port* de control). En la relación procesador-interfaz-periférico, el *port* está asociado al bus de sistema en su relación con la CPU y al bus E/S en su relación con el periférico; su función es lograr la transferencia elemental. El registro de control tiene una función doble: en primer lugar, recibe un comando que puede enviar al periférico y, en segundo lugar, recibe señales de control de estado; o sea que cada uno de sus bits se puede interpretar como indicadores de estado o *flags* que informen acerca de la transferencia.

Cuando la CPU inicia una transferencia, coloca un comando en este registro, que se conoce como comando de inicialización; este comando determina si la operación va a ser de entrada o de salida y genera las señales de "entrada" o "salida" que indican el sentido de la transferencia.

Presentaremos la actividad de la interfaz-periférico denominada *handshake* para una operación de salida (y luego de iniciada la transferencia). La CPU escribe el mensaje en un registro de dato del puerto o *port* de datos. La interfaz genera una señal de TE-ENVÍO-DATO al periférico. Cuando el periférico la toma, genera a su vez una señal de DATO-RECONOCIDO hacia la interfaz. La CPU testea el registro de control de la interfaz, que le informa si la transferencia del mensaje fue exitosa. El procedimiento se puede repetir tantas veces como sea necesario. En una operación de entrada (y luego de iniciada la transferencia), el periférico coloca el mensaje para transferir (actúa de emisor) en un bus de dato E/S y genera una señal TE-ENVÍO-DATO para la interfaz. La interfaz "contesta" con una señal de DATO-RECONOCIDO al periférico. La CPU testea el registro de control en la interfaz y toma el mensaje vía el bus de dato.

Esta conexión entre la interfaz y el periférico permite que ambos conozcan el estado de la otra unidad y se asegure así el éxito de la transferencia.

Las señales de control (TE-ENVÍO-DATO, DATO-RECONOCIDO) se habilitan con un estado lógico y se invalidan con el estado opuesto. Cuando una interfaz está constituida por dos puertos o más, la CPU indica la dirección del puerto implicado.

13.3.4.2 Interfaz serie

Es un dispositivo hardware que permite el control de la transferencia de bits en serie entre el bus y un dispositivo de E/S. Está asociado con una lógica de direccionamiento que permite establecer que esa interfaz fue seleccionada por el procesador para la transferencia. Los registros de la interfaz constituyen el denominado puerto serie. La interfaz se inicializa colocando un byte en su registro de control, que establece el tipo de transferencia. En el modo salida la interfaz recibe un dato enviado por la CPU a través del bus de dato; este dato se almacena en un registro de recepción relacionado con un registro de desplazamiento, que provoca la serialización de la unidad de información para transferir hacia el dispositivo. En el modo entrada otro registro de desplazamiento recibe la información bit tras bit desde el dispositivo y la coloca a disposición de la CPU cuando la unidad de información está completa en el registro. La interfaz serie acepta las modalidades sincrónica y asincrónica; sin embargo, cuando los datos se transmiten en una línea desde un punto al otro, se produce un retardo de la señal en el medio de transmisión, que provoca incertidumbre acerca de dónde termina un bit y dónde empieza el otro. Además, otra desventaja con la transmisión paralela es que la transmisión serie crea la dificultad de la delimitación de los caracteres; esto es, al enviarse los bits uno tras otro, no se sabe dónde empieza y dónde termina un carácter. La técnica utilizada para resolver el problema depende de que la transmisión sea sincrónica o asincrónica.

Caso ejemplo: tipos de interfaz para disco rígido

Una interfaz de disco es un estándar que comprende los conectores, las señales, los comandos y el protocolo de transferencia de datos. Estos estándares avanzaron desde las IDE hasta las SCSI. En el presente los fabricantes de discos incluyen en la unidad de disco la interfaz integrada.

Como ejemplo, observe el anuncio siguiente publicado por el fabricante de discos Seagate®.

The screenshot shows a web page from the Seagate website. At the top right, there's a large circular callout highlighting the product information for a "DISCO DURO [1-00-300GBATA]". The product details include "Item #": 103 and a price of "\$1,862". Below this, the "Item Description" section provides a detailed technical overview of the hard drive's capabilities and applications. To the left of the main content is a sidebar with a navigation menu titled "Menú de Categorías" containing links like "Accesorios", "Placas", "Cámaras", "Cámaras", "Procesadores", "Memoria RAM", "Procesadores", and "Placas".

Ejemplos de interfaces relacionadas con discos:

- Interfaz IDE (ATA)
- Interfaz E-IDE (*Enhanced IDE*) (ATA-2).
- Interfaz Ultra ATA (Ultra DMA/33).
- Interfaz SCSI
- Interfaz SATA.

Ejercicio:

Investigue las características técnicas de estas interfaces y realice una tabla comparativa que contemple las diferencias entre ellas, considerando los parámetros modo y velocidad de transferencia.

13.3.5 Canales o procesador E/S

Es un procesador "dedicado" o "específico" para controlar las transferencias de E/S sin intervención de la CPU en la ejecución del software de E/S. El IOP (*Input-Output Procesor*) realiza sus actividades en paralelo con la actividad que involucra a la CPU. Obtiene de la memoria y ejecuta las instrucciones de E/S y también puede realizar cálculos, saltos y otras tareas propias de la CPU pero orientadas a la gestión de E/S. Cuando la CPU ejecuta un proceso y decodifica una instrucción de E/S, inicializa un programa para el canal (denominado programa de canal); a partir de allí, el IOP se encarga de hacer efectiva la gestión, mientras la CPU continúa con la ejecución de otro proceso. La velocidad de la transferencia depende del periférico que involucre.

13.3.6 Transferencias de entrada/salida

Los aspectos fundamentales para resolver durante una transferencia son:

- Sincronizar los tiempos de transferencia entre la CPU-memoria y el periférico. Intuitivamente podemos entender que el tiempo que pierde una impresora en imprimir una línea en el soporte, en este caso el papel, es mucho mayor que el que tarda la CPU en enviar los caracteres que constituyen la línea desde la memoria principal hacia el intermediario.
- Decodificar los bits que identifican al dispositivo. Cada dispositivo de E/S se refiere con una agrupación de bits que constituye su dirección, y que lo identifica respecto de los demás. El dispositivo debe incluir una lógica de direccionamiento que le permita reconocerlo.
- Convertir, si es necesario, un mensaje serie a paralelo, o al revés.
- Convertir, si es necesario, el mensaje enviado de un formato a otro. Es posible que la unidad de transferencia del dispositivo no sea compatible con la unidad de trabajo de la CPU.
- Convertir, si es necesario, el mensaje enviado de un código a otro.
- Controlar, si es posible, que el mensaje enviado se reciba en forma correcta.
- Decodificar un comando para el dispositivo.
- Controlar las banderas de estado.

Se denominan maestros a los dispositivos que tienen el control del bus en un momento determinado. Por ejemplo, son maestros la CPU o el controlador DMA (cuya gestión describiremos más adelante). Un maestro puede enviar las señales de control, dirección y dato sobre el bus; conoce la dirección del emisor del mensaje y la dirección del receptor. Los dispositivos restantes, conectados al bus pero que no lo controlan, se denominan esclavos; un esclavo puede pedir un servicio de transferencia (*request*) pero no inicializarla. En los sistemas más simples, la CPU es el maestro absoluto y controla toda transferencia a través del bus. En los más complejos y, por lo tanto, más frecuentes, el uso del bus es compartido alternativamente por maestros ocasionales; incluso pueden armarse sistemas de varios maestros fijos que comparten el control de las transferencias sobre el bus. La estrategia de control del bus depende de la arquitectura diseñada para el sistema y se denomina "arbitraje" del bus.

Todas las responsabilidades descriptas se reparten, de alguna manera, entre la CPU, la interfaz y el controlador del periférico. El hardware "obedece" a los servicios de E/S provistos por el sistema operativo para efectivizar la transferencia. El software de entrada/salida incluye los comandos que se pueden aplicar al periférico para hacer efectiva la transferencia. El hardware determina las relaciones físicas que se establecen entre la CPU, la memoria, los controladores de sistema, los controladores de periférico y los intermediarios utilizados, guiando el flujo de la transferencia con señales de control y dato.

13.3.7 Drivers

Son aquellos programas que "conocen" el dispositivo periférico. Como ya establecimos, todas las particularidades de los distintos tipos de periféricos son conocidas por el sistema operativo por medio de *drivers* que, como ya se indicó, son programas en cuya codificación se hace referencia a los comandos propios para cada periférico. Así como un controlador puede atender varios periféricos idénticos, un *driver* de un periférico es un programa bastante diferente a un *driver* de otro. Cada *driver* actúa como un receptor de requerimientos de otros programas, que pertenecen a otro nivel y desconocen las peculiaridades de cada uno de los

distintos dispositivos externos. Cuando un *driver* es "llamado" para producir un servicio, debe conocer qué controles y actividades se deben realizar y en qué orden se debe enviar la secuencia de operaciones al controlador del periférico. Una instrucción del *driver* permite la copia de un comando en un registro de control asociado al periférico; este programa, por lo general, espera que el controlador complete la operación comandada. Conociendo que muchos controladores aceptan más de un comando por vez, es posible que la ejecución del *driver* se suspenda hasta que el controlador envíe un aviso de "ya termine" (este aviso suele entregarse mediante una señal de interrupción). Además, el *driver* debe contener una serie de instrucciones que permitan la evaluación final de la o las operaciones realizadas y debe poder "amar" un informe de errores (en caso de que los haya) para el programa llamador de nivel superior.

13.3.7.1 Comandos de los periféricos

Los periféricos son controlados por configuraciones de bits que hacen referencia a "qué es lo que el dispositivo debe hacer". Los comandos se pueden clasificar en:

- Comandos de verificación. Son aquellos que permiten evaluar si el periférico está:
 - Prendido o apagado.
 - Ocupado (p. ej., la impresora está imprimiendo).
 - No operable (p. ej., la impresora no tiene papel).
 - En error de operación (p. ej., el papel se atascó en la impresora).
- Comandos de control. Son aquellos que ordenan al periférico:
 - Prenderse o apagarse.
 - Saltar de página en una impresora.
 - Leer (operación de entrada).
 - Escribir (operación de salida).

13.4 Modalidades de entrada/salida

El sistema operativo cuenta con programas que gestionan las transferencias de entrada/salida pero esta vez en un nivel superior o, si lo queremos ver de otro modo, un nivel más abstracto. Estos programas realizan funciones comunes a todos los dispositivos periféricos, más allá de las particularidades de cada uno de ellos; por ejemplo, la medida real de un sector de disco no es importante en este nivel, sino que se asume un tamaño lógico que involucra a uno o varios de ellos como bloque de información. Desde el punto de vista de un programa de aplicación, un proceso que ejecuta la CPU actualmente puede ir escribiendo datos en un "área de almacenamiento" hasta completar el bloque, en este momento recién se llama al servicio para realizar la transferencia del bloque completo; si seguimos con el mismo ejemplo, a su vez, este servicio pide la intervención del *driver* que gestiona la transferencia de unidades menores (sectores) "a la medida" del disco que se ha de acceder.

Un programa de aplicación puede hacer referencia a datos organizados en un archivo, la forma en que un programa llama a un archivo, por ejemplo, si se identifica con los parámetros unidad lógica, directorio, nombre. Esta referencia es la misma para un disco rígido que para un disco creado en un *pen drive*, o sea que es de un nivel más abstracto y por completo independiente de la organización particular de cada unidad, que es bastante distinta (p. ej., un *memory stick* no tiene platos ni cilindros, ni sectores).

Por lo tanto, el mapeo del nombre simbólico de un archivo a su dispositivo físico correspondiente también se gestiona en este nivel. Los programas de aplicación llaman a los programas



Un *memory stick* es un dispositivo de almacenamiento que utiliza memoria tipo flash, que lo convierte en extraíble y no volátil, y permite un cómodo traslado de la información; por ejemplo, en este momento toda la información de este libro se haya en un *memory stick* formateado por Windows XP con FAT 32.

de E/S del sistema operativo mediante una "llamada al sistema", que puede expresarse de distintas maneras según el sistema operativo que se utilice.

La relación entre el programa de aplicación en estado de ejecución como peticionario del servicio de transferencia y el hardware de entrada/salida como su proveedor se muestra en el siguiente ciclo:

Aplicación → Programa de E/S del nivel abstracto → *Driver* → Hardware

Se indicó que una transferencia elemental es un mensaje que migra de un emisor a un receptor. Una transferencia completa involucra varias transferencias elementales.

Para llevar a cabo la transferencia completa se pueden identificar modos o métodos:

- Transferencia controlada por programa.
- Transferencia iniciada por interrupción.
- Transferencia con acceso directo a memoria.
- Transferencia a través de un procesador IOP o modo canal.

13.4.1 Transferencia controlada por programa

Cuando la transferencia se denomina **controlada por programa** (en inglés, *program I/O*; en siglas PIO), la que ejecuta el programa de E/S es la CPU, que para comunicarse utiliza un bloque hardware denominado interfaz. De esta manera, es la CPU la que controla el acceso a memoria para ubicar el dato, a través de sus propios registros internos (p. ej., el acumulador) y los de la interfaz (p. ej., un número de puerto). En esta modalidad la CPU debe verificar el estado de la interfaz a través del **puerto de control** en forma continua.

Suponga una transferencia de entrada desde un periférico hacia memoria. La CPU ejecuta un comando de verificación del estado de la interfaz; si ésta no se encuentra disponible, la CPU permanece ejecutando un bucle de consulta hasta que el resultado de la verificación sea afirmativo. El puerto de control de la interfaz es actualizado por el periférico, así que si éste es lento respecto de la CPU, el indicador de su estado se consultará miles de veces antes de que la CPU pueda salir de la ejecución del bucle. Así, el tiempo de CPU que se pierde en la consulta es mucho y se genera un mal aprovechamiento del recurso.

Cada instrucción de entrada/salida hace referencia con un grupo de bits a un registro de la interfaz asociada al controlador del periférico, y con otro grupo de bits a un registro general de la CPU, o bien a una posición de memoria. La CPU ejecuta el programa de E/S en el que una instrucción genera sobre el bus de direcciones el número que identifica el **puerto de dato**, esto es, decir el área de almacenamiento que actúa de nexo para lograr la transferencia. Si la dirección asignada al puerto pertenece a un área separada de la memoria, se indica que se utiliza la técnica de direccionamiento "aislada". Las instrucciones que provocan una lectura o una escritura (*input* o *output*) para una transferencia de E/S son distintas a las instrucciones que provocan esa operación en la memoria (*read* o *write* o R/W). En cambio, si la dirección asignada al puerto está en el área de espacio de direccionamiento de memoria se indica que se utiliza la técnica de **mapeo de memoria** (*memory mapped*). En este caso no hay instrucciones separadas para la lectura y la escritura de memoria y de E/S y sólo se reconoce su diferencia por el valor de la dirección.

Esta modalidad de entrada/salida también se conoce como "por sondeo" que significa que es la CPU quien se ocupa de verificar el estado del periférico y no el periférico el que interrumpe a la CPU.

13.4.2 Transferencia iniciada por interrupción

Una transferencia iniciada por interrupción (*interrupt driven*) proporciona una mejora al procedimiento anterior, que implica que la CPU, que es quien solicita la transferencia, no verifique de manera continua el estado de un dato listo para transferirse desde el dispositivo externo, sino que la interfaz asociada genere un aviso que indique que está preparada para transferir. En este caso se expresa que la interfaz provoca una interrupción y, por supuesto, esto se indica con un bit de estado denominado **bit de interrupción** (*Interrupt Request Query* o IRQ). Por cada ciclo de ejecución de una instrucción la CPU consulta por interrupciones externas; si es así, guarda la información suficiente del estado del proceso que va a interrumpir, para luego poder reanudarlo.

Entonces la CPU queda en condiciones de ejecutar el servicio de tratamiento de interrupciones que permitirá la transferencia entre el dispositivo y un área de memoria interna. Finalizada la transferencia se reanuda el proceso suspendido.

Cuando un sistema operativo admite multiprogramación o *multitask* (varias tareas compartiendo los recursos) y se requiere una transferencia de datos desde un dispositivo externo para continuar con el procesamiento del programa en ejecución, se genera una llamada al sistema y se coloca el programa en ejecución en estado "en espera"; así, la CPU sigue con otro proceso. El servicio convocado se coloca en una cola de ejecución y cuando detecta por interrupción que el dispositivo ya tiene los datos solicitados y está listo para la transferencia, almacena los contenidos de los registros de CPU actuales en la memoria (resfuerzo de contexto), y ejecuta el servicio de E/S que atiende este tipo de transferencia.

13.4.3 Transferencia con acceso directo a memoria

Las interfaces pueden conectarse a la memoria a través del controlador de acceso directo a memoria (en inglés, *direct memory access*; en siglas, DMA). Los DMA están asociados a dispositivos rápidos y que transfieren la información en bloques, grupos de bytes.

Supongamos que una transferencia se realiza sin DMA, en ese caso el controlador de un disco lee el soporte bit tras bit hasta completar su buffer; al finalizar esta actividad en la CPU se debería procesar el programa de E/S que respete el algoritmo descripto antes. Por cada palabra transferida del buffer a la memoria principal vía la interfaz, el programa debe actualizar una variable de cuenta y otra variable puntero; la primera variable permite controlar la cantidad de transferencias elementales hasta haber leído todo el buffer, el puntero se incrementa cada vez para permitir la escritura en la siguiente palabra. La ejecución del programa hace que la CPU gaste todo este tiempo hasta lograr la transferencia completa. Si la gestión la lleva a cabo un controlador con acceso directo a memoria, la CPU sólo interviene indicándole como parámetros la cantidad de palabras a transferir, la posición inicial de la palabra en memoria interna y la dirección del dispositivo, y queda liberada para realizar otra actividad.

Cuando un maestro (p.ej., la CPU) accede a que un esclavo (p. ej., un controlador DMA) utilice el bus, el primero activa el estado de "alta impedancia" de su *buffer* triestado "desconectándose" y liberando el bus.

La técnica consiste en relacionar la funcionalidad DMA con el bus que conecta la CPU con la memoria principal o interna, y que ese controlador establezca la relación memoria-interfaz sin intervención de la CPU. El uso del bus se comparte de manera alternativa. El diálogo entre la CPU y el controlador DMA se establece utilizando ciertas líneas del bus común, descriptas en la sección "Buses de entrada/salida". Dicho diálogo se enumera a continuación.

13.4.3.1 Relación CPU-DMA para activar la transferencia

La CPU se relaciona con el DMA como lo hace con cualquier unidad interfaz, esto es, le envía a través del bus de direcciones un address que reconocerá como propio, y así activará su operación. Además, la CPU debe enviar al controlador DMA la dirección de memoria principal donde debe tomar los bits a escribir o donde debe almacenar los bits leídos (caso de escritura o de lectura), la cantidad de palabras de memoria a transferir (total de palabras del bloque a transferir), una orden de lectura o de escritura (según sea el caso) y un comando que se denomina de inicialización para que el DMA comience la transferencia.

Una vez enviada toda esta información la CPU se desentiende de la E/S, deja el programa que estaba ejecutando y requería la E/S en suspenso, y comienza a procesar otra tarea, si la hay.

13.4.3.2 Relación CPU-DMA durante la transferencia

El DMA consulta el estado del dispositivo; cuando éste se encuentra disponible, le envía al DMA una señal de "disponible". El DMA envía a la CPU una señal de que "solicita" el uso del bus común. Cuando la CPU termina la instrucción en curso atiende la señal y "se desconecta" del bus, liberándolo y dejándolo a disposición del DMA; luego envía al DMA una señal de "bus disponible". A partir de este momento, el bus es controlado por el DMA.

Ahora puede ocuparse de efectivizar la transferencia del bloque; para ello deja a la CPU inactiva todo el tiempo que dure la transferencia completa, o bien, el período que tarde en efectivizar una transferencia elemental para que la inactividad sólo implique un ciclo de memoria. Esta última modalidad se conoce como "robo de ciclo".

13.4.3.3 Relación DMA-dispositivo durante la transferencia

Cuando el DMA recibe la señal "bus disponible", el DMA envía a la memoria principal la dirección de la palabra a transferir a través del bus de direcciones y envía una señal al dispositivo que implique "transferencia". En el caso de una salida el DMA envía la señal de lectura a la memoria, que responde con la palabra sobre el bus. En el caso de una entrada y tras recibir la señal de "transferencia" el dispositivo coloca sobre el bus la palabra a escribir y luego el DMA genera la señal de "escritura" a la memoria principal. Por cada transferencia elemental el DMA decremente su registro de "cantidad de palabras a transferir" e incrementa su registro de "dirección de palabra a transferir". Luego verifica la señal "disponible" del dispositivo; si ésta se encuentra activada y la "cantidad de palabras a transferir" no llegó a cero, el DMA inicia la transferencia siguiente. Si opera en modo "robo de ciclo" inhabilita la señal "bus disponible", que posibilita el coloquio DMA-CPU. Cuando el dispositivo envía la señal "dispositivo disponible", el DMA vuelve a enviar la señal "solicita" y se repite el procedimiento.

13.4.3.4 Fin de la transferencia con acceso directo a memoria

Cuando el registro de "cantidad de palabras a transferir" llega a cero el DMA inhabilita la señal "solicita" y genera una señal de "interrupción" para la CPU. Cuando la CPU testea la señal "interrupción" controla el registro "cantidad de palabras transferidas".

Ejemplo de señales del bus para la utilización del DMA:

- **DREQ (DMA Request).** En teoría es similar a las líneas *IRQ*, pero se utiliza para solicitar acceso directo a memoria. Hay tres de ellas, señaladas *DREQ1* a *DREQ3*.
- **DACK (DMA Acknowledge).** Se utiliza para acusar recibo de la petición DREQ correspondiente. Hay cuatro, señaladas *DACK0* a *DACK3*, aunque el mecanismo DMA sólo utiliza las tres últimas; *DACK0* se utiliza en el XT para señalar un ciclo de refresco de memoria.
- **MEMR (Memory Read).** Cuando se activa, esta señal indica a la memoria conectada al bus que escriba los datos en el bus de datos.

- MEMW (*Memory Write*). Cuando se activa, indica a la memoria que almacene los datos situados en el bus de datos.
- T/C (*Terminal Count*). Sirve para señalar que el controlador DMA alcanzó el final de una transferencia.

13.5 Resumen

Un bus es un enlace de comunicación que permite la transferencia de información entre dos o más dispositivos (en éste último caso alternadamente); esto significa que es un recurso hardware compartido. Las computadoras poseen buses organizados en distintos niveles. El primer nivel corresponde al bus que conecta la CPU y la memoria principal y se denomina bus del sistema.

Un bus de sistema puede conectarse directamente con módulos de entrada/salida constuyéndose en un sistema de bus único; sin embargo, en las arquitecturas abiertas actuales este esquema no se concibe debido al gran número de dispositivos, cuyas solicitudes debería coordinar produciéndose la espera de pedidos, el alto tráfico del bus no le permitiría a la CPU acceder a memoria principal tan rápidamente como lo haría si no lo compartiese.

Una solución más eficiente consiste en utilizar uno o más buses "de expansión" que se encarguen de enviar información entre el bus de sistema y los dispositivos de entrada/salida, de modo tal de lograr una independencia entre las transferencias CPU-Memoria y las relacionadas con la entrada/salida.

Por ejemplo, en una arquitectura que incluye acceso a red se necesitará una placa adaptadora de red y se puede contar con una placa aceleradora de gráficos y controladores para periféricos SCSI. Aquellos dispositivos de velocidad menor pueden conectarse al bus de expansión, que utiliza una interfaz para adaptar el tráfico entre el bus de expansión y un bus de alta velocidad que "acerque" en términos de tiempo, a los dispositivos de prestaciones más altas a la CPU a la vez que los independiza.



El primer bus de expansión fue el bus ISA utilizado con el CPU 8088, el DMA 8237 y el controlador de interrupciones 8259. Distinguía dos ciclos de bus: el controlado por CPU y el controlado por DMA. En el primer caso, la CPU generaba direcciones físicas de memoria principal o de un puerto y controlaba el sentido de la transferencia en el bus de datos. En el segundo caso, el DMA solicitaba el control del bus y cuando lo obtenía establecía la relación memoria a DMA controladora.

Actividades por cada ciclo de bus:

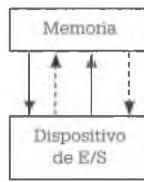
1. Lectura de Datos/Instrucciones del CPU.
2. Escribir datos desde el CPU a memoria.
3. Leer datos desde un puerto de E/S para la CPU.
4. Escribir datos desde la CPU a puerto de E/S.
5. Leer datos desde el controlador de interrupción a la CPU.
6. Leer datos de memoria y escribirlos en controladora DMA.
7. Leer datos de controladora y escribirlos en memoria.

13.6 Ejercicios propuestos

1. Realice un diagrama de flujo para representar la actividad HANDSHAKE

En el siguiente esquema indique con el número que corresponda el siguiente "diálogo"

1. Solicitud de Lectura REQ
2. Reconocimiento de Solicitud ACK
3. Señal de Datos Listos DATA READY
4. Reconocimiento de Datos Listos ACK



- 2- Realice un diagrama de flujo para representar una entrada salida programada
- 3- Dibuje una tabla de actividades organizadas en secuencia y a una por fila que represente la dinámica entre la CPU y el DMA o entre el DMA y el dispositivo según corresponda para cada una de las etapas de la transferencia que se describen a continuación:

- a) Relación CPU-DMA para activar la transferencia.
 b) Relación CPU-DMA durante la transferencia
 c) Relación DMA-dispositivo durante la transferencia.
 d) Fin de la transferencia con acceso directo a Memoria

- 4- En su computador personal ingrese al setup y responda:
 ¿Cuántas líneas IRQ puede visualizar?
 ¿Cuáles están disponibles y cuáles ya han sido configuradas?
 ¿En qué dirección de memoria se encuentra el puerto serial y cuál es su nivel de IRQ?
 ¿Cuál es el IRQ que corresponde al DMA?
 5- Indique el nombre en inglés y la función de cada uno de los dispositivos que corresponden a las siguientes siglas:
 SCSI
 PCI
 IDE
 SATA2
 AGP
 DACK
 DRQ

Responda el siguiente cuestionario

¿Qué es la señal de CLK?

¿Qué es un ciclo de bus?

¿Qué diferencia existe entre una transferencia de entrada/salida sincrónica y asincrónica?

¿En qué casos se utiliza una y otra?

¿Cuál es el motivo por el que una transferencia asincrónica requiere un protocolo de comunicación?

¿Qué es una E/S mapeada?

¿Qué es un E/S aislada?

¿En qué unidades se expresa la velocidad de transferencia?

¿Cuántos bits se transfieren por vez en una transferencia serie?

¿Por qué líneas del bus se envían señales de request y acknowledgment?

¿Cuáles son las unidades principales de un controlador y cuál es su función?



13.7 Contenido de la página Web de apoyo

El material marcado con asterisco (*) sólo está disponible para docentes.

Resumen gráfico del capítulo

Autoevaluación

Video explicativo (01:39 minutos aprox.). Capítulos 12 y 13

Audio explicativo (01:39 minutos aprox.). Capítulos 12 y 13

Evaluaciones Propuestas*

Presentaciones*

14

Procesadores avanzados

Contenido

14.1 Introducción	318
14.2 Paralelismo a nivel instrucción	319
14.3 Paralelismo a nivel arquitectura.....	322
14.4 Descripción de microprocesadores avanzados	326
14.5 Resumen.....	339
14.6 Contenido de la página Web de apoyo.....	340

Objetivos

- Comprender el paradigma de procesamiento en paralelo a nivel instrucción.
- Comprender el paralelismo a nivel arquitectónico, según la taxonomía de Flynn.
- Comprender las características de los manuales de los fabricantes con procesadores avanzados.



En la página Web de apoyo encontrará un breve comentario de la autora sobre este capítulo.

14.1 Introducción

El propósito de este capítulo es presentar cada uno de los paradigmas de “ejecución en paralelo”. Cuando se habla de paralelismo, lo que se persigue es aumentar la velocidad de cómputo incrementando la cantidad de instrucciones que se ejecutan durante un intervalo. Para dar un ejemplo simplemente ilustrativo, en este primer acercamiento al problema planteado, piense en una medida de tiempo cualquiera, como el nanosegundo.

Se puede aspirar a que una computadora ejecute una instrucción en un nanosegundo ($1/1000\,000\,000\text{ seg}$, es decir en una de mil millones de partes de segundo) o se puede producir el mismo efecto, un poco grandilocuente, con *1000 “unidades de ejecución”* que ejecuten una instrucción en $1 / 1000\,000\text{ seg}$, o sea una de un millón de partes de segundo.

Cuando se habla de procesamiento en paralelo, la idea central es multiplicar la cantidad de “unidades de ejecución” que operen concurrentemente, ya sea ejecutando distintas instrucciones en paralelo o ejecutando distintas etapas de instrucciones simultáneamente; sean estas “unidades” tanto procesadores (en cuyo caso hablamos de arquitecturas en paralelo basadas en multiprocesadores), o sean éstas “unidades” dentro de un procesador, por ejemplo, dos unidades de coma flotante, pueden realizar dos operaciones simultáneas (en cuyo caso hablamos de paralelismo a nivel ejecución de instrucción).

Como hemos visto en capítulos anteriores, la ejecución de una instrucción se puede dividir en etapas y, a su vez, cada etapa puede ser llevada a cabo por unidades físicas independientes. Esto permite que la ejecución de “distintas etapas” de “distintas instrucciones” se procesen simultáneamente; esta técnica se conoce como ***pipelining*** o ***segmentación de instrucciones***. Cabe aclarar que esta “segmentación” nada tiene que ver con la segmentación de memoria y no deben confundirse los términos.

Como ya dijimos, cuando el objetivo es lograr mayor velocidad a nivel ejecución de instrucciones, es lógico pensar que esto se logra aumentando la frecuencia del reloj que regula el secuenciamiento de microinstrucciones. Esta es una buena alternativa, sin embargo, no es la única y a veces no es la mejor.

Pensar que dos instrucciones consecutivas se puedan ejecutar en la mitad del tiempo, acelerando su ejecución secuencial, se puede lograr también utilizando la misma frecuencia de reloj si ambas se ejecutan en unidades diferentes. Esta propuesta da lugar a múltiples técnicas que fundamentan el paralelismo de actividades, este capítulo describe las más utilizadas y propone el estudio concreto de las características de dos microprocesadores del mercado actual.

Conviene introducir el capítulo con algunos conceptos, por ejemplo, ***segmentado*** o ***pipelining***: Comenzaremos con el ejemplo de una línea de producción de gaseosas, considerando tres unidades de ejecución que se encargan de las siguientes actividades: Llenar la botella con gaseosa, tapar la botella y colocar la etiqueta con la marca.

El proceso de embotellamiento en serie implica que para tres botellas se cumplan nueve etapas equivalentes a nueve unidades de tiempo, primero se llena la botella 1, luego se la tapa, luego se coloca la etiqueta, luego se llena la botella 2, luego se tapa, y así hasta el final.

La propuesta segmentada sugiere que las botellas estén en una cola y pasen delante de cada “unidad de ejecución” una tras otra, así cuando la botella 1 está en la etapa de colocación de etiqueta, la botella 2 está en la etapa de tapar la botella y la botella 3 está en la etapa de llenar con gaseosa. Se puede decir que esta línea de producción está operando “en tres botellas simultáneamente”; si este ejemplo se lleva a instrucciones se puede decir que el ***grado de paralelismo*** es igual a tres. Este modelo no implica que se haya “tripli-

cado" la velocidad de ejecución pues hay momentos en los que no todas las botellas están en la línea de ejecución. Este ejemplo se puede ilustrar así:

Tabla 14-1. Ejemplo de segmentación en una embotelladora.

Unidades de Tiempo	Colocar Etiquetas	Tapar botellas	Llenar botellas	Cola Llena		
				B1	B2	B3
1			B 1			
2		B1	B2	B3		
3	B1	B2	B3			
4	B2	B3				
5	B3					

Cola Vacía

El "tiempo de ejecución" fue de 5 en lugar de 9, con lo cual se puede valorar la "disminución" del tiempo de ejecución. Si duplicamos la capacidad de producción de la fábrica, duplicando las líneas de procesamiento, es decir, generando una nueva cadena de ejecución decimos que por cada unidad de tiempo se procesan simultáneamente varias cadenas. Esta nueva propuesta se denomina **superescalar**.

Tabla 14-2. Línea de ejecución 1 y 2.

Línea de ejecución 1

Unidades de Tiempo	Colocar Etiquetas	Tapar botellas	Llenar botellas	Cola Llena		
				B1	B2	B3
1			B 1			
2		B1	B2	B3		
3	B1	B2	B3			
4	B2	B3				
5	B3					

Cola Vacía

Línea de ejecución 2

Unidades de Tiempo	Colocar Etiquetas	Tapar botellas	Llenar botellas	Cola Llena		
				B1	B2	B3
1			B 1			
2		B1	B2	B3		
3	B1	B2	B3			
4	B2	B3				
5	B3					

Cola Vacía

De este modo, se procesan seis botellas en 5 unidades de tiempo. En la medida que se aumenta la **escalabilidad**, también se produce el efecto de disminuir el tiempo de ejecución.

14.2 Paralelismo a nivel instrucción

Como vimos en el capítulo Micróprocesadores, uno de los métodos usados para incrementar el rendimiento es iniciar la búsqueda de la instrucción en memoria y su decodificación antes de que la ejecución de la instrucción anterior haya terminado. Esta técnica se encuadra dentro de lo que se conoce como paralelismo a nivel instrucción, que consiste en la simultaneidad de ejecución por etapas. La cantidad de etapas definidas se denomina grado de paralelismo.

Tabla que muestra la ejecución en 5 etapas.

Tabla 14-3. Pipeline de 5 etapas.

<i>Etapa 1</i>	<i>Etapa 2</i>	<i>Etapa 3</i>	<i>Etapa 4</i>	<i>Etapa 5</i>	
Instrucción 0					
	Instrucción 1	Instrucción 1	Instrucción 1	Instrucción 1	Instrucción 1
		Instrucción 2	Instrucción 2	Instrucción 2	Instrucción 2
			Instrucción 3	Instrucción 3	Instrucción 3
				Instrucción 4	Instrucción 4
					Instrucción 4
					Instrucción 4

Para dar un ejemplo más detallado de las unidades que "operan" cada etapa, se indican las del primer procesador *IA-32*, donde para cada una de las unidades se observa un verbo distinto, resaltado y entre comillas, que indica que la función que realiza esa etapa es distinta de las otras.

- La unidad de gestión de memoria (*Memory Manager Unit*) **traduce** direcciones lógicas en direcciones absolutas.
- La unidad de interfaz del bus (*Bus Interface Unit*) **accede** a memoria y a dispositivos de E/S.
- La unidad de precarga de instrucciones (*Code Prefetch Unit*) recibe 2 bytes del bus y los **precarga** en la cola de 16 bytes presentada en el capítulo Microprocesadores, en el apartado Número de instrucciones.
- La unidad de decodificación de instrucciones (*Instruction Decode Unit*) **decodifica** el código de operación y lo **traduce** a microcódigo.
- La unidad de ejecución (*Execution Unit*) **ejecuta** el microcódigo.

Durante el procesamiento, en el *pipeline* puede ocurrir que el resultado de una instrucción "dependa" del resultado de la anterior, razón por la que sería necesario concluir la instrucción n para ejecutar la instrucción $n+1$. Este conflicto se conoce como "dependencia de datos". Se puede anticipar la presencia de este conflicto que interfiere en la evolución del *pipeline*, pues está previsto y se puede resolver, por ejemplo, retrasando la ejecución de una o más etapas, o sea que se resuelve a nivel ejecución. Otra forma de resolverlo es a nivel compilación, donde se puede aplicar una técnica que produce un reordenamiento de las instrucciones sin afectar la lógica del programa. A esta técnica se la denomina "ejecución fuera de orden". Observe el ejemplo:

Si un programa resuelve las siguientes operaciones aritméticas:

- 1) $c = a + b$
- 2) $m = c^2$ donde "m depende de c"
- 3) $d = e + (a * b)$
- 4) $z = x + y$

Es lo mismo que ejecutar

- 1) $c = a + b$

$$3) d = e + (a * b)$$

$$4) z = x + y$$

$$2) m = c^*2$$

El cambio en el orden de ejecución no afectó la lógica del programa y otorga el tiempo necesario para que si se ejecutan las tres primeras reordenadas, al ejecutar la cuarta, $m = c^*2$, no haya espera, pues ya se calculó el valor de c sin producir retardos.

Como sabemos, cada etapa es llevada a cabo por una unidad específica del procesador. Por lo tanto, si por cada etapa se duplica la cantidad de unidades de ejecución, se indica que el diseño es superescalar, o sea que se crean múltiples vías de procesamiento paralelo a nivel instrucción. En la tabla que se exhibe a continuación, al duplicarse cada unidad de ejecución en cada etapa se logra el efecto de duplicar la cantidad de instrucciones por ciclo de reloj.

La microarquitectura P5 tenía dos unidades de coma fija o unidades de enteros superescalares y una unidad punto flotante; en la microarquitectura P6 una de las mejoras fue incluir una FPU superescalar.

Tabla 14-4. Pipeline superescalar de 5 etapas

Etapa 1	Etapa 2	Etapa 3	Etapa 4	Etapa 5	
Instrucción 0					
Instrucción 1					
	Instrucción 2				
	Instrucción 3				
	Instrucción 4				
	Instrucción 5				
	Instrucción 6				
	Instrucción 7				
	Instrucción 8				
	Instrucción 9				

Durante la ejecución, las múltiples instrucciones se leen y pasan a un planificador, que determina cuáles de ellas se pueden ejecutar en paralelo. Aquí la mayor complejidad de diseño de CPU se relaciona con el planificador, que a su vez requiere una afluencia constante de instrucciones para analizar, que, por lo tanto, deben estar disponibles en una memoria caché (memoria ultrarrápida) asociada al microprocesador. En la medida en que el paralelismo aumenta, se incrementan los posibles conflictos de dependencias o saltos inesperados. Mientras

Escalar:

Un procesador escalar en la taxonomía de Flynn es de tipo SISD.

Superescalar:

Un procesador superescalar en la taxonomía de Flynn es de tipo MIMD. Cuando nos referimos a una microarquitectura superescalar, indicamos que implementa paralelismo a nivel instrucción y a nivel flujo..

que la ejecución fuera de orden está orientada a resolver el conflicto de dependencia de datos, la técnica de predicción de bifurcación procura predecir qué "camino" o bifurcación tomará una instrucción de salto condicional. Intel denomina a este camino rama de ejecución. La ejecución especulativa de cada rama consiste en ejecutar todas las trayectorias posibles, aunque luego del salto condicional haya que desechar alguna de ellas esta técnica de tratamiento de saltos se denomina de flujos múltiples.

Hasta ahora nos ocupamos del paralelismo a nivel de instrucción, que se conoce como granulado fino, en éste el paralelismo se aplica a cada etapa de ejecución de la instrucción, y por lo general es el hardware el que detecta instrucciones y datos no dependientes, y planifica su ejecución en paralelo.

Los multiprocesadores explotan el paralelismo a nivel de proceso, o paralelismo de granulado grueso; en estos casos el paralelismo lo resuelve el compilador y las funciones del sistema operativo. Es ahora cuando se puede incluir el concepto de *thread* o hilo de ejecución; el paralelismo a nivel de hilo de ejecución (*Thread Level Parallelism* o TLP) tiene como objetivo incrementar el número de programas individuales que una CPU pueda ejecutar en forma simultánea. En términos más específicos, los hilos son rutinas concurrentes que comparten variables globales y el mismo espacio de direccionamiento; su mejora en el rendimiento global se apoya en la habilidad de solapar cálculos con operaciones de entrada/salida. En el contexto de diseño de un solo chip, las dos metodologías principales usadas para lograr el TLP son el multiprocesamiento a nivel de chip (*Chip-Level Multiprocessing* o CMP-) y el multihilado simultáneo (*Simultaneous Multithreading* o SMT).

Los multiprocesadores en un solo chip, en general denominados procesadores de n núcleos, están extendiéndose en el mercado de manera muy notable. En estos multiprocesadores se hace necesario generar los hilos de código de forma que puedan ejecutarse de manera simultánea n hilos en los n núcleos. Se crearon herramientas software que permiten realizar las "particiones" del código sin necesidad de crear, destruir o comunicar distintos hilos, por lo menos en una forma explícita; la más popular de estas herramientas es openMP.



Retrieve de memoria es el tiempo que se tarda en recibir información solicitada a memoria.

14.3 Paralelismo a nivel arquitectura

Las arquitecturas basadas en microprocesadores tienen características de rendimiento limitadas en el campo de los cálculos científicos de la física cuántica o de la realidad virtual (por nombrar sólo dos aspectos en los que las necesidades de procesamiento los superan ampliamente). Para lograr un alto rendimiento, los microprocesadores no sólo deben ejecutar las instrucciones de una manera más rápida, sino que también es preciso que ejecuten más instrucciones por ciclo de reloj. La ejecución en paralelo requiere técnicas de predicción de saltos o especulación de datos; en este último caso, para aliviar el *retrieve* de memoria. Cuando todo esto no alcanza para lograr el rendimiento deseado, se debe llevar el "paralelismo a nivel procesador" a un grado mayor que se alcanza en las arquitecturas denominadas paralelas.

Una computadora con más de un procesador ejecutando en paralelo y de forma coordinada puede pertenecer (salvo la primera) a una de las siguientes categorías determinadas por la taxonomía de Michael J. Flynn. Ésta es la clasificación más difundida en arquitecturas que soportan el tratamiento de instrucciones concurrentes; en el cuadro siguiente se brindan las características de cada clasificación:

14.3.1 Taxonomía de Flynn. Una clasificación de arquitecturas paralelas

Tabla 14-5. Clasificación de arquitecturas paralelas.

			instrucciones	datos
SISD	<i>Single Instruction Single Data</i>	von Neumann	1	1
MISD	<i>Multiple Instruction Single Data</i>	Paralelismo redundante	Muchas	1
SIMD	Varios flujos de datos	Vectorial	1	Muchos
MIMD	Varios flujos de instrucciones en diversos procesadores con varios juegos de datos	Memoria compartida	Muchas	Muchos
		Memoria distribuida		

Las máquinas RISC mostraron que es posible una mejora en el rendimiento por un factor de entre 10 y 20, como consecuencia de la eliminación del micropograma, pero aún son máquinas von Neumann, con todas sus limitaciones.

14.3.1.1 SISD (*Single Instruction Single Data*)

Es el utilizado en el modelo tradicional de computación secuencial mostrado en el capítulo Diseño de una computadora digital, con procesador único, donde la CPU recibe una sola secuencia de instrucciones que operan en una única secuencia a los datos y no hay paralelismo; se trata de la computadora secuencial clásica de von Neumann con un flujo de instrucciones y un flujo de datos, que realiza una operación por vez.

14.3.1.2 MISD (*Multiple Instruction Single Data*)

Las computadoras MISD se utilizan para ejecutar distintos procesos sobre un único flujo de datos. Cada proceso se desarrolla en un procesador con su propia unidad de control y accediendo a una memoria común a todos. El paralelismo está dado porque cada procesador resuelve una parte del problema en forma simultánea con los otros, en cada procesador se ejecuta uno de los tantos módulos algorítmicos que contribuyen a la ejecución total del problema computacional a resolver.

14.3.1.3 SIMD (*Single Instruction Multiple Data*)

Este tipo de computadoras es de procesamiento paralelo sincronizado. Cada procesador opera en sincronismo con los demás, ejecuta la misma secuencia de instrucciones sobre diferentes flujos de dato. Las máquinas con arreglos de procesadores como ICL DAP (*Distributed Array Processor*) y las computadoras vectoriales son de arquitectura SIMD. El ejemplo típico es una aplicación donde haya que ejecutar siempre las mismas operaciones sobre distintas matrices de datos.

14.3.1.4 MIMD (*Multiple Instruction Multiple Data*)

Este tipo de computadora es de procesamiento paralelo asincrónico. Para cada procesador se asigna una secuencia de instrucciones y datos. Si se tienen N procesadores, se cuenta con N secuencias de instrucciones y N secuencias de datos. También se utiliza el término MIMD a nivel microarquitectura, en este caso cada unidad de ejecución opera como un procesador independiente.

Dentro de la clasificación MIMD se puede presentar otra en función de la relación de los procesadores con la memoria:

Sistemas de alto rendimiento computacional (*High Power Performance Computer* o HPC) significa una gran demanda de procesamiento de datos en procesadores, memoria y otros recursos de hardware, donde la comunicación entre ellos debe ser muy veloz; el término se relaciona con el término supermáquinas.



Los procesadores vectoriales son aquellos en los que una sola instrucción opera en forma simultánea sobre un conjunto grande de datos.

Tabla 14-6. Clasificación MIMD en función de la relación de los procesadores con la memoria.

SUBCLASIFICACIÓN	DETALLE	CARACTERÍSTICAS DESTACABLES
Memoria compartida	El espacio de direccionamiento se comparte y se accede a él por un bus común, por lo que se evita el acceso simultáneo a una zona de memoria compartida.	Al tener una memoria compartida, todos los procesadores consideran el mismo tiempo de latencia, que debe ser el más bajo posible para agilizar el procesamiento. Esto encarece el precio de la memoria a incluir en estos sistemas. Al aumentar el número de procesadores que accede a memoria, se tiende a la generación de cuellos de botella, debido a la pretensión de accesos simultáneos. En estas arquitecturas los procesadores comparten el mismo sistema operativo.
Memoria distribuida	Los procesadores comparten información por medio de mensajes o sistemas de procesamiento paralelo en masa (MPP o <i>Multiple Parallel Processors</i>).	Tienen alta escalabilidad, esto es, partiendo de un sistema básico se puede agregar más memoria y procesadores. Cada procesador opera con diferentes programas partes de él y puede ser administrado por distintos sistemas operativos.
Memoria compartida y distribuida	Los buses a memoria son independientes. Un procesador puede acceder al espacio de direccionamiento de otro, al que considera propio, o, expresado de otra manera, se constituye un espacio global.	Escalabilidad. No se producen cuellos de botella debido a que están interconectados por dispositivos de alta velocidad.

Tabla 14-7. Resumen de sistemas paralelos.

COMPUTADORAS PARALELAS	MIMD	MULTIPROCESADOR Información compartida en memoria compartida	DSM-NUMA	DSM-SVM	
				DSM-COMA	
				DSM-nccNUMA	
				DSM-ccNUMA	Servidores Origin Silicon Graphics
		MULTICOMPUTADORA Información compartida por paso de mensajes	UMA	SMP Escalable	
				SMP bus	HP-UX SMP
	SIMD	Computadoras vectoriales			NEC SX-9
		Máquina de conexión			
	MISD				
	NoW				
REDES DE COMPUTADORAS*	Malla				
	Cluster	Cluster simple			
		Constelación			

*No se profundiza el concepto de redes de computadoras por exceder el interés de esta obra; sin embargo, vale su mención por ser parte de los sistemas paralelos.

Sin realizar un estudio detallado enunciaremos características de algunos de los sistemas paralelos mencionados en el cuadro anterior.

Caso ejemplo MIMD Memoria Compartida DSM

Los sistemas de DSM (*Distributed Shared Memory*) tienen como objetivo no enviar mensajes en forma explícita entre los nodos componentes de un sistema distribuido. Asimismo, proveen la ilusión de un espacio global de memoria compartida por todos los procesadores; esto implica monitorear los accesos a las memorias remotas produciendo mensajes sobre la red de nodos. Los sistemas DSM se pueden diferenciar en aquellos manejados por hardware y por software.

Hay sistemas de DSM por hardware con un control de granularidad fina ejercido en el caché y con una red de interconexión propietaria.

Los sistemas DSM hardware se subclasifican en *Cache Only Memory Architecture* o COMA, que utilizan pseudomemorias caché para almacenar bloques de información compartidos, y *cache coherent non uniform memory access* o ccNUMA, en los que se utiliza el caché tradicional del procesador para alojar los bloques compartidos. El controlador de caché es el encargado de mantener la coherencia de sus contenidos, en general ayudado por un directorio que favorece la localización rápida de los bloques que son propiedad del nodo local.

Cuando las cachés almacenan datos privados utilizados por un solo procesador, reducen el tiempo de acceso al dato y permiten un mejor aprovechamiento de su ancho de banda; cuando almacenan datos compartidos por varios procesadores se produce el mismo efecto.

Los problemas que presentan deben ser solucionados por el sistema de coherencia de caché en el multiprocesador, pues, por ejemplo, si se tienen copias del mismo dato en diferentes cachés del sistema y se actualiza alguno de los valores, se tienen copias del mismo dato con distinto valor. En el caso de datos privados se debe posibilitar la migración, que consiste en posibilitar que un dato se pueda almacenar en una caché y utilizarse allí de forma transparente. En el caso de datos compartidos, se debe posibilitar la replicación en diferentes cachés.

Los sistemas de DVSM, o memoria virtual compartida distribuida, están compuestos por un cluster de workstations en las que se utiliza el hardware de protección de memoria virtual para implementar un control de granularidad mucho más gruesa; los nodos se interconectan con protocolos estándar de comunicación.

Caso ejemplo MIMD Memoria Compartida "SMP"

Los multiprocesadores simétricos (SMP o *Symmetric Multiprocessors*) son computadoras MIMD de memoria compartida que presentan un acceso simétrico a toda la memoria principal desde cualquier procesador, o sea que "el retardo en el acceso a cualquier posición de memoria es el mismo" con independencia del procesador desde el que se realice la operación.

Esta arquitectura se denomina acceso uniforme a memoria (*Uniform Memory Access* o UMA) y se consigue con una memoria que, además de estar compartida, está centralizada.

Un ejemplo es HP-UX SMP de 1 TB, utilizado para aplicaciones Business Intelligent de Oracle, y en rango medio para servers basados en bus denominados K-Class, que contienen un grupo de procesadores y una única memoria física con un bus que los interconecta.



Datawarehouse: Repositorio completo de datos de empresa, donde se almacenan datos estratégicos, tácticos y operativos, con el objeto de obtener información estratégica y táctica que ayuda al proceso de toma de decisiones gerenciales.



Datamining: Repositorio de datos que se almacenan con el objetivo de extraer conocimiento procesable implícito en ellos utilizando inteligencia artificial, análisis estadístico y patrones de comportamiento.

Caso ejemplo SIMD. Computadoras vectoriales

Se denomina vector a un conjunto de elementos escalares del mismo formato almacenados en posiciones consecutivas en memoria. Una instrucción que opera sobre un vector o una operación vectorial es la repetición del mismo cálculo sobre éstos. La instrucción vectorial especifica: el código de operación, que identifica la unidad de cálculo que realizará la operación; la dirección donde comienza el vector; el incremento respecto de esta dirección (considerada como base) para acceder a cada elemento de la estructura, y el tamaño de la estructura. El ejemplo típico para fundamentar el uso de estas computadoras es simple, piense en la suma de los elementos de dos vectores con el siguiente ejemplo de programación que muestra que X, Y y Z son arreglos unidimensionales en los que la suma de los elementos del vector Z a los elementos del vector Y genera los elementos del vector X; esta suma está embebida en una estructura repetitiva de tipo for-next que se produce tantas veces como elementos tengan los vectores.

```
FOR i= 1 to n
  X[i]:= Y[i]+Z[i]
  Next i
```

El compilador puede generar el código de máquina para lograr las sumas "simultáneas" si el hardware así lo permite. SIMD *single instruction* implica que el mismo grupo de instrucciones operará sobre los múltiples datos de cada vector.

El modelo lanzado por NEC Corporation es el SX-9, que, clasificado como supercomputadora, puede ejecutar 839 Tflops, utiliza 1 TB de memoria compartida y velocidad de interconexión de 128 GB/seg. Se utiliza en aplicaciones del campo científico, como predicción de tiempo, simulación ambiental y nanotecnología.

14.4 Descripción de microprocesadores avanzados

14.4.1 Descripción de la arquitectura Itanium

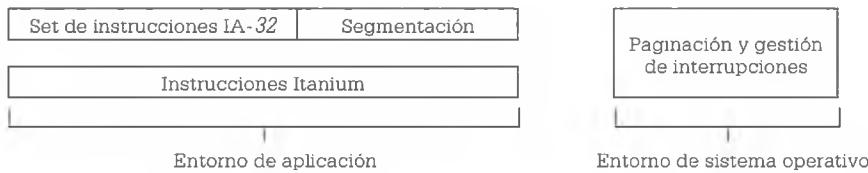
La arquitectura de Itanium fue diseñada con el fin primario de obtener un mayor paralelismo a nivel instrucción, pero con técnicas de predicción, especulación y predicción. Su arquitectura se constituye en lo que se denomina "una arquitectura de la bifurcación", o "de la rama", como la identifica Intel. Estas características apoyan las necesidades de aplicaciones de alto nivel, como *datawarehousing* y *datamining*. El objetivo secundario del diseño fue mejorar las limitaciones de arquitecturas tradicionales de la industria x86, orientado al direccionamiento de 64 bits necesarios en el almacenamiento de grandes bases de datos y servidores.

Cuenta con una nueva filosofía en el diseño de FPU que se traduce en un mayor rendimiento en aplicaciones de diseño de gráficos e ingeniería, y para cálculos avanzados (al igual que en AMD), como en AMD64, desarrollado a continuación. Admite compatibilidad de ejecución de aplicaciones de 32 bits, bajo la gestión de un sistema operativo que lo permita.

La arquitectura Itanium está provista de mecanismos que le permiten al código compilado gestionar "la forma de uso del procesador a nivel hardware" corroborando información del entorno en tiempo de ejecución que permita salvar fallas (p. ej., se puede evitar una penalidad asociada con un fallo en caché).

14.4.1.1 Modos de operación

El modo IA-32 soporta sistemas operativos de 32 bits, como el Windows XP. El modo Itanium soporta sistemas operativos basados en la arquitectura Itanium, como Windows Server 2008 (para sistemas basados en Itanium) y SUSE Linux Enterprise Server 9 (para Itanium Linux, Service Pack 4).



El procesador puede ejecutar instrucciones de IA-32 o del set Itanium en lo que se denomina entorno de ejecución Itanium. Para el cambio de transición entre uno y otro sólo se utilizan tres instrucciones. Cuando se ejecutan instrucciones IA-32 se conmuta a Itanium con la instrucción impe; cuando se ejecutan instrucciones Itanium se conmuta a IA-32 con la instrucción br.ia.m. Si se produce una interrupción, siempre se opera en entorno Itanium y se retorna del servicio con rfi.

14.4.1.2 Intel Itanium arquitectura EPIC

EPIC es la sigla de las palabras inglesas *Explicitly Parallel Instruction Computing*. Esta tecnología permite la ejecución en paralelo de la mayoría de las instrucciones ISA-64bit (*Instruction Standard Architecture*), que definen la arquitectura, además de contar con gran cantidad de registros internos. El paralelismo explícito significa que las dependencias de instrucciones se evalúan a nivel compilación, mientras que a nivel ejecución las instrucciones de distintas ramas de un salto condicional son marcadas por registros para ejecutarse en forma simultánea; además, se admite la carga especulativa de memoria. A continuación, detallaremos sus características

14.4.1.3 Paralelismo explícito

En el paralelismo explícito la tarea de lograr el paralelismo está a cargo del compilador que organiza de manera eficiente el código para su procesamiento y *hace explícito* el pedido para que de este modo el procesador pueda enfocarse en la ejecución simultánea de instrucciones de la forma más efectiva.

Registros internos: los principales son 128 registros de enteros y coma flotante, 64 registros de predicado de un bit y 8 registros rama o de salto.

14.4.1.4 Tabla de registros de propósito general (GRs)

Tabla 14-8. Registros GRs	
63	0
	GR0
	GR1
	GR2
	GR126
	GR127

Los registros generales que utilizan las aplicaciones son 128 de 64 bits cada uno, y se denominan GR0-GR127; cuando se ejecutan aplicaciones de 32 bits sobre datos enteros se utilizan los registros GR0-GR31. Entre éstos están incluidos los registros de segmento. Estos primeros 32 registros se denominan estáticos, mientras que los restantes se llaman en inglés *stacked general registers*; las aplicaciones los pueden utilizar para almacenar el "marco de pila de registros", que es un grupo de registros "programable" y sólo "visible" para la aplicación. Bajo Itanium no se utiliza la carga y descarga de registros en un stack de memoria para guardar cambios de registros de CPU.

Cuando se produce un cambio de contexto, por ejemplo, cuando se detecta un *call*, durante la compilación se renombran los registros (por eso la denominación de "registros programables"). Cuando se produce el *call*, se habilita un nuevo marco para el procedimiento llamado. El acceso a un registro se hace renombrando los identificadores lógicos en las instrucciones utilizando un registro base en los registros físicos. El proceso "llamado" puede utilizar los registros que quedan libres. Cuando se vuelve al proceso "llamador" el registro base vuelve a recargarse con el valor original anterior al llamado, y se renombran y reutilizan los valores que tenían los registros antes del "*call*".

14.4.1.5 Registros de coma flotante

Tabla 14.9. Registros FRs	
FR0	
FR1	
FR127	
81	0

Los registros para datos en coma flotante son 128 y se identifican como de FR0 a FR127. Cuando se ejecutan aplicaciones de 32 bits, tanto con instrucciones de coma flotante como con instrucciones multimedia, se utilizan los registros identificados como FR8 a FR31.

14.4.1.6 Los registros *predicado o PRs*

Son los utilizados para el tratamiento de saltos. Se trata de 64 registros de 1 bit y se identifican como PR0-PR63.

PRO	PR1	PR2	PR63
-----	-----	-----	------

Los predicados se utilizan en instrucciones de salto condicionado. O sea que puede ocurrir que la condición se cumpla o no. Esto genera dos caminos de ejecución o ramas posibles. Cuando el compilador encuentre una instrucción de salto condicionado en el código fuente, marcará todas las instrucciones que representan cada camino de la rama con un identificador en la instrucción único, llamado predicado (véase formato de instrucción). Cuando la CPU encuentre en tiempo de ejecución un salto predicado, comenzará a ejecutar el código de las dos ramas. No guarda el resultado mientras los valores de los registros del predicado no estén definidos. Una vez que se evalúa la condición, el procesador guarda un 1 en el registro del predicado (PRn) que corresponde a *destino verdadero* y 0 en los otros. Antes de guardar los

resultados, la CPU chequea cada registro PRn, y si el registro contiene un 1, las instrucciones son válidas, así que la CPU retirará la instrucción y almacenará el resultado, caso contrario rechazará el resultado.

14.4.1.7 Los registros "Rama" o BRs

Son 8 registros de 64 bits cada uno y se identifican como BR7-BR0.

Tabla 14-10. Registros BRs	
BR0	
BR1	
BR2	
BR2	
BR4	
BR5	
BR6	
BR7	
63	0

Hay características que mejoran el rendimiento del paralelismo a nivel instrucción (ILP o *Instruction Level Parallelism*). Respecto de esta última, a continuación se explican las técnicas de *especulación, predicción y predicción de saltos*.

14.4.1.8 Especulación

Como ya se ha mencionado en la sección paralelismo a nivel instrucción, la especulación es una técnica que, aplicada a instrucciones, consiste en ejecutar una instrucción antes de tener la seguridad de que su resultado se vaya a utilizar. Así, la ejecución en paralelo de las dos ramas de una sentencia de tipo

```
IF cond THEN I1
           I2
ELSE I3
      I4
      I5
```

"especula" con anticipar la ejecución, independientemente de cual de las dos ramas vaya a ser desechada en función de que no cumpla con la condición. *Control Speculation* es el término aplicado para este caso y una de las formas de llevar a cabo esto es el uso de predicados, como se explicará en la sección siguiente.

Otra forma de "especular" es a nivel datos; la técnica consiste en leer anticipadamente un dato, suponiendo que una escritura (en general, efectuada por otro hilo de ejecución) pueda modificarla. *Data Speculation* es el término utilizado en este caso. Veamos un ejemplo concreto.

Los compiladores capaces de utilizar la técnica de especulación deben transformar una lógica que ha sido programada en forma "secuencial" en una cierta cantidad de "hilos de ejecución" en paralelo. El compilador debe determinar qué partes del código del programa se benefician con este procedimiento. Las estructuras repetitivas son un claro ejemplo: suponga un ciclo FOR-NEXT representado por el siguiente pseudocódigo

```
for i = 1 to 6
  a(i)=b(i) + 2
next i
```

Si su ejecución se divide en 3 hilos, uno podría ejecutarse en una ALU1, otro en la ALU2 y el último en la ALU3, aprovechando las tres unidades de enteros con que cuenta cierta CPU

ALU1	ALU2	ALU3
$a(1) = b(1) + 2$	$a(3) = b(3) + 2$	$a(5) = b(5) + 2$
$a(2) = b(2) + 2$	$a(4) = b(4) + 2$	$a(6) = b(6) + 2$

Se observa la ventaja de la multiplicidad del hardware aprovechado por el compilador. Un compilador o ensamblador de este tipo especifica directamente el nivel de paralelismo, indicando en "la instrucción" explícitamente cómo y cuándo ejecutar cada etapa de un hilo, dependiendo de la cantidad de unidades de ejecución que disponga la CPU. El término "la instrucción" está entre comillas, ya que cada una de ellas es en realidad un *bundle* constituido por varias instrucciones simples y una máscara o *template*, como se detallará en la sección "Formato de instrucción y *bundle* de 128 bits".

En la estructura descripta arriba no hay dependencia de datos, debido a que la ejecución de un paso no depende del valor calculado en otro anterior.

Una primera aproximación al *bundle* se grafica así:

	INSTRUCCION 1	INSTRUCCIÓN 2	INSTRUCCIÓN 3	TEMPLATE
Bundle 1	$a(1) = b(1) + 2$	$a(3) = b(3) + 2$	$a(5) = b(5) + 2$	"en paralelo"
Bundle 2	$a(2) = b(2) + 2$	$a(4) = b(4) + 2$	$a(6) = b(6) + 2$	"en paralelo"

El siguiente ejemplo muestra una dependencia de datos en otra estructura FOR-NEXT

```
for i = 1 to 6
    if a(i) = 5      then a(i)=a(i-3) - 2
                           else a(i)=b(i) +2
next i
```

ALU1	ALU2	ALU3
$a(1) = b(1) + 2$	$a(3) = b(3) + 2$	$a(5) = a(2) - 2$
$a(2) = b(2) + 2$	$a(4) = b(4) + 2$	$a(6) = b(6) + 2$

Como se observa, el cálculo de $a(5)$ depende del valor de $a(2)$, que se calculará cuando se ejecute el *bundle* 2. El valor obtenido es incorrecto y es un problema por solucionar. Lo "especulativo" consiste en asumir que esta situación no iba a producirse.

Si asignamos valores

	1	2	3	4	5	6
a	3	7	2	1	2	6
b	5	2	4	3	5	3

Los resultados serían

ALU1	ALU2	ALU3
$a(1) = 7$	$a(3) = 6$	$a(5) = 7 - 2 = 5$
$a(2) = 4$	$a(4) = 5$	$a(6) = 5$

En $i = 5$ se produce una lectura anticipada de $a(2) = 2$ en el *bundle 1* cuando aún no se ha escrito la actualización de su valor en el *bundle 2* $a(2) = 4$. Esta situación se conoce técnicamente como RAW (Read After Write). En $i = 5$ se toma el valor anticipado de $a(2)$, siendo que debería usarse por la lógica secuencial $a(2) = 4$. Esto se soluciona con un "chequeo" luego de la ejecución de un hilo. En este caso, el hilo 3 utiliza una variable compartida con el hilo 1 por lo que la decisión para tomar es desechar toda la ejecución del tercer hilo, quedando el esquema de ejecución cómo sigue:

ALU1	ALU2	ALU3
$a(1) = 7$	$a(3) = 6$	$\cancel{a(5) = 7 - 2 = 5}$
$a(2) = 4$	$a(4) = 5$	$\cancel{a(6) = 5}$
		$\cancel{a(5) = 4 - 2 = 2}$
		$a(6) = 5$

Se dice que una relación de dependencia de datos se produce cuando el valor que va a ser calculado –en este caso de $a(2)$ en el *bundle 2*– fue consumido con un valor anterior por un hilo sucesor.

14.4.1.9 Predicación

Se denomina predicación, o *predication*, a la ejecución de secuencia de instrucciones, que "dependen" de que una condición se cumpla.

Por ejemplo, en una arquitectura x86 el flujo de ejecución va por una de las ramas según la condición

```
IF (a=b) THEN a = a + 2
      I2
      I3
      I4
ELSE   b = b + 2
      I6
      I7
      I8
```

Si la condición se cumple, se ejecutan $a = a + 2$ y la secuencia de instrucciones que le sigue.

Si la condición no se cumple, se ejecuta $b = b + 2$ y la secuencia de instrucciones que le sigue.

Por lo tanto, la ejecución depende de comparar a con b y analizar el resultado del *flag Z*, en cuyo caso se sigue por una u otra rama.

Cuando se aplica la técnica de predicación, todas las instrucciones se ejecutan aún cuando dependen del valor de su predicado. Si se pueden ejecutar ambas ramas en forma simultánea, el salto puede evitarse utilizando instrucciones denominadas "código de predicado". Por ejemplo:

```
PREDV, PREDF = compare a <> b;
```



Un compilador para EPIC genera secuencias de instrucciones independientes, denominadas hilos o *threads*, para que se ejecuten en paralelo, aprovechando unidades de ejecución múltiples.



El final de un hilo se "marca" con una instrucción de parada implícita.

es un pseudo-código de predicado, donde la variable PREDV (predicado verdadero) será igual a 1 si, luego de comparar la bandera Z , ésta es 1, lo que indica igualdad; en caso contrario, PREDV es igual a cero. Por complemento, si la bandera $Z = 0$, lo que indica desigualdad, quien asume el valor 1 es la variable PREDF (predicado falso).

Para $a = 3$ y $b = 7$, entonces, la ejecución de la comparación da los valores PREDV = 0 y PREDF = 1. Por lo que el hilo de ejecución de $b = b + 2$ y sus sucesoras calculan valores, mientras que el hilo de ejecución $a = a + 2$ y sus sucesoras se comportan como instrucciones NOP.

Para $a = 5$ y $b = 5$, entonces, la ejecución de la comparación da los valores PREDV = 1 y PREDF = 0. En este caso, las ramas de ejecución se comportan de manera inversa.

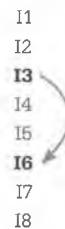
Las instrucciones en pseudocódigo se pueden representar así:

```
Rama 1 IF PREDV THEN a = a + 2; I2; I3; I4
Rama 2 IF PREDF THEN b = b + 2; I6; I7; I8
```

Lo que se logra es que se independice el salto provocado por la condición de la ejecución de las instrucciones.

El siguiente ejemplo ilustra la penalización de un salto cuando se cumple la condición para un pipeline de cinco etapas: CI (calcular la dirección de la instrucción), OI (obtener la instrucción), DE (decodificar), CO (calcular la dirección del operando), OO (obtener el operando), EJ (ejecutar).

El programa sin instrucciones predicadas es el siguiente:



donde I_3 es un salto condicional que, si se da la condición, bifurca a I_6 .

En la siguiente secuencia consideraremos que no se cumple la condición y, por lo tanto, la ejecución de las cinco instrucciones no predicadas se logra en diez etapas.

	1	2	3	4	5	6	7	8	9	10
I_1	CI	OI	DE	CO	OO	EJ				
I_2		CI	OI	DE	CO	OO	EJ			
I_3			CI	OI	DE	CO	OO	EJ		
I_4				CI	OI	DE	CO	OO	EJ	
I_5					CI	OI	DE	CO	OO	EJ

En la siguiente secuencia la condición evaluada en la etapa 8 (EJ de la I_3) hace que recién en la etapa 9 comience la ejecución de la sexta instrucción y que lo ejecutado para las instrucciones I_4 e I_5 no sirva. La ejecución total se cumple en diecisésis etapas.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
I1	CI	OI	DE	CO	OO	EJ										
I2		CI	OI	DE	CO	OO	EJ									
I3			CI	OI	DE	CO	OO	EJ								
I4				CI	OI	DE	CO	OO	EJ							
I5					CI	OI	DE	CO	OO	EJ						
I6							CI	OI	DE	CO	OO	EJ				
I7								CI	OI	DE	CO	OO	EJ			
I8									CI	OI	DE	CO	OO	EJ		

En una secuencia predicada, las instrucciones de ambas ramas del salto se ejecutan en paralelo. Supongamos que luego de la decodificación de la instrucción *I3* ya pueden entrar a ejecutarse.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
I1	CI	OI	DE	CO	OO	EJ										
I2		CI	OI	DE	CO	OO	EJ									
I3			CI	OI	DE	CO	OO	EJ								
I4					CI	OI	DE	CO	OO	EJ						
I5						CI	OI	DE	CO	OO	EJ					
I6						CI	OI	DE	CO	OO	EJ					
I7							CI	OI	DE	CO	OO	EJ				
I8								CI	OI	DE	CO	OO	EJ			

En el ejemplo presentado, la condición se evalúa generando PREDV y PREDF en la etapa 8 y antes de la etapa EJ de cualquiera de las instrucciones. En las ramas la ejecución total se cumple en trece etapas.

14.4.1.10 Predicción de saltos

Hay dos clases de predicciones: la primera se denomina estática, ya que la predicción se produce a nivel compilación; el compilador estima una dirección para cada una de las instrucciones de salto. En los saltos incondicionales siempre se conoce la dirección, por lo tanto, no es estimativa; en los saltos iterativos, lo más probable es que éstos se produzcan al inicio de la iteración y en los condicionales es posible suponer la probabilidad de que se produzcan y, en función de ello, estimar la dirección de salto con mayor probabilidad.

La segunda clase de predicción se denomina dinámica y ocurre durante la ejecución, en la que se construye una tabla de saltos y se guarda en un registro su comportamiento con el fin de decidir cuál es la dirección más probable de la próxima instrucción (Itanium cuenta con esta facilidad).

14.4.1.11 Formato de una instrucción

Las instrucciones basadas en EPIC identifican el set de instrucciones como ISA-64 bits (*Instruction Set Architecture*).

Código de operación	Registro de predicado	registro origen 1	registro origen 2	registro destino	Extensión del código de operación/alcance del salto/nada
---------------------	-----------------------	-------------------	-------------------	------------------	--

Por efecto de la compilación se juntan tres instrucciones en un sólo grupo, que se denomina *bundle*; asimismo, por cada uno se asocia una plantilla, que indica todas las posibilidades de ejecución de las tres instrucciones, esto es:

- Todas en serie.
- La primera y la segunda en paralelo, y la última serie.
- La primera y la última en paralelo, y la segunda serie.
- La segunda y la última en paralelo, y la primera serie.
- Las tres en paralelo.
- Las tres en paralelo con las del siguiente *bundle*.

14.4.1.12 Bundle de 128 bits

Instrucción 1	Instrucción 2	Instrucción 3	Plantilla o template
127			0

El objetivo de describir este procesador fue detallar el entorno de aplicaciones que puede ejecutar y, sobre todo, indicar las técnicas utilizadas para obtener alto rendimiento en la ejecución del software a nivel paralelismo de instrucciones.

14.4.2 Descripción de la arquitectura AMD64

Se puede afirmar que es una arquitectura con filosofía de 32 bits expandida a 64, de forma que permite la compatibilidad "hacia atrás" para sostener el software implementado para la industria x86 (que es mirada como un estándar en su modo de trabajo "herencia o legado").

En ella los registros se expanden a 64 bits, de modo que el software recompilado se ejecute con mayor eficiencia que en las plataformas de x86. Sin embargo, la ejecución admite aplicaciones y sistemas operativos de 32 y de 16 bits sin necesidad de recompilarlos.

Su filosofía de implementación es constituir una plataforma que abarque la ejecución del software existente y las aplicaciones de alto rendimiento de 64 bits. Los servidores, los sistemas que operan con grandes bases de datos y las aplicaciones de diseño gráfico requieren el acceso a un gran espacio de almacenamiento en memoria principal y virtual, por esta razón aporta un gran beneficio ampliar el espacio potencial de direccionamiento a 2^{64} bytes.

Además de ampliar el tamaño de los registros, se agregan unos cuantos más. El aumento del número de registros permite que los compiladores tengan una mayor flexibilidad para alojar más variables de memoria en los registros internos; esto beneficia las aplicaciones por la menor cantidad de accesos a memoria. Admite varios modos de operación, los que emulan las arquitecturas anteriores (modo herencia o *legacy*) y el modo 64 bits (largo o *long*).

14.4.2.1 Modos de operación

Modo largo (*long mode*)

Es la expansión a 64 bits del modo protegido legado de las x86. Está constituido por dos submodos, el modo 64-bit y el modo compatible.

Modo 64 bits Soporta el direccionamiento virtual con direcciones de 64 bits y las características de los registros extendidos. El sistema operativo debe ser de 64 bits y habilita este modo en un segmento de código; las aplicaciones existentes de plataformas antecesoras pueden ejecutarse sin necesidad de recompilarlas. Las características de direccionamiento incluyen la utilización del puntero de instrucción de 64 bits y el direccionamiento relativo a este puntero. Utiliza el modelo de memoria denominado plano o *flat*.



Utilizando instrucciones prefijadas con el literal REX, los registros se acceden en su modalidad extendida, esto es, todos los registros generales en 64 bits. Se puede direccionar el primer byte o todos ellos; esto implica que los compiladores pueden ajustar el registro al tamaño de los datos: Byte, Word, Dobleword y Quadword. Se agregan todos los registros XMM, que describiremos a continuación.

Aun cuando el espacio de direccionamiento virtual es de 64 bits, se ejecutan las aplicaciones compiladas para un espacio virtual de tamaño menor.

Los operandos pueden tener tamaño de 32 o 64 en función de que las instrucciones utilizadas sean "no REX" o "REX", respectivamente.

Las instrucciones de transferencia de control pueden utilizar el "modo relativo a programa", que permite relacionar el campo DATA de la instrucción con el valor actual del puntero de instrucción. Por ejemplo, "saltar quince bytes por debajo del actual IP".



Las aplicaciones en modo virtual-8086 se ejecutan en una "máquina virtual", que no interactuará con el resto de las máquinas virtuales. Este mecanismo de independencia entre las distintas tareas se ofrece desde la implementación del 386. El software que permite implementar una máquina virtual se denomina "monitor de V86".

Modo compatible. El segundo submodo del modo long permite que los sistemas operativos de 64 bits puedan ejecutar las aplicaciones existentes en modo 16 y 32, generadas por plataformas x86, sin recomilar y con la limitación de acceder sólo al espacio de direccionamiento de 4 gigabytes. La habilitación del modo está a cargo del sistema operativo, que crea un segmento de código individual igual que en el modo anterior; sin embargo, la técnica de segmentación utilizada en los x86 se aplica sin cambios. Desde el punto de vista de la ejecución del programa, genera el mismo entorno protegido de los sistemas anteriores. Desde el punto de vista de gestión del sistema operativo, se utilizan las técnicas del modo 64-bit para el manejo de interrupciones, la traducción de direcciones y las estructuras de datos.

Modo herencia. Este modo no sólo admite la compatibilidad de las aplicaciones de 16 y 32 bits, sino que también soporta sus sistemas operativos, lo que implica soportar los tres submodos antecesores: *modo real*, *modo virtual-8086* y *modo protegido*. Los microprocesadores de arquitectura AMD64 "bootean" en el modo real del modo herencia como lo hacen los microprocesadores de arquitecturas x86.

Recordemos las características de los tres modos en forma resumida:

- En modo real una aplicación utiliza segmentación de memoria, sin paginación, las aplicaciones de 16 bits utilizan un espacio de direccionamiento de 1 mega. Es un modo de operación que se ofrece desde el 286, donde las aplicaciones tienen acceso irrestricto al software de la BIOS. No se utiliza el concepto de protección de memoria y gestión multitarea. Los sistemas operativos DOS operan en modo real.
- En el modo virtual-8086 se utiliza memoria segmentada con la opción de memoria paginada. Ejecutar una aplicación en este modo permite ejecutar un programa creado para modo real con la supervisión de un programa monitor que controla todos sus pasos. Lo más importante a destacar en este modo es el manejo de las interrupciones y las excepciones, que se tratan como si se estuviese ejecutando en modo protegido; o sea, el procesador cambia de modo virtual-8086 a modo protegido antes de llamar al servicio de interrupción; cuando se ejecuta la instrucción IRET (retorno de interrupción), el procesador vuelve a cambiar de modo. Recuérdese que cuando se produce una interrupción o excepción, el procesador utiliza el número de interrupción para entrar en la tabla de vectores de interrupción (IDT, *Interrupt Descriptor Table*), donde encontrará la dirección del segmento donde se encuentra el servicio. El procesador almacena a continuación todos los registros de CPU en la pila y recarga los registros CS y EIP con la dirección del servicio (los demás registros de segmento serán puestos a 0).
- El modo protegido se ofrece desde el 286, con características orientadas a la "gestión segura" de entorno multitarea, como la estabilidad del sistema, la protección de memoria y el soporte hardware para la gestión de memoria virtual. A partir del 386 y los

procesadores posteriores de 32, se agregó la técnica de paginación como parte del modo protegido. Los sistemas operativos que se ejecutan en este modo son LINUX, OpenBSD y Windows a partir de la versión 3.0, sin incluir Windows Vista.

14.4.2.2 Tabla de registros de propósito general (GPRs)

Tabla 14-11. Registros GPRs

63	0	
		RAX
		RBX
		RCX
		RDX
		RBP
		RSI
		RDI
		RSP
		R8
		R9
		R10
		R11
		R12
		R13
		R14
		R15

Los registros generales utilizados y descriptos hasta ahora eran EAX, EBX, ECX, EDX, EBP, ESI, EDI y ESP; un total de 8 de 32 bits que se utilizan en modo *herencia*, mientras que en modo *largo* son 16 registros identificados como RAX, RBX, RCX, RDX, RBP, RSI, RDI, RSP, R8–R15, todos de 64 bits de ancho. La tabla muestra la división en los ocho primeros para que los programas puedan acceder al “medio registro de orden inferior”.

Las instrucciones asociadas con estos registros son las de transferencia, que permiten la carga de datos *en* registros internos *desde* la memoria principal y que permiten la transferencia de datos alojados *en* los registros hacia la memoria principal, además de instrucciones de tipo aritmético o lógico que operan sobre datos alojados en ellos.

14.4.2.3 Tabla de registros MMXn (64 bits)

63	0	
		MMX0
		MMX1
		MMX2
		MMX2
		MMX4
		MMX5
		MMX6
		MMX7

En modo *legacy* o compatible se utilizan los 8 registros de 64 bits para instrucciones MMXn y los 8 primeros registros de 128 bits denominados XMMn de la tabla de registros XXMn.

Las instrucciones a las que están asociados son las denominadas “de tecnología MMX™”, y las de AMD, conocidas como “de tecnología 3DNow!™”, son instrucciones que cargan y almacenan datos y que los operan en forma escalar y vectorial, o sea que son de tipo SIMD a nivel microarquitectura.

Las instrucciones vectoriales pueden ejecutar una operación que afecte, de manera simultánea, a múltiples datos en forma independiente en cada uno de ellos.

14.4.2.4 Tabla de registros XMMn (128 bits)

Tabla 14-13. Registros XMMn (128 bits)	
XMM0	
XMM1	
XMM2	
XMM3	
XMM4	
XMM5	
XMM6	
XMM7	
XMM8	
XMM9	
XMM10	
XMM11	
XMM12	
XMM13	
XMM14	
XMM15	
127	0

En modo 64 los registros aumentan al doble en capacidad y número, y se denominan XMM0 a XMM15.

Las instrucciones asociadas a ellos se denominan instrucciones multimedia de 128 bits y son de tipo SIMD (*Single Instruction Multiple Data*), en sus modalidades SSE, SSE2, SSE3, SSE4A (vistas en el apartado SIMD). Permiten la operación de carga y almacenamiento, como las enunciadas para los registros de propósito general y las operaciones escalares y vectoriales para datos de coma fija de 128 bits y datos de coma flotante. Estas instrucciones se utilizan en las complejas aplicaciones de cálculo científico y multimedia.

14.4.2.5 Tabla de registros "X87"

Tabla 14-14. Registros "X87"	
FPR0	
FPR1	
FPR2	
FPR3	
FPR4	
FPR5	
FPR6	
FPR7	
79	0

Los ya conocidos registros "x87" identificados como de FPR0 a FPR7 son de 80 bits. Las instrucciones asociadas se denominan instrucciones de punto flotante X87 y se utilizan en modo herencia.

14.4.2.6 Modelo de memoria

En este apartado se describe cómo se "visualiza la memoria" desde el punto de vista de las aplicaciones en los distintos modos enunciados. Los registros de segmentos descriptos a continuación son el soporte para el direccionamiento a memoria.

Registros de segmento

Una dirección está constituida por dos campos: el selector y el de desplazamiento. Vimos que en modo real o modo 16 bits la primera entidad es de 16 y la segunda también, y una dirección posible es el siguiente ejemplo: 3452:789B. En este caso el selector opera como base de segmento y la cadena hexadecimal luego de los dos puntos indica el desplazamiento dentro del segmento, con un alcance de 16 bits ($2^{16} = 64\text{ K}$).

En modo 32 el ejemplo sería 4527:45637AB0, donde 4527 es el campo selector y luego de los dos puntos la cadena hexadecimal hace referencia a un desplazamiento de 32 bits, esto es, $2^{32} = 4\text{ G}$. En esta arquitectura se visualizan en los modos herencia y compatible todos los registros de segmento enunciados para las arquitecturas antecesoras y que describimos a continuación.

Tabla 14-15. Registros de segmento.

CS	
SS	
DS	
ES	
FS	
GS	

15

0

Tabla 14-16. Registros de segmento en modo 64-bits.

Sólo atributos	
No se usa	
No se usa	
No se usa	
FS sólo base	
GS sólo base	

15

0

Como se observa al comparar las tablas, el modo 64 bits utiliza una dirección virtual de 64; el bit de menor orden indica si el valor de los 63 bits de orden superior es de todos ceros o todos unos, que es una dirección canónica, necesaria para la gestión de memoria virtual. Sólo se utilizan los registros CS, FS y GS, los dos últimos con la utilidad de ser registros base para el cálculo de direcciones.

El modo-64 bit utiliza un modelo segmentado plano, que en realidad significa que la memoria virtual se visualiza como un único bloque, el sistema operativo determina el selector para un segmento de código, de datos o de pila por cuestiones de protección, pero la dirección virtual de todos los segmentos es canónica.



Fig. 14.1. Traducción de la dirección virtual.

Dado que la mayoría de los sistemas operativos actuales gestiona la técnica de segmentación exclusivamente vía software, sin el requerimiento del apoyo del hardware de direccionamiento (unidad de segmentación), propia de las arquitecturas *x86*, en este modo una dirección virtual se "mapea" a física por medio de la unidad de paginación.

En modo compatible se sigue el modelo presentado como modelo de segmentos paginados, del capítulo 9.

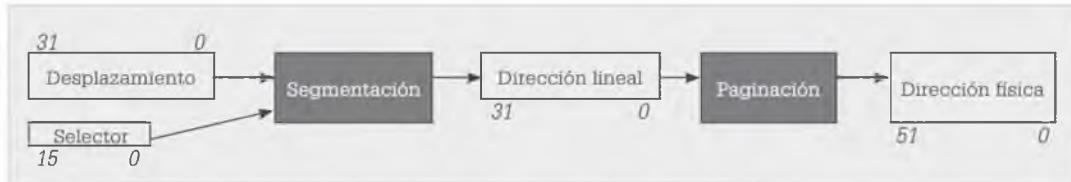


Fig. 14.2. Traducción de dirección virtual a física.

En modo herencia

La gestión de memoria varía en función de los tres modos heredados que se describieron.

En modo real soporta aplicaciones de 16 bits que utilizan direcciones segmentadas con 16 bits para la base del segmento a la que se la multiplica por diez en base 16 y se le suman los 16 bits de desplazamiento para lograr la dirección física o lineal de 20 bits.

En modo virtual-8086 se ejecutan aplicaciones de 16 bits en modo protegido, la dirección física se obtiene igual pero puede "mapearse" en un espacio de direccionamiento de 2^{32} bytes utilizando la unidad de paginación.

En modo protegido se soportan aplicaciones de 16 y 32 bits. La dirección virtual está constituida por el campo selector de 16 bits que permite acceder a la tabla de descriptores de segmento para obtener la base del segmento de 32 bits y sumarle a este valor los 32 bits del desplazamiento. En casos eventuales la dirección lineal de 32 bits puede "mapearse" en la unidad de paginación

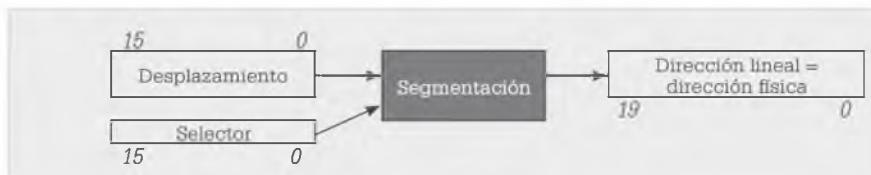


Fig. 14.3. Modo real.

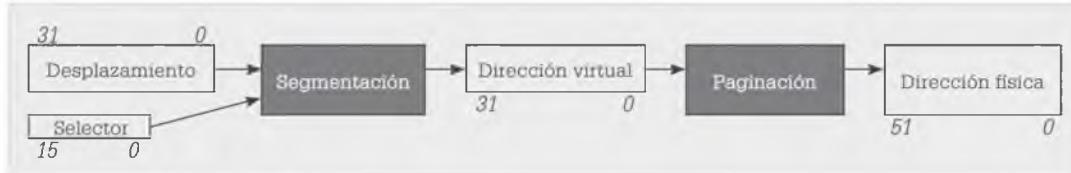


Fig. 14.4. Modo virtual-8086 y modo protegido.

14.5 Resumen

En este capítulo se ha presentado el paralelismo como una forma de optimización de rendimiento de los procesadores que han producido un impacto en el mercado de las computadoras de venta masiva y que operan con microprocesadores de 64 bits, el Itanium de Intel y el AMD64. El Itanium de Intel pertenece a la tecnología denominada arquitectura EPIC, que se diferencia de la clásica arquitectura CISC en que se puede implementar la ejecución en paralelo a nivel instrucción preparando el código desde el proceso de compilación, los más comercializados, Itanium e Itanium II, fueron desarrollados para el ámbito de los servidores.

Aún cuando se ha desarrollado un Windows XP Profesional de 64 bits para ellos, su precio y su entorno de soporte hardware en la placa de sistema (que es totalmente diferente), no ha sido implementado en plataformas de computadoras personales. Para éste último nicho del mercado se ha vendido ampliamente el AMD64, que es básicamente un procesador de 32 bits ampliado, fundamentalmente, en lo que se refiere al la cantidad de registros y a la cantidad de líneas asociadas al bus de direcciones físicas que sugiere un potencial de espacio de direcciones de 2⁶⁴ y en la diversidad de "modos de operación del procesador".

14.6 Contenido de la página Web de apoyo



El material marcado con asterisco (*) sólo está disponible para docentes.

Resumen gráfico del capítulo

Animación

Demostración de las ventajas del *Pipelining*.

Autoevaluación

Video explicativo (01:12 minutos aprox.)

Audio explicativo (01:12 minutos aprox.)

Evaluaciones Propuestas*

Presentaciones*

Bibliografía

- Brey, Barry B. - *The Intel Microprocessors 8086/8088, 80186/80188, 80286, 80386, 80486, Pentium, and Pentium Pro Processor Architecture, Programming, and Interfacing* 6ta ed. - Prentice Hall, New Jersey 2002.
- Floyd, Thomas y Cox, Jerry - *Digital Fundamentals Solutions Manual Digital Experiments Troubleshooting* 6ta ed. - Pearson, Old Tappan 1996.
- Gajski Daniel D., Dutt Nikil D., Wu, Allen C-H, Lin, Steve Y-L - *High-level Synthesis: Introduction to Chip and System Design* - Springer, New York 1992.
- Gajski Daniel D. - *Principles of digital design* - Prentice-Hall, New Jersey 1996.
- Handy, Jim - *The Cache Memory Book* - Academic Press, San Diego 1993.
- Hamacher, Carl, Vranesic, Zvonko y Zaky, Safwat - *Organización de Computadores* 5ta ed. - McGraw-Hill/Interamericana de España, Madrid 2003.
- Hennessy, John L. y Patterson, David A. - *Computer Architecture. A quantitative Approach* 2da ed. - Morgan Kaufmann, San Francisco 1996.
- Hwang, Kai y Briggs Fayé A. - *Arquitectura De Computadoras y Procesamiento Paralelo* - McGraw Hill, México D.F. 1988.
- Ifrah, Georges - *The Universal History of Computing. From the Abacus to the Quantum Computer. Volume II* - John Wiley & Sons, New York 2000.
- Ifrah, Georges - *The Universal History of Numbers From Prehistory to the Invention of the Computer. Volume I* - John Wiley & Sons, New York 1999.
- Khurshudov, Andrei - *The Essential Guide to Computer Data Storage* - Prentice Hall, New Jersey 2001.
- Knuth, Donald E. - *The Art of Computer Programming Vol 2* - Addison-Wesley Professional, Massachusetts 1998.
- Lipovski, G. Jack - *16 and 32 bits microcomputers interfacing* - Prentice-Hall, New Jersey 1990.
- Lloris Ruiz, Antonio y Prieto Espinosa, Alberto - *Diseño Lógico* - McGraw-Hill/Interamericana de España, Madrid 1996.
- Maljugin, Vitaly, Izrailevich, Jacov, Lavin, Semyon y Sopin, Aleksandr - *The Revolutionary Guide To Assembly Language* - Wrox Press, Chicago 1993.
- Mandado, Enrique - *Sistemas Electrónicos Digitales* 7ma ed. - Marcombo, Barcelona 1992.
- Mano, Morris - *Lógica Digital y Diseño de Computadores* - Prentice-Hall, New Jersey 1989.
- Mano, Morris - *Computer System Architecture* 3ra ed. International edition - Prentice Hall, New Jersey 1992.
- Morgan, Christopher y Waite, Mitchell - *Introducción al Microprocesador 8086/8088* - McGraw Hill, Madrid 1993.
- Parhami, Behrooz - *Computer architecture: from microprocessors to supercomputers* - Oxford University Press, New York 2005.
- Shen, John P. y Lipasti, Mikko H. - *Arquitectura de Computadores: Fundamentos de los Procesadores Escalares* - McGraw-Hill, Madrid 2006.

Taft, S. Tucker and Duff, Robert A. - *Ada 95 Reference Manual: Language and Standard Libraries* - Springer, New York 1997.

Wakerly, John F. - *Digital design: principles and practices 3ra ed.* - Prentice Hall, New Jersey 1999.

Manuales electrónicos

AMD64 Architecture. *Programmer's Manual. Volume 1: Application Programming Publication No. Revision Date.* 24592 3.14, September 2007.

AMD64 Architecture. *Programmer's Manual. Volume 2: System Programming. Publication No. Revision Date.* 24593 3.14, September 2007.

AMD64 Architecture. *Programmer's Manual. Volume 3: General-Purpose and System Instructions. Publication No. Revision Date.* 24594 3.14, September 2007.

IA-32 Intel Architecture Software Developer's Manual. *Volume 1: Basic Architecture.*

IA-32 Intel Architecture Software Developer's Manual. *Volume 2: Instruction Set Reference Manual.*

Intel® Itanium® Processor Reference Manual for Software Development. *Volume 1: Application Architecture.* Revision 2.2, 2006.

Publicaciones

Bernal J, Witzgall C. - *Integer Representation of Decimal Numbers for Exact Computations.* Journal of Research of the National Institute of Standards and Technology. Vol. 111 #2, pp79–88. National Institute of Standards and Technology, marzo - abril 2006.

Duncan R. - *Improving the Scalability of Shared Memory Systems through Relaxed Consistency. A Survey of Parallel Computer Architectures.* IEEE Computer, febrero 1990.

Flynn M. - *Some Computer Organizations and Their Effectiveness.* IEEE Trans. Comput., Vol. C-21, 1972.

Furukawa, Koichi - *Philosophy of New Generation Computing. New Generation Comput.* Volume 25, Number 1, Ohmsha, página 144, noviembre 2006.

Goldberg, David - *What Every Computer Scientist Should Know About Floating-Point Arithmetic.* - ACM Computing Surveys Volume 23, Issue 1, página 5, marzo 1991.

International Business Machine, Inc. - 64Mb Synchronous DRAM, IBM Data Sheet 364164, enero 2001.

Institute of Electrical and Electronics Engineers, IEEE. - Standard for Binary floating-Point Arithmetic, ANSI/IEEE Standard 754-1985, agosto 1985.

Sterbenz PH. - Floating-point Computation. SIAM Review. Society for Industrial and Applied Mathematics. Vol. 18, No. 1, pp. 138-139 (review consists of 2 pages), enero 1976.

Páginas Web

How to Optimize Instruction Latencies in Assembly Code for 64-Bit Intel® Architecture.

<http://software.intel.com/en-us/articles/how-to-optimize-instruction-latencies-in-assembly-code-for-64-bit-intel-architecture/>

[Marzo de 2010]

INFO: IEEE Floating-Point Representation and MS Languages.

<http://support.microsoft.com/kb/36068/en-us?fr=1>

[Marzo de 2010]

Índice analítico

- Ábaco 5
Acarreo 77
Acceso
 asociativo 210, 215
 random 208, 212, 223
 secuencial 209
Acumulador 5, 75, 76, 157
Álgebra de Boole 5, 88, 89, 90, 91, 92, 93, 94, 100
Algoritmo 2, 5, 13, 15, 19, 82, 278
Aiken, Howard 7
Alimentación 20, 100, 193, 199
Almacenamiento 6, 9, 13, 14, 19, 20, 21, 22, 208, 215, 217, 229, 233, 234, 290, 294
Alta impedancia 101, 129, 313
ALU 74, 75, 77, 78, 79, 150, 151, 157, 158, 170, 171, 172, 173, 174, 175
Analógica 5, 9, 307
Ancho de banda 231, 232
Aplicación 4, 7, 8, 12, 15, 23
Archivo 14, 244, 245, 246, 247, 250, 252, 253
Aritmética
 binaria 74
 decimal 74, 79
Arquitectura
 de buses 303
 de computadoras 3, 4, 12, 15, 23, 38, 40, 41, 62, 63, 64, 65, 67
 del set de instrucciones 4
 Von Neumann 150
ASCII 50, 52, 53, 54, 55, 56, 57, 69
Assembler 146, 147, 148, 172, 282, 284
ATA 292, 309
ATA-2 309
Babbage, Charles 5
Banderas 60, 76, 77, 157, 162, 167, 171, 172
Base de datos 12, 245, 290
BCD 56, 57, 58, 79, 80, 82, 83, 121
Biestable 131, 133, 134, 135, 136, 137, 138, 139, 214
Big-endian 234
Binario 26, 27, 28, 29, 30, 31, 32, 33, 35, 36, 37, 43, 44
BIOS 192, 193, 194, 195
Bit de paridad 68
Boole, George 5
BPI 244
Brazo actuador 246, 248
Buffer 22, 101, 253, 293
Bundle 330, 331, 334
Bus (es) 2, 7, 17, 21, 126, 130, 181, 219, 300, 301, 302, 303, 305, 307
 de control 129
 de dato 220
 de direcciones 220
Búsqueda de pista a pista 292
Byte 16, 26
Caché de disco 248, 253
Caching 221, 222
Calculadora digital 5
Campo DATA de la instrucción 187, 191, 238, 335
Carga
 continua 10
 especulativa 202, 327
Carga por lotes 10
CAS 229, 230
CAS Latency 230
CD-R 295
CD-ROM 254, 295
Celda binaria 211, 229
Chip/s 180, 181
Chip de memoria 229
Chip Select 129
Ciclo de
 bus 300
 instrucción 152
 de máquina 166, 167
 de memoria 166
 de reloj 166, 167
Cilindro/s 290, 291, 293
Circuito 114, 115, 116, 117, 118, 119, 120, 121, 123, 124, 127, 128, 129, 131, 132, 133
 integrado 18
 digitales 88, 89
CISC 19, 184, 188, 200
Cluster 247, 249, 250, 251
CMP 322
Codificador 120, 121
Código 52
 de detección de errores 68
 de instrucción 151, 153, 157, 164, 165
 de máquina 147, 148, 157, 173, 175
 de operación 149, 151, 158, 167, 170
Código nativo 147
Cola de espera 10
Compilador 4, 50, 282, 284
Complemento
 a 2 42
 binario 82
Compuertas lógicas 93, 114
Computadora 2
 de propósito general 149, 150
 secuencial automática 7
Condensador 211, 214, 229
Conjuntos 223, 224
Controlador de bus 300
Controladora de disco 291
CPU 2, 8, 17, 50, 150, 151
CU 7
DACK 314
Dato 6, 8, 13, 14, 15, 16
DDR 232
Decimal 26, 27, 28, 29, 30, 31, 32, 33, 34, 37, 38, 40, 41, 43, 44
Decodificador 120
Densidad 13, 211, 244, 246, 247, 295, 296
Desempeño 291
Desfragmentación 250
Despachador 280
Desplazamiento 17, 74, 79, 126, 138, 139, 167, 173, 188, 240, 271
Digitales 8, 9, 16
Dígito binario 16
DIMM 230, 231, 233
Dirección
 física 212, 222, 227
 lineal 239, 240, 242
 lógica 235, 236, 237
 segmentada 187
 virtual 237, 238, 239, 240, 241
Direccionamiento
 a una pila 268
 de la CPU asociado a registros 265
 directo por registro 266
 directo de memoria 263
 implícito 263
 indexado 266
 indirecto 264
 inmediato 264
 relativo a la base 267
Directorio 250, 251, 252
Disco magnético 10
Diseño de algoritmos 4
Dispositivo electrónico 2
Dispositivos de entrada/salida 2, 280, 290, 300, 304
DMA 303, 305, 306, 309, 310, 313, 314
Dobleword 335
Dominios 291
DRAM 211, 216, 217, 220, 222, 229, 230, 231, 232, 233
Driver 281
DRQ 314
DSM-NUMA 324
DSM-SVM 324
DVD 290, 294, 295
DVD-RAM 295
DVD-ROM 295
E-IDE 309
EISA 304
Ejecución especulativa 322
ENIAC 7
Enlazador 284
Ensambladores 282
Entrada/salida 14, 281, 300, 303, 306, 307, 310, 311, 312, 313
EPIC 327, 331, 333, 339
EPROM 127, 129
Escalar 321
Espacio lineal de direcciones 227, 235
Estructuras de datos 22
Etiquetas 217, 218, 219, 220, 222, 223, 224, 225
Excepciones 192, 193, 238, 242, 335
FAT 250, 251

- Firmware** 15
Flag/s 76, 77
Flynn, Michael J 322
Formas canónicas de una función 104, 105
Formato
 de datos 22, 150
 de instrucción 149
Frecuencia 166
Función booleana 92, 102
Gates 93
GHz 20, 188, 228
Grado de paralelismo 318, 319
Handshaking 305
Hardware 3, 4, 12, 13, 15, 22, 23
 Hexadecimal 26, 28, 29, 31, 32, 33, 34, 35, 36, 39, 41, 42
Hollerith, Herman 6
Host 292, 293, 294, 297
Hz 166
IBM 6
 Información 3, 5, 6, 7, 9, 12, 13, 14, 15, 16, 17, 18, 19, 21, 22
 Instrucciones 2, 3, 4, 6, 7, 12, 13, 15, 16, 17, 19, 20
 Inteligencia artificial 12, 13
Interfaz serie 308
Intérpretes 282, 284
 Interrupción/es 188, 192, 193, 194, 195, 196
IOP 305, 309, 312
IRQ 303, 313, 314
ISA 304
 ISA-64 bits 333
JEDEC 214, 229
Latch 303
 Latencia rotacional 292, 293, 294
Lenguaje
 de alto nivel 284
 de máquina 147, 148
 de programación 2, 5
 de señales 4
Little-endian 234
 Lógica digital 18, 111, 114, 142, 228
Look-aside 219
Look-through 219
LRU 225
LSB 35
LSI 11, 19
Macro 284
 Magnitud continua 5
Mainframe 274
Malla 324
 Manejador de dispositivo 307
 Mantisa 52, 53, 62, 63, 64, 65, 66, 67, 78
 Mapas de Karnaugh 88, 109
 Máquina de medidas 5
 Máquina virtual 11, 12, 335
 Mb/seg 211
 Memoria(s) 208
 de lectura/escritura 149
 de sólo lectura 129
 principal 208
 dinámicas 229, 230
 estáticas 229
 perennes 208
 vivas 211
 volátiles 208
 virtual 237
MEMR 303, 314
MEMW 303
MHz 231, 232
 Microarquitectura 4, 321, 323, 337
MicroChannel 304
 Microcódigo 180, 183, 184, 200
 Microcomputadoras 21
 Microcontrolador 181
 Microinstrucción 184
 Microoperaciones 15, 148
 Microprocesador 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 192, 193, 194, 196
 Micropograma 323
 Microsegundo 243
 Milisegundos 293, 294
MIMD 321, 323, 324, 325
MISD 323, 324
MMU 239
MO 294, 295, 296
Modo
 compatible 334, 335, 338
 de direccionamiento 262
long 335
 protegido 334, 335, 336, 339
 real 335, 338, 339
 virtual 335, 339
MSB 35
 Multiplexor 124, 125, 126, 127
 Multiprogramación 10, 11
 Multitarea 11, 281
 Multiusuario 11, 281
 Nanosegundo 20, 318
 Napier, John 5
Nibble 16
 Niveles de caché 227
 Normalizado 302
 Octal 26, 27, 28, 29, 31, 32, 34, 35, 36, 39, 40
 Organización de computadoras 23
OTP 296, 297
 Overflow 60, 63, 65
 Página 235, 237, 238, 239, 242, 243
PAL 130
 Palabra
 de CPU 180, 185
 de memoria 182
 Paralelismo
 a nivel ejecución 191, 318
 a nivel instrucción 180, 319, 321, 326, 329
 de granulado grueso 322
 explícito 327
 Paridad 115, 116
 Partición 248, 249, 252, 253
PCI 304
PCMCIA 296, 297
 Periféricos 6, 21, 22, 23, 24
 Pila 186, 195, 196, 240, 268
Pipeline 320, 321, 332
Pipelining 318
Pista/s 246, 247
PLA 130
Plantilla 261
Plug and Play 304
Pop 197
Port 22, 23, 307, 308
 Precisión doble
 doble 68
 simple 68
Predicación 326, 329, 331
Predicado 202
 Predicción de saltos 322, 329, 333
 Procesador vectorial 12
 Procesamiento de datos 3, 4, 6, 13, 14
 Proceso 279, 280, 281, 283, 284
 Programa 2, 5, 6, 7, 8, 9, 10, 12, 13, 14, 15, 23
 Programa 7, 148, 194, 312, 340
 ejecutable 278, 280, 282, 283, 284
 fuente 278, 282, 283, 284
 Programación 2, 4, 5, 15
Prolog 13
PROM 127, 129
Puerto 300, 302, 307
 Pulso de reloj 167
Push 197
Quadword 335
 RAM 208, 209, 210, 211, 212, 213, 214, 215, 217, 218, 219, 220, 221, 222, 223, 224, 225, 227, 228, 229, 230, 233, 243, 254
 Rama de ejecución 322
 Ramificaciones condicionales 201
RAS 230
 Redes 8, 11
 Referencia a memoria 163, 164, 269
Refresco 211, 229
Refresh 211, 229
 Registro(s) 124, 126, 127, 136, 137, 138, 139, 140
 acumulador 159, 170, 174
 contador 137, 138
 de desplazamiento 139
 de estado 165, 171
 índice 186
 lógico 244, 247
 puntero de instrucción 188
 de cálculo 185, 186, 195
 de segmento 185, 187
 invisibles 184
 paralelo-paralelo 136

- punteros 185, 186, 195
rama 188
visibles 184
Reloj 20, 134, 166, 167
Resguardo de contexto de CPU 194
Residente 193
Restauración de contexto de CPU 194
Retrieve de memoria 322
RISC 184, 188, 200
RND 225
ROM 127, 128, 129, 130, 208, 209, 216, 249, 254
RPM 291, 293
SCSI 308, 309
SDRAM 232
Sector 208, 223, 225, 247, 248, 249, 253
Secuenciador 168, 170
Segmento 120, 185, 186, 187, 189, 190, 192, 195, 196, 198
de código 187, 189, 190, 192, 195
de datos 196
de pila 187, 195
Semiconductor/es 208, 209, 211, 214, 216, 217, 229, 243, 244
Set de instrucciones 2, 4, 19
Seudoinstrucción 283, 284
Silicio 10, 18, 180
Simbólico de máquina 282
SIMD 323, 324, 326, 337
SIMM 230
SISD 321, 323
Sistema de información 13
Sistema operativo 3, 4, 6, 9, 10, 11, 12, 23, 278, 279, 280, 281
Sistemas interactivos 11
SMP bus 324
SMP Escalable 324
SMT 322
Software 3, 4, 12, 13, 15, 22, 23
de administración de 278
de aplicación 4, 278, 279
de base 23, 149, 278
de comunicaciones 278
Soporte 244
SRAM 211, 214, 217, 297
SSI 10, 19
Stack 187, 194, 195, 268
Status register 63, 77, 171, 175
Strings 53, 55
Subrutina 195, 196, 217, 284
Supercomputadoras 12
Superescalar 319, 321
Tarjetas Flash 297
Task 281
T/C 315
TDA 284
Teleproceso 11
Temporizador 137, 138, 188
Teoría de circuitos 5, 146
- Thread* 322
Tiempo
compartido o *time sharing* 11
de acceso a disco 293
de acceso a los datos 293
de acceso a memoria 166
real 11, 188, 281
Tipos de datos 50
TLB 243
TLP 322
TNSE 284
Transductor 244
Transferencia
con acceso directo a memoria 312, 313, 314
controlada por programa 312
elemental 300, 304, 305, 307, 308, 312, 314
iniciada por interrupción 312, 313
Transistor 18, 19, 100
Triestado 101
Ultra ATA 309
UMA 324, 325
Underflow 63, 65
Unicode 50
Unidad (s)
central de proceso 2, 17, 50
de control 2, 3, 7, 15, 24, 50, 147, 149, 151, 165, 170
de paginación 239, 242, 338, 339
de segmentación 239, 241, 338
de cálculo 2, 60, 181
USB 300, 307
Usuario 11
Utilitarios o utilidades 278
Vector 194, 266, 326
Velocidad de transferencia del disco 294
VESA 304
Via 224
VL 304
VLSI 19
Voltios 110
Von Leibniz, Gottfried Wilhelm 5
Word 153, 335
WORM 295
Write back 226
Write through 226
XMM 335

