

**¡¡APRUEBA TU EXAMEN CON SCHAUM!!**

# Conceptos de Informática

*Schaum*

**Alberto Prieto Espinosa • Beatriz Prieto Campos**

**REDUCE TU TIEMPO DE ESTUDIO**

**INCLUYE 155 EJEMPLOS PRÁCTICOS Y 560 PREGUNTAS TEST**

**CONTIENE 487 PROBLEMAS, 219 DE ELLOS, RESUELTOS**

Utilízalo para las siguientes asignaturas:

✓ FUNDAMENTOS DE COMPUTADORES

✓ INTRODUCCIÓN A LOS COMPUTADORES

✓ INTRODUCCIÓN A LA INFORMÁTICA

✓ INTRODUCCIÓN AL USO DE LOS COMPUTADORES

✓ INTRODUCCIÓN A LA INGENIERÍA INFORMÁTICA

# **Conceptos de informática**



# Conceptos de informática

**Alberto Prieto Espinosa**  
**Beatriz Prieto Campos**

*Departamento de Arquitectura y Tecnología de Computadores*  
*Universidad de Granada*



**MADRID • BOGOTÁ • BUENOS AIRES • CARACAS • GUATEMALA • LISBOA • MÉXICO**  
**NUEVA YORK • PANAMÁ • SAN JUAN • SANTIAGO • SÃO PAULO**  
**AUCKLAND • HAMBURGO • LONDRES • MILÁN • MONTREAL • NUEVA DELHI • PARÍS**  
**SAN FRANCISCO • SIDNEY • SINGAPUR • ST. LOUIS • TOKIO • TORONTO**

## **CONCEPTOS DE INFORMÁTICA**

No está permitida la reproducción total o parcial de este libro, ni su tratamiento informático, ni la transmisión de ninguna forma o por cualquier medio, ya sea electrónico, mecánico, por fotocopia, por registro u otros métodos, sin el permiso previo y por escrito de los titulares del Copyright.

DERECHOS RESERVADOS © 2005, respecto a la primera edición en español, por  
McGRAW-HILL/INTERAMERICANA DE ESPAÑA, S. A. U.  
Edificio Valrealty, 1.ª planta  
Basauri, 17  
28023 Aravaca (Madrid)

ISBN: 84-481-9857-3

Depósito legal: M.

Editora: Concepción Fernández Madrid  
Compuesto en Puntographic, S. L.  
Impreso en

IMPRESO EN ESPAÑA - PRINTED IN SPAIN

*Dedicado a Laura Núñez Prieto*



# CONTENIDO

PRÓLOGO	.....	XV
CAPÍTULO 1	Introducción .....	1
	1.1. Conceptos básicos .....	1
	1.2. Estructura funcional de los computadores .....	3
	1.2.1. Unidades funcionales .....	3
	1.2.2. Parámetros para caracterización de prestaciones .....	6
	1.3. Programas e instrucciones .....	8
	1.4. Tipos de computadores .....	9
	1.4.1. Clasificación según la generalidad de su uso .....	9
	1.4.2. Clasificación atendiendo a la potencia de cálculo .....	10
	1.5. Niveles conceptuales de descripción de un computador .....	12
	1.6. Aplicaciones de la informática .....	13
	1.7. Conclusión .....	15
	Test .....	15
	Problemas resueltos .....	17
	Problemas propuestos .....	22
CAPÍTULO 2	Representación de la información en los computadores .....	25
	2.1. Representación de textos .....	25
	2.2. Representación de sonidos .....	28
	2.3. Representación de imágenes .....	29
	2.3.1. Mapa de bits .....	30
	2.3.2. Mapas de vectores .....	31
	2.4. Representación de datos numéricos .....	32
	2.4.1. Datos de tipo entero .....	33
	2.4.2. Datos de tipo real .....	36
	2.5. Detección de errores .....	41
	2.6. Compresión de datos .....	43



2.7. Conclusión .....	46
Test .....	46
Problemas resueltos .....	49
Problemas propuestos .....	58

## CAPÍTULO 3 El nivel de lógica digital ..... 61

3.1. Sistemas combinacionales .....	61
3.1.1. Principios del álgebra de conmutación .....	63
3.1.2. Minimización algebraica de funciones .....	65
3.1.3. Formas normalizadas .....	67
3.1.4. Implementación de funciones. Puertas lógicas .....	69
3.1.5. Bloques combinacionales básicos .....	72
3.1.5.1. Sumador binario .....	72
3.1.5.2. Decodificador .....	75
3.1.5.3. Codificador y codificador de prioridad .....	75
3.1.5.4. Multiplexor .....	76
3.1.5.5. Demultiplexor .....	76
3.1.5.6. Puerta tri-estado .....	77
3.1.5.7. Dispositivos lógicos programables: ROM, PLA y PAL .....	78
3.1.5.8. Unidades aritmético-lógicas (ALU) .....	82
3.2. Sistemas secuenciales .....	82
3.2.1. Elementos de memoria. Biestables .....	83
3.2.2. Bloques secuenciales básicos .....	85
3.2.2.1. Registros .....	85
3.2.2.2. Contadores .....	87
3.2.2.3. Bancos de registros .....	87
3.2.2.4. Memoria RAM .....	88
3.3. Conclusiones .....	90
Test .....	90
Problemas resueltos .....	93
Problemas propuestos .....	103

## CAPÍTULO 4 El procesador ..... 105

4.1. Elementos internos de un procesador .....	105
4.2. Esquema de funcionamiento de un procesador .....	107
4.2.1. Gestión de las instrucciones de transferencias de datos y aritmético-lógicas .....	108
4.2.2. Gestión de las instrucciones de control .....	111
4.3. Programación en código máquina (CODE-2) .....	114
4.3.1. Elementos a los que se tiene acceso desde el lenguaje máquina .	115
4.3.2. Formatos de instrucciones y datos .....	115
4.3.3. Repertorio de instrucciones máquina .....	116

4.3.4. Algunos trucos de programación .....	124
4.3.5. Ejemplo de programa .....	127
4.4. Lenguaje ensamblador .....	129
4.5. Microprocesadores y microcontroladores .....	129
4.6. Procesadores RISC y CISC .....	130
4.7. Conclusión .....	131
Test .....	131
Problemas resueltos .....	135
Problemas propuestos .....	170

## CAPÍTULO 5 La memoria ..... 175

5.1. Memoria interna .....	175
5.1.1. Memoria principal .....	175
5.1.2. Memoria caché .....	180
5.2. Jerarquía de memoria .....	181
5.3. Memoria externa .....	183
5.3.1. Escritura y lectura de información en forma magnética. ....	183
5.3.2. Discos magnéticos. ....	184
5.3.2.1. Principios de funcionamiento. ....	185
5.3.2.2. Unidades RAID .....	189
5.3.3. Cintas magnéticas. ....	190
5.3.4. Discos ópticos. ....	193
5.3.4.1. Discos compactos (CD) .....	194
5.3.4.2. Disco digital versátil (DVD) .....	196
5.4. Conclusión .....	197
Test .....	197
Problemas resueltos .....	200
Problemas propuestos .....	210

## CAPÍTULO 6 Periféricos de E/S ..... 213

6.1. Definición y tipos .....	213
6.2. Teclados .....	215
6.3. Entradas manuales directas .....	215
6.4. Detectores ópticos .....	218
6.5. Detectores de datos magnetizados .....	221
6.6. Monitores de visualización .....	221
6.6.1. Pantallas de tubo de rayos catódicos (CRT) .....	223
6.6.2. Pantallas planas .....	224
6.6.3. Controlador de vídeo .....	225
6.7. Impresoras .....	227
6.8. Periféricos para aplicaciones multimedia .....	230
6.9. Entradas/salidas de señales analógicas .....	231
6.10. Conclusiones .....	234

	Test .....	235
	Problemas resueltos .....	237
	Problemas propuestos .....	243
<b>CAPÍTULO 7</b>	<b>Estructuras de computadores .....</b>	<b>245</b>
	7.1. Estructuras básicas de interconexión .....	245
	7.2. Buses .....	248
	7.3. Ejemplo de computador: un PC compatible .....	252
	7.4. Paralelismo en computadores .....	254
	7.5. Conclusión .....	259
	Test .....	259
	Problemas resueltos .....	261
	Problemas propuestos .....	264
<b>CAPÍTULO 8</b>	<b>Redes de computadores e Internet .....</b>	<b>265</b>
	8.1. Redes de computadores .....	265
	8.1.1 Topología de redes .....	266
	8.1.2. Modelo OSI .....	268
	8.1.3 Redes de área local (LAN) .....	270
	8.1.4 Redes de área amplia (WAN) .....	272
	8.1.4.1. Línea telefónica convencional (analógica) .....	272
	8.1.4.2. Líneas digitales .....	272
	8.1.5 Dispositivos de interconexión en redes .....	273
	8.2. Internet .....	274
	8.2.1. Direccionamiento en Internet .....	275
	8.2.2. Protocolo TCP/IP .....	276
	8.2.3. Aplicaciones básicas de Internet .....	277
	8.2.4. World Wide Web (www) .....	279
	8.2.4.1. Direccionamiento en la web (URL) .....	279
	8.2.4.2. Protocolo HTTP .....	280
	8.2.4.3. Navegadores web .....	280
	8.2.4.4. Aplicaciones de la web .....	281
	8.3. Conclusiones .....	282
	Test .....	282
	Problemas resueltos .....	286
	Problemas propuestos .....	289
<b>CAPÍTULO 9</b>	<b>Sistemas operativos .....</b>	<b>291</b>
	9.1. El software de un computador .....	291
	9.2. Definición y evolución de sistemas operativos .....	292
	9.3. Gestión del procesador .....	294
	9.3.1. Multiprogramación .....	295
	9.3.2. Medidas de prestaciones de un sistema .....	297

9.3.3.	Modos de asignación del procesador a los procesos . . . . .	297
9.3.4.	Algoritmos de planificación . . . . .	298
9.3.5.	Modos de procesamiento . . . . .	299
9.4.	Gestión de la memoria . . . . .	300
9.4.1.	Particiones estáticas . . . . .	301
9.4.2.	Particiones dinámicas . . . . .	302
9.4.3.	Segmentación . . . . .	303
9.4.4.	Paginación . . . . .	303
9.4.5.	Memoria virtual . . . . .	305
9.5.	Gestión de entradas/salidas . . . . .	309
9.6.	Gestión de archivos . . . . .	312
9.7.	Conclusiones . . . . .	313
	Test . . . . .	313
	Problemas resueltos . . . . .	318
	Problemas propuestos . . . . .	331

## CAPÍTULO 10 Tipos y estructuras de datos . . . . . 335

10.1.	Tipos de datos . . . . .	336
10.1.1.	Enteros . . . . .	336
10.1.2.	Reales . . . . .	336
10.1.3.	Lógicos . . . . .	337
10.1.4.	Caracteres . . . . .	338
10.1.5.	Enumerados . . . . .	339
10.1.6.	Subrango . . . . .	340
10.2.	Estructuras de datos . . . . .	340
10.2.1.	Arrays . . . . .	340
10.2.2.	Cadenas de caracteres . . . . .	343
10.2.3.	Registros . . . . .	343
10.2.4.	Listas . . . . .	344
10.2.5.	Pilas . . . . .	346
10.2.6.	Colas . . . . .	347
10.2.7.	Árboles . . . . .	350
10.2.8.	Estructuras definidas por el usuario. Clases . . . . .	353
10.3.	Conclusiones . . . . .	354
	Test . . . . .	355
	Problemas resueltos . . . . .	357
	Problemas propuestos . . . . .	369

## CAPÍTULO 11 Algoritmos . . . . . 373

11.1.	Concepto de algoritmo . . . . .	273
11.2.	Representación de algoritmos . . . . .	374
11.2.1.	Pseudocódigo . . . . .	374
11.2.2.	Organigramas . . . . .	375

11.3. Estructuras de control .....	376
11.4. Subalgoritmos .....	380
11.5. Recursividad .....	381
11.6. Proceso de creación de un programa .....	382
11.7. Conclusiones .....	382
Test .....	383
Problemas resueltos .....	384
Problemas propuestos .....	393

## CAPÍTULO 12      Lenguajes de programación ..... 395

12.1. Concepto de lenguaje de programación .....	395
12.1.1. Lenguajes máquina .....	395
12.1.2. Lenguajes ensambladores .....	397
12.1.3. Lenguajes de alto nivel .....	397
12.2. Elementos de un programa en un lenguaje de alto nivel .....	398
12.2.1. Sentencias de asignación .....	400
12.2.2. Sentencias de control .....	401
12.2.3. Sentencias de entrada/salida .....	401
12.2.4. Subprogramas .....	403
12.2.5. Funciones .....	405
12.3. El proceso de traducción .....	406
12.4. Pasos a seguir para la ejecución de un programa .....	409
12.5. Clasificación de los lenguajes de programación .....	410
12.5.1. Estilos de programación .....	412
12.5.2. Dominios de aplicación de los lenguajes de programación ..	415
12.6. Conclusiones .....	416
Test .....	417
Problemas resueltos .....	419
Problemas propuestos .....	428

## CAPÍTULO 13      Archivos ..... 431

13.1. Concepto de archivo .....	431
13.2. Tipos de archivos .....	431
13.3. Organización secuencial .....	433
13.4. Organización directa .....	435
13.4.1. Acceso indexado .....	436
13.4.2. Acceso parcialmente indexado .....	437
13.4.3. Acceso calculado ( <i>hash</i> ) .....	438
13.5. Parámetros de utilización de un archivo .....	440
13.6. Ejemplo: gestión de archivos en Windows y UNIX .....	440
13.7. Conclusión .....	443
Test .....	444

Problemas resueltos . . . . .	447
Problemas propuestos . . . . .	455

## CAPÍTULO 14 Bases de datos . . . . . 459

14.1. Conceptos generales . . . . .	459
14.2. Estructura de una base de datos . . . . .	460
14.3. Sistemas de gestión de bases de datos . . . . .	462
14.4. Modelos de bases de datos . . . . .	463
14.4.1. Modelo jerárquico . . . . .	463
14.4.2. Modelo en red . . . . .	464
14.4.3. Modelo relacional . . . . .	464
14.5. Tipos de bases de datos . . . . .	473
14.5.1. Bases de datos distribuidas . . . . .	473
14.5.2. Bases de datos orientadas a objetos . . . . .	473
14.6. Conclusiones . . . . .	474
Test . . . . .	475
Problemas resueltos . . . . .	477
Problemas propuestos . . . . .	481

## CAPÍTULO 15 Ingeniería del software . . . . . 485

15.1. Introducción . . . . .	485
15.2. Planificación y gestión de proyectos . . . . .	486
15.3. Ciclo de vida del software . . . . .	487
15.4. Etapas y modelos de desarrollo de software . . . . .	487
15.5. Análisis . . . . .	490
15.6. Diseño . . . . .	492
15.6.1. Modelos de diseño . . . . .	493
15.6.2. Modularidad . . . . .	494
15.7. Implementación . . . . .	495
15.8. Prueba . . . . .	495
15.9. Calidad . . . . .	496
15.10. Conclusiones . . . . .	497
Test . . . . .	497
Problemas resueltos . . . . .	501
Problemas propuestos . . . . .	506

## APÉNDICE 1 Sistemas de numeración usuales en informática . 509

A1.1. Representación posicional de los números . . . . .	509
A1.2. Operaciones aritméticas con variables binarias . . . . .	510
A1.3. Representación en complementos . . . . .	512
A1.4. Transformaciones entre bases distintas . . . . .	513
A1.4.1. Transformaciones con la base decimal . . . . .	513

	A1.4.2. Transformaciones con la base octal . . . . .	515
	A1.4.3. Transformaciones con la base hexadecimal . . . . .	517
APÉNDICE 2	Principales códigos de Entrada/Salida . . . . .	519
APÉNDICE 3	Algunas medidas de uso común en informática .	521
APÉNDICE 4	Solución a los test . . . . .	523
BIBLIOGRAFÍA	. . . . .	525

## PRÓLOGO

Esta obra pretende facilitar al estudiante la comprensión de los conceptos básicos de la ingeniería informática, y trata, en cierta medida, de servir de complemento al texto *Introducción a la Informática*, cuyos autores son A. Prieto, A. Lloris y J. C. Torres y editado por McGraw-Hill (3.<sup>a</sup> edición, 2002). Nos ha animado a editar la recopilación de ejercicios y problemas que aquí se recogen las numerosas peticiones de nuestros alumnos de disponer de una colección de problemas amplia, con una gran parte de ellos resueltos con detalle. También tratamos de cubrir el vacío existente en el mercado editorial por la inexistencia de un libro de esta naturaleza en el campo de la introducción a la informática o a los computadores.

Los temas seleccionados tratan de cubrir tanto los aspectos hardware como software del computador. Consideramos que para entender y tener un conocimiento profundo de la Informática es necesario mostrar una perspectiva global de ella, que permita visualizarla como un todo donde encajan sus diferentes partes.

Como es tradicional en los libros de la serie Schaum, los capítulos se componen de dos partes: una primera donde se describen aspectos teóricos, que hemos procurado presentar lo más sucintamente para que el lector no se pierda en los detalles, y una segunda parte que incluye ejercicios, tanto de test como de problemas. La sección de problemas aparece, a su vez, dividida en dos partes, una de problemas con su solución explicada paso a paso y otra de problemas planteados para ser resueltos por el lector. Las soluciones de los test aparecen en el último apéndice (Apéndice 4).

Deseamos agradecer a las personas que han colaborado directa o indirectamente en la confección de este libro de problemas, especialmente a los coautores del texto al que se pretende complementar, profesores Antonio Lloris y Juan Carlos Torres. También hay que resaltar la contribución de los compañeros del Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Granada, principalmente a Begoña del Pino, Carlos García Puntonet, Ignacio Rojas, Eduardo Ros, David Palomar, Jesús González, Héctor Pomares, Antonio Prados, José Luís Bernier, Pedro Castillo y Fernando Rojas. A Juan María Prieto, Comandante de Iberia, le agradecemos nos haya facilitado la terminología que utilizamos en algunos ejemplos de los Capítulos 13 y 14. En otro orden de cosas, nuestra gratitud a Concepción Fernández Madrid, Directora Editorial de la División Universitaria de McGraw-Hill Interamericana, por sus ánimos y buen quehacer editorial.

Esperamos sinceramente que con esta obra les sea más fácil a los alumnos asimilar las ideas básicas sobre el funcionamiento y uso de los computadores, y, en consecuencia, ello redunde en un mayor éxito en sus calificaciones de las asignaturas correspondientes, objetivo arduosamente buscado no sólo por ellos sino también por sus profesores.

Granada, 6 de enero de 2005.

ALBERTO y BEATRIZ PRIETO





# Introducción

En este capítulo se presentan unas definiciones y nociones generales sobre informática, la mayoría de las cuales son ampliadas en los siguientes capítulos. Pretendemos que el lector, además de comprender la terminología más usual de la informática, tenga una visión panorámica del contenido de este libro, de forma tal que, cuando se adentre en los sucesivos capítulos, sepa enmarcar el sentido e importancia de cada uno de ellos.

De esta forma en la Sección 1.1 se presentan las definiciones básicas de informática, computador, codificación, bit y byte. En la Sección 1.2 se esboza la estructura física de un computador en términos de sus unidades funcionales (entradas, salidas, procesador y memoria). A los conceptos de programas e instrucciones se les dedica la Sección 1.3. Posteriormente (Sección 1.4) se presentan distintas clasificaciones de los computadores para a continuación (Sección 1.5) describir los distintos niveles conceptuales bajo los que se puede analizar o diseñar un computador. El capítulo concluye (Sección 1.6) relacionando los principales campos de aplicación de la informática.

## 1.1. Conceptos básicos

Informática es una palabra de origen francés formada por la contracción de los vocablos **INFOR**mación y auto**MÁTICA**. La Real Academia Española define la **Informática** como el conjunto de conocimientos científicos y técnicas que hacen posible el tratamiento automático de la información por medio de ordenadores.

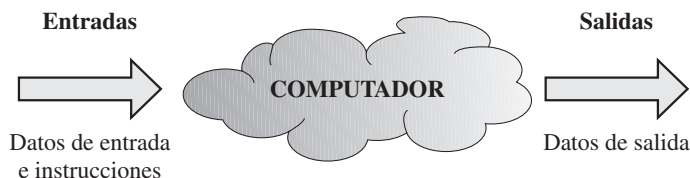
El término información hace referencia aquí a la yuxtaposición de símbolos, con los que se representan convencionalmente hechos, objetos o ideas.

La informática, como disciplina, avanza gracias a la utilización de las metodologías seguidas para los desarrollos de tipo teórico y de tipo experimental, así como para el diseño de sistemas, por lo que puede considerarse tanto una ciencia como una ingeniería. La disciplina de Informática es el cuerpo de conocimiento que trata del diseño, análisis, implementación, eficiencia y aplicación de procesos que transforman la información.

**Computador, computadora u ordenador** es una máquina capaz de aceptar unos datos de entrada, efectuar con ellos operaciones lógicas y aritméticas, y proporcionar la información resultante a través de un medio de salida; todo ello mediante el control de un programa de instrucciones previamente almacenado en el propio computador.

Se entiende por operaciones lógicas funciones tales como comparar, ordenar, seleccionar o copiar símbolos, ya sean numéricos o no numéricos.

Un computador puede considerarse como un sistema, Figura 1.1, cuyas salidas o resultados son función (dependen) de sus entradas, constituidas por datos e instrucciones.



**Figura 1.1.** El computador como sistema que actúa con su exterior.

Considerando la definición de computador, se puede decir que informática o ciencia o ingeniería de los computadores es el campo de conocimiento que abarca todos los aspectos del diseño y uso de los computadores.

De acuerdo con su definición, el computador actúa con dos tipos de informaciones: instrucciones (que indican a la máquina qué es lo que tiene que hacer) y datos (que son los elementos que procesa o genera el programa). Es decir, la palabra dato se utiliza como contraposición a instrucción, y en informática se utiliza con un sentido más amplio que en física o matemáticas, por ejemplo, ya que aquí se considera que un dato es un símbolo, o conjunto de ellos, utilizado para expresar o representar un valor numérico, un hecho, un objeto o una idea en la forma adecuada para ser objeto de tratamiento.

### EJEMPLO 3.1

En informática no sólo es dato una temperatura (25 °C) o una altura (38,5 m), o una medida experimental, sino que también lo es una matrícula de coche (7784 BBZ), el nombre de un individuo (Marta Prieto Campos) o una frase de un libro.

En informática es frecuente codificar la información. Codificación es una transformación que representa los elementos de un conjunto mediante los de otro, de forma tal que a cada elemento del primer conjunto le corresponda un elemento distinto del segundo.

### EJEMPLO 3.2

Ejemplos de códigos son:

- La matrícula de un coche; que es una identificación más corta que hacerla, por ejemplo, por el nombre de su propietario, su marca, color y fecha de compra.
- El código postal asociado a las viviendas de un Estado.
- El código de enfermedades definido por la Organización Mundial de la Salud (OMS). A cada enfermedad se le asigna un código.
- El número de un carné de identidad. A cada persona se le asocia un número, pudiendo referirse administrativamente a ella por medio de ese código.

En el interior de los computadores la información se almacena y se transfiere de un sitio a otro según un código que utiliza sólo dos valores (un código binario) representados por 0 y 1. En la entrada y salida del computador se efectúan automáticamente los cambios de código oportunos para que en su exterior la información sea directamente comprendida por los usuarios. Todos los aspectos relativos a la representación de la información, tanto en el exterior como en el interior del computador, se detallan en el Capítulo 2.

La unidad más elemental de información es un valor binario, conocido como **bit**, que representa una posición o variable que toma el valor 0 o 1. La capacidad de memoria de un computador puede medirse en bits; aunque en la práctica no se hace así, por ser el bit una unidad excesivamente pequeña.

Otra unidad de información muy utilizada es el **byte**, que tradicionalmente era el número de bits necesarios para almacenar un carácter; sin embargo, en la actualidad se utiliza como sinónimo de grupo de 8 bits. La capacidad de almacenamiento de un computador o de un soporte de información (tal como un disco magnético u óptico) se suele medir en bytes. Como el byte es una unidad relativamente pequeña, es usual utilizar los siguientes múltiplos, que son similares a los utilizados en física, pero con valores diferentes:

1 **Kilobyte** (o KB) =  $2^{10}$  bytes = 1.024 bytes  $\approx$  103 bytes.

1 **Megabyte** (o MB) =  $2^{20}$  bytes = 1.048.576 bytes  $\approx$   $10^6$  bytes.

1 **Gigabyte** (o GB) =  $2^{30}$  bytes = 1.073.741.824 bytes  $\approx$   $10^9$  bytes.

1 **Terabyte** (o TB) =  $2^{40}$  bytes  $\approx$   $10^{12}$  bytes.

1 **Petabyte** (o PB) =  $2^{50}$  bytes  $\approx$   $10^{15}$  bytes.

1 **Exabyte** (o EB) =  $2^{60}$  bytes  $\approx$   $10^{18}$  bytes.

Los prefijos anteriores (K, M, G, T, P y E) no sólo se utilizan con bytes, sino también con otras unidades internas de información. Así 1 Gb (o gigabit) son 1.073,741.824 bits.

En general, utilizaremos de aquí en adelante una “b” para indicar bit y una “B” para indicar byte.

### EJEMPLO 3.3

Se verifican las siguientes igualdades:

$$1 \text{ MB} = 1.024 \text{ KB} = 8 \cdot 1.024 \cdot 1.024 \text{ bits} = 8.388.608 \text{ bits}$$

## 1.2. Estructura funcional de los computadores

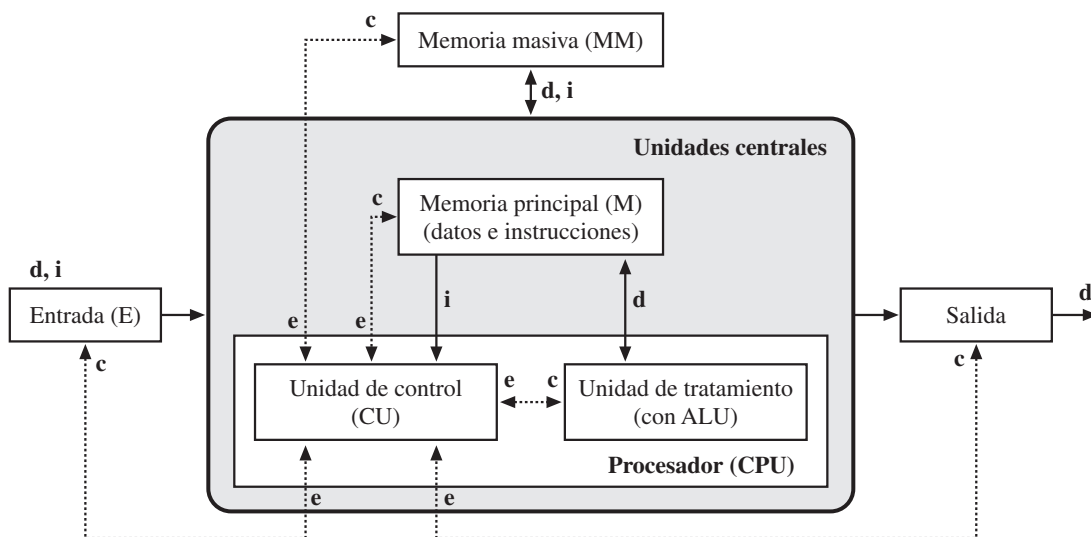
En esta sección se describen los elementos básicos de un computador. Para ello en primer lugar (Sección 1.2.1) se analizan las unidades básicas que lo constituyen, después (Sección 1.2.2) se definen parámetros que sirven para determinar la capacidad y rendimiento de un computador.

### 1.2.1. UNIDADES FUNCIONALES

En la Figura 1.2 puede verse un esquema general de un computador sencillo, que se compone de los siguientes elementos funcionales: unidades de entrada, unidades de salida, memoria principal, memoria masiva, unidad aritmético-lógica y unidad de control. Este diagrama corresponde a los primeros computadores electrónicos, denominados **computadores von Neumann**, pero sigue siendo conceptualmente válido hoy día. A continuación se describen brevemente las distintas unidades funcionales, que serán estudiadas con más detalle en los Capítulos 4 y 5 del presente texto.

#### • Unidad de entrada (E)

Es un dispositivo por donde se introducen en el computador los datos e instrucciones. En estas unidades se transforma la información de entrada en señales binarias de naturaleza eléctrica. Un mismo computador puede tener distintas unidades de entrada (en la Figura 1.2 se incluye una sola por simplificar). Son unidades de entrada: un teclado, un ratón, un escáner de imágenes, una lectora de tarjetas de crédito, etc.



d: dato; i: instrucciones; e: señales de estado; c: señales de control.

**Figura 1.2.** Esquema que muestra las unidades funcionales de un computador.

- **Unidad de salida (S)**

Es un dispositivo por donde se obtienen los resultados de los programas ejecutados en el computador. La mayor parte de estas unidades (un computador suele tener varias de ellas) transforman las señales eléctricas binarias en información perceptible por el usuario. Son dispositivos de salida unidades tales como una pantalla, una impresora o un altavoz.

- **Memoria principal (M)**

Es la unidad donde se almacenan tanto los datos como las instrucciones, durante la ejecución de los programas. La memoria principal actúa con una gran velocidad y está ligada directamente a las unidades más rápidas del computador (unidad de control y unidad aritmético-lógica). Para que un programa se ejecute debe estar almacenado (*cargado*) en la memoria principal. En los computadores actuales la memoria está formada por circuitos electrónicos integrados (chips).

Normalmente hay una zona de la memoria que sólo se puede leer (**memoria ROM**) y que es permanente (al desconectar el computador su información no se pierde), y otra en la que se puede leer y escribir (**memoria RAM**) y que es volátil. La memoria ROM de los computadores viene grabada de fábrica, y contiene programas y datos relevantes del sistema operativo que deben permanecer constantemente en la memoria principal.

- **Memoria masiva (MM)**

La memoria principal es muy rápida (puede leer o escribir millones de palabras en un solo segundo), pero no tiene gran capacidad para almacenar información y su zona RAM es volátil. Para guardar masivamente información se utilizan otros tipos de memoria, tales como discos magnéticos, discos ópticos y cintas magnéticas, que son más lentos pero tienen mucha más capacidad que la memoria principal (del orden de un millón de veces más lentos y de mil veces más capaces, en el caso de un disco magnético). El conjunto de estas unidades se denomina **memoria masiva**, **memoria auxiliar**, **memoria externa** o **memoria secundaria**. Usualmente los datos y programas se graban en la memoria

masiva, de forma que cuando se ejecuta varias veces un programa o se utilizan repetidamente unos datos no es necesario darlos de nuevo a través del dispositivo de entrada. La información guardada en un disco o cinta permanece indefinidamente hasta que el usuario expresamente la borre.

- **Unidad aritmético-lógica (ALU)**

La unidad aritmético-lógica o ALU (*Arithmetic Logic Unit*) contiene los circuitos electrónicos con los que se hacen las operaciones de tipo aritmético (sumas, restas, etc.) y de tipo lógico (comparar dos números, hacer operaciones del álgebra de boole binaria, etc.). Esta unidad también suele denominarse **unidad de tratamiento** o **camino de datos** (o **ruta de datos**) ya que aparte de contener los circuitos específicos para realizar las operaciones aritmético-lógicas (ALU, propiamente dicha), incluye otros elementos auxiliares por donde se transmiten (**buses**) o almacenan temporalmente (**registros**) los datos para operar con ellos.

- **Unidad de control (CU, *Control Unit*)**

La unidad de control detecta *señales eléctricas de estado* procedentes de las distintas unidades, indicando su situación o condición de funcionamiento (Figura 1.2). También capta secuencialmente de la memoria las instrucciones del programa, y, de acuerdo con el código de operación de cada una de ellas y con las **señales de estado** procedentes de los distintos elementos del computador, genera **señales de control** dirigidas a todas las unidades, ordenando las operaciones que implican la ejecución de la instrucción.

La unidad de control contiene un **reloj**, que sencillamente es un **generador de pulsos** que sincroniza todas las operaciones elementales del computador. El período de esta señal se denomina **tiempo de ciclo** ( $T$ ), y es del orden de nanosegundos. La **frecuencia del reloj**,  $F$ , es el inverso del tiempo de ciclo:

$$F = \frac{1}{T} \quad [1.1]$$

y suele darse en millones de ciclos/segundo (megahercios o, abreviadamente, MHz), o en miles de millones de ciclos/segundo (gigahercios, GHz). La ejecución de cada instrucción supone la realización de un conjunto de operaciones elementales consumiendo un número predeterminado de ciclos; dependiendo de la arquitectura de la unidad de control y de la complejidad de la instrucción, ésta puede consumir más o menos ciclos de reloj.

Unas unidades están interconectadas con otras según se indica en la Figura 1.2; ahora bien, existen diversas variantes de este esquema, dependiendo de la estructura o configuración concreta del computador, según se analizará en la Sección 7.1. Los distintos elementos de un computador se interconectan a través de conjuntos de hilos, líneas o pistas eléctricamente conductores que suelen llevar en un instante dado (*en paralelo*) la información completa de una instrucción, un dato o una dirección. Un conjunto de conductores que transmite información del mismo tipo entre unidades distintas se denomina **bus**. El **ancho de un bus** es el número de hilos que contiene, o número de bits que transmite simultáneamente, en paralelo.

Los distintos elementos de un computador se suelen agrupar como se indica en la Figura 1.3. Los **periféricos** de un computador son sus unidades de E/S y de memoria masiva. Al resto de unidades, es decir, memoria principal y unidades de control y ALU, las denominaremos **unidades centrales**. La **unidad de procesamiento central** (CPU, *Central Processing Unit*) o sencillamente **procesador** es el conjunto de unidad de control y unidad de tratamiento.

El grado de miniaturización alcanzado en la integración de circuitos electrónicos ha llegado a ser tan alto que en un único chip se pueden incluir todos los elementos de un procesador. Un **microprocesador** es un procesador (CPU) implantado en un circuito integrado (o en un conjunto reducido de ellos).

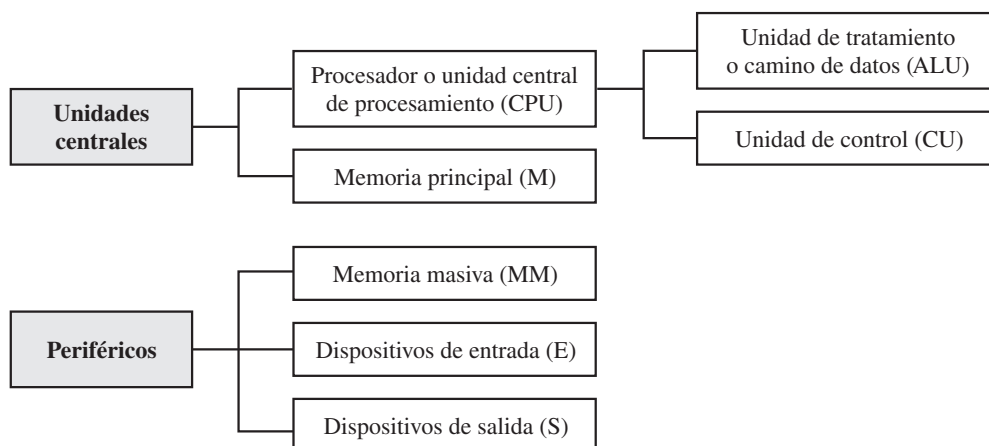


Figura 1.3. Agrupaciones de las unidades funcionales de un computador.

### 1.2.2. PARÁMETROS PARA CARACTERIZACIÓN DE PRESTACIONES

Existen varias magnitudes que determinan las prestaciones de las distintas unidades que componen un computador. En relación con ellas a continuación vamos a examinar los conceptos de capacidad de almacenamiento, tiempo de acceso, longitud de palabra, ancho de banda y rendimiento del procesador. Otros parámetros de interés, como la productividad y tiempo de respuesta, y que dan una valoración global del sistema, considerando tanto el comportamiento de la máquina como del sistema operativo, se considerarán en el Capítulo 9 (Sección 9.3.2).

La **capacidad de almacenamiento** se refiere a las posibilidades de una unidad para almacenar datos o instrucciones de forma temporal o fija. El procesador contiene registros de uso general, y su capacidad de almacenamiento viene dada por el número de ellos y su longitud. La capacidad de la memoria principal y dispositivos de memoria masiva se da en bytes (MB, GB, TB, etc.).

El **tiempo de acceso** de una unidad es el intervalo de tiempo que transcurre desde el instante en que se proporciona a la misma la **posición** concreta del dato o instrucción que se quiere leer o escribir, y el instante en que se obtiene (lee) o graba (escribe) el mismo.

Para muchas operaciones que se realizan en el computador, el byte (8 bits) es una unidad de información pequeña; así las ALU suelen operar con datos de mayor longitud, normalmente de un número entero de bytes: 8, 16, 32, 64 o 128 bits. Se denomina **palabra** al conjunto de bits que forma un dato con los que operan la ALU, y coincide, en general, con el número de bits de cada uno de los registros del procesador. La **longitud de una palabra** es el número de bits que la forman; así, si una ALU opera con datos de 32 bits, la longitud de palabra de ese procesador es de 32 bits. Con frecuencia la longitud de la palabra coincide con el ancho del bus de datos que conecta el procesador con la memoria. También se habla de **palabra de memoria**, que es sencillamente la información que se graba en cada una de las posiciones especificadas a través del bus de direcciones. Este último término es más confuso ya que normalmente la memoria se suele organizar en módulos que actúan en paralelo y pueden escribirse o captarse datos de distinta longitud. En efecto, en la mayoría de los computadores de longitud de palabra de 32 bits, el direccionamiento a memoria se efectúa por bytes, y es posible acceder directamente a bytes (8 bits), medias palabras (16 bits) y palabras (32 bits); algunos incluso permiten acceder a dobles palabras (64 bits).

Otro parámetro de interés ligado a la implementación del computador es el **ancho de banda**, que representa la cantidad de información transferida por segundo entre una unidad y otra. Por ejemplo, decir que el *ancho de banda entre la memoria y el procesador es de 133 MB/s*, quiere decir que en un segundo se pueden transferir 133 megabytes entre las unidades citadas.

Desde el punto de vista de los usuarios interesa una medida más global del funcionamiento del computador, que pueda servir para comparar dos equipos distintos. Dado un determinado programa diremos que un computador tiene un mayor rendimiento que otro, si el primero lo ejecuta en menos tiempo.

El tiempo de ejecución de un programa,  $T_E$ , es el tiempo que transcurre desde su inicio hasta que finaliza su ejecución. El rendimiento de un computador en la ejecución de un programa es la inversa de su tiempo de ejecución. Si denominamos  $N_I$  al número de instrucciones que se ejecutan en un programa<sup>1</sup>, y  $N_{CI}$  al número medio de ciclos de reloj que consume cada instrucción, el tiempo de ejecución del programa vendrá dado por:

$$T_E = N_I \cdot N_{CI} \cdot T = \frac{N_I \cdot N_{CI}}{F} \quad [1.2]$$

Por otra parte, el **rendimiento** es la inversa del tiempo de ejecución:

$$\eta_E = \frac{1}{T_E} \quad [1.3]$$

Uno de los objetivos básicos de la arquitectura de computadores es reducir el valor de  $T_E$ , para lo cual se debe aumentar  $F$  o disminuir  $N_I$  y  $N_{CI}$ . El valor de  $F$  viene determinado por la velocidad de funcionamiento de los circuitos integrados, y es responsabilidad de la tecnología electrónica; sin embargo, la disminución de  $N_I$  y  $N_{CI}$  viene determinada por la arquitectura del computador; aunque los tres parámetros están interrelacionados. Claramente de la expresión [1.2] se deduce que un procesador con mayor frecuencia que otro puede ser más lento ejecutando el mismo programa.

### EJEMPLO 3.4

Según indicamos anteriormente, la ejecución de cada instrucción para un procesador consume un número determinado de ciclos de reloj, existiendo instrucciones más rápidas que otras. Así, por ejemplo, en el procesador Pentium III una instrucción puede consumir 14 ciclos de reloj, mientras que otra puede consumir 45. Debido a lo anterior no se puede comparar el rendimiento de dos computadores con procesadores diferentes por sus frecuencias de reloj, ya que un computador  $A$  con frecuencia de reloj menor que otro  $B$  puede que esté diseñado de forma que sus instrucciones se ejecuten consumiendo menos ciclos que el  $B$ , pudiendo ser en este caso, por tanto, el rendimiento del  $A$  mayor que el del  $B$ . En realidad, como hemos puesto de manifiesto con la expresión [1.2], el rendimiento de un procesador depende de tres factores: el repertorio de instrucciones (que determina el valor de  $N_I$ ), la frecuencia de reloj ( $F$ ) y el número de ciclos asociados a cada instrucción ( $N_{CI}$ ).

Con frecuencia es necesario comparar el rendimiento de dos o más computadores. Pero hemos definido el rendimiento para la ejecución de un programa concreto, y el hecho de que un programa se ejecute más rápidamente en un computador que en otro no implica que vaya a ocurrir lo mismo para cualquier otro programa (depende de las instrucciones concretas que formen cada programa). Para poder evaluar lo más correctamente posible el rendimiento de un computador, la comunidad informática ha establecido unos **conjuntos de programas de prueba** (*benchmark*). De acuerdo con la definición de rendimiento dada, el rendimiento de un computador se puede dar dividiendo el número (en millones) de instrucciones total del conjunto de programas de prueba entre el tiempo total de su ejecución, y se da, o bien en **MIPS** (millones de instrucciones por segundo) si las operaciones se realizan con datos en forma de números enteros, o bien en **MFLOPS** (Mega FLOPS o millones de ins-

<sup>1</sup> Este número no tiene por qué coincidir con el número de instrucciones que tiene el programa, ya que éste puede contener bucles o lazos que se ejecuten múltiples veces e instrucciones en ramas del programa que no se ejecuten.



trucciones de coma flotante por segundo) si se realizan con datos en forma de números reales (de coma flotante, ver Sección 3.5).

Uno de los conjuntos de prueba más conocidos y completos es el SPEC (*System Performance Evaluation Cooperative* o conjunto para evaluación del rendimiento de computadores). El conjunto SPEC da como medida un valor relativo a un procesador determinado, así el SPEC95 utiliza como referencia el computador Sun SPARCstation 10/40 y el SPEC2000 un computador UltraSPARC10 con un procesador UltraSPARC-III de  $F = 300$  MHz, de esta forma la medida no tiene dimensiones. El SPEC2000 incluye 19 aplicaciones de prueba nuevas (compresión de datos, procesamiento de textos, juego del ajedrez, etc.), no consideradas previamente en el conjunto SPEC95.

### EJEMPLO 3.5

El Pentium Pro de 200 MHz tiene un valor SPEC95 de valor 8 para enteros, y 7 para reales, lo que quiere decir que se considera 8 veces más rápido que el procesador Sun citado en aplicaciones que utilicen números enteros, y 7 en el caso de números reales.

La velocidad *SPEC* para cada programa,  $i$ , de prueba se obtiene de acuerdo con la siguiente expresión:

$$v_{SPEC,i} = \frac{T_{Et} \text{ (en computador a caracterizar)}}{T_{Et} \text{ (en computador de referencia)}} \quad [1.4]$$

y la velocidad *SPEC* final del computador se obtiene con la media geométrica de las velocidades de los  $n$  programas de referencia establecidos:

$$v_{SPEC} = \sqrt[n]{\prod_{i=1}^n v_{SPEC,i}} \quad [1.5]$$

## 1.3. Programas e instrucciones

Una **instrucción** es un conjunto de símbolos que representa una orden de operación o tratamiento para el computador. Las operaciones suelen realizarse con datos.

Un **programa** es un conjunto ordenado de instrucciones que se dan al computador indicándole las operaciones o tareas que se desea realice.

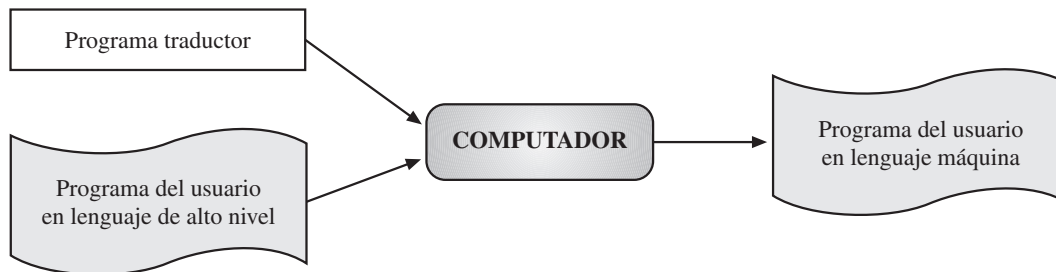
Las instrucciones se forman con elementos o símbolos tomados de un determinado repertorio, y se construyen siguiendo unas reglas precisas.

Todo lo relativo a los símbolos y reglas para construir o redactar con ellos un programa se denomina **lenguaje de programación**.

La unidad de control del computador sólo puede interpretar instrucciones de un determinado lenguaje, denominado **lenguaje máquina**. Las instrucciones de este lenguaje están formadas por bits (ceros y unos) agrupados usualmente en al menos dos bloques o campos. Uno de ellos es el código de operación y el otro una dirección. El **código de operación** (abreviadamente, **codop**) indica la operación correspondiente a la instrucción. Este código se obtiene de una tabla o repertorio en el que figuran las instrucciones, que puede ejecutar la máquina y el código binario asociado a cada una de ellas. El **campo de dirección** especifica el lugar (posición de memoria o registro) dónde se encuentra el dato o los datos con los que hay que operar, o que hay que transferir, o dónde hay que llevarlos, en consonancia con el **codop**.

El lenguaje máquina tiene serios inconvenientes, como son: depende del procesador, el repertorio de instrucciones es muy reducido (conteniendo sólo operaciones muy elementales), es muy laborioso programar con él por tener que utilizar sólo números, etc. Para evitar estos problemas se han ideado **lenguajes de alto nivel**, que no dependen del computador, y se han proyectado pensando en facilitar

la tarea de programación. Es necesario disponer de unos programas **traductores**, que al ejecutarlos en el propio computador e introduciendo como datos programas escritos en el lenguaje de alto nivel, generan como resultado programas en lenguaje máquina (Figura 1.4).



**Figura 1.4.** Proceso de traducción de un programa en lenguaje de alto nivel (LAN) a lenguaje máquina (LM).

Son lenguajes de alto nivel FORTRAN, COBOL, BASIC, Lisp, Prolog, Logo, Pascal, C, Ada, C++, Java, Visual Basic, etc. (Capítulo 12).

Además de los programas traductores, el constructor del computador proporciona otros programas que son necesarios para el control y para la utilización eficiente y cómoda del computador. El conjunto de programas que controla y gestiona los recursos del computador se denomina sistema operativo. Un **sistema operativo** es una colección de programas que juntos suministran una interfaz entre el hardware del computador y los usuarios, facilitando su uso, y realizan el control adecuado y la asignación de recursos del sistema para asegurar un funcionamiento adecuado y eficaz. Los programas del sistema operativo se utilizan con un lenguaje específico denominado **lenguaje de control**. A las instrucciones del lenguaje de control se les denomina **órdenes** (*commands*). El Capítulo 9 de esta obra se dedica a estudiar los sistemas operativos.

## 1.4. Tipos de computadores

Los computadores pueden clasificarse atendiendo a distintos criterios que se resumen en la Figura 1.5. La clasificación de los computadores atendiendo al modo de representar físicamente la información se describirá en el Capítulo 3 y el paralelismo se analizará en el Capítulo 7.

### 1.4.1. CLASIFICACIÓN SEGÚN LA GENERALIDAD DE SU USO

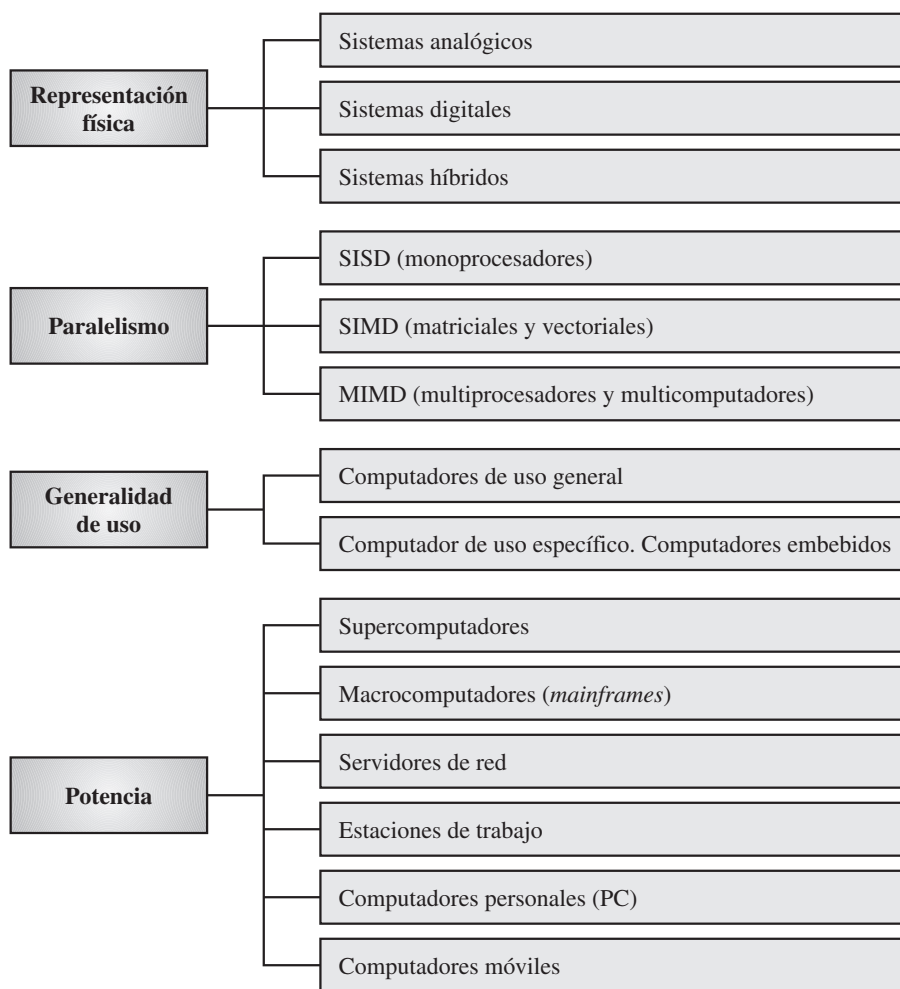
Este criterio de clasificación hace referencia al *uso* o *propósito* para el que fueron diseñados y contruidos.

#### Computador de uso general

Es el computador que puede utilizarse para distinto tipo de aplicaciones, tales como gestión administrativa, cálculo científico o cálculos técnicos. Que realice una aplicación u otra depende del programa que el usuario ordene ejecutar.

#### Computador de uso específico

Es el computador que *únicamente* puede utilizarse para un grupo de aplicaciones determinado o una aplicación muy concreta, y por lo general se construye con microprocesadores. Ejemplos de compu-



**Figura 1.5.** Criterios para clasificación de los computadores.

tadores de uso específico son: un videojuego de bolsillo, el computador que contiene un robot, el que contiene un misil para guiar su trayectoria, o un computador para control de tráfico. Por lo general estos computadores tienen la estructura indicada en la Sección 1.2.1, siendo válido para ellos todo lo indicado en este y sucesivos capítulos. La mayoría de computadores de uso específico son **computadores embebidos**, que forman parte de algún sistema pero no se puede acceder a ellos directamente, tal es el caso de los relojes de cuarzo, cámaras, grabadores de vídeo y otros muchos equipos domésticos e industriales.

#### 1.4.2. CLASIFICACIÓN ATENDIENDO A LA POTENCIA DE CÁLCULO

Una clasificación muy frecuente se hace atendiendo a la potencia o capacidad del computador. Esta clasificación es muy difusa, y se efectúa considerando parámetros tales como longitud de palabra, velocidad de funcionamiento, capacidad de la memoria principal y número de terminales (o usuarios) interactivos conectables. Describiremos esta clasificación de mayor a menor potencia.

### Supercomputadores

La característica fundamental de este tipo de computadores es su rapidez; son de longitud de palabra grande (64 bits), con varios procesadores o computadores trabajando en paralelo (sistemas multiprocesador o multicomputador, ver Capítulo 7). Se utilizan para realizar cálculos intensivos a gran velocidad con grandes cantidades de datos, que son necesarios en muy diversas aplicaciones de tipo científico y de ingeniería, tales como diseño de nuevas medicinas, simulación del modelo de polución de una gran ciudad (que involucra a cientos de miles de variables), modelado del flujo de aire alrededor de un prototipo de avión, modelado de fisión nuclear, simulación del enfriamiento de las galaxias, etc. Una aplicación típica es la predicción climatológica, ya que requiere efectuar cálculos muy complejos (que simulan matemáticamente el comportamiento de la atmósfera), con grandes cantidades de datos provenientes de multitud de terminales (del orden de 10.000) ubicados en estaciones meteorológicas, y a tiempo para poder realizar las predicciones.

### Macrocomputadores o grandes computadores (*Mainframes*)

En este grupo se incluyen los computadores clásicos. Son grandes computadores de uso general con amplias posibilidades de procesamiento, gran memoria y terminales remotos de E/S. Son utilizados por instituciones que procesan la información de grandes **bases de datos**. Otras características típicas de estos computadores es procesar la información en modo de *transacciones* (ver Capítulo 9), con gran cantidad de usuarios conectados a través de redes y disponer de una gran capacidad de memoria masiva.

### Servidores de red

Son equipos de rango medio, para utilizar interactivamente por múltiples usuarios simultáneamente, similares a los macrocomputadores, pero a escala reducida de prestaciones y precio, y suelen ser utilizados en empresas o departamentos de tipo medio. Actúan interconectados en una red de área local o de gran área (internet), pudiendo atender simultáneamente de decenas a cientos de accesos de estaciones de trabajo, PC o terminales conectados a la red.

### Estaciones de trabajo (*Workstations*)

Estos equipos suelen utilizarse en forma monousuario y pueden considerarse la gama alta de los computadores personales. Su diseño inicialmente fue proyectado para aplicaciones que requieren mucho cálculo y una alta capacidad gráfica, como pueden ser las de diseño con ayuda de computador, las de animación, etc.

### Computadores personales (PC, *Personal Computers*)

Son microcomputadores monousuario, usualmente con decenas de MB de memoria principal, discos duros con decenas de GB, unidad CD-ROM, módem, tarjeta de sonido y otros periféricos. Es el tipo de computadores más difundido y entre las características de estos sistemas están la gran cantidad de programas disponibles para ellos, y la gran compatibilidad entre unos y otros.

### Computadores móviles

Se caracterizan por su pequeño tamaño (aproximadamente el de una agenda,  $20 \times 10 \times 4$  cm), peso reducido (unos 300 g) y alimentación a baterías; todo ello para obtener una gran movilidad. En este grupo incluimos, además de las calculadoras programables de bolsillo, una serie de sistemas que en la actualidad están proliferando notablemente y que se suelen utilizar para aplicaciones tales como creación de pequeñas hojas de cálculo, agenda de direcciones, planificación horaria, directorio tele-

fónico. Entre otros sistemas aquí podemos incluir: **asistentes digitales personales** (*Personal Digital Assistant, PDA*), **computadores de bolsillo** (*Palmtop Computers*), **organizadores o agendas personales** (*Personal Organizers*), **comunicadores personales** y **calculadoras programables de bolsillo**.

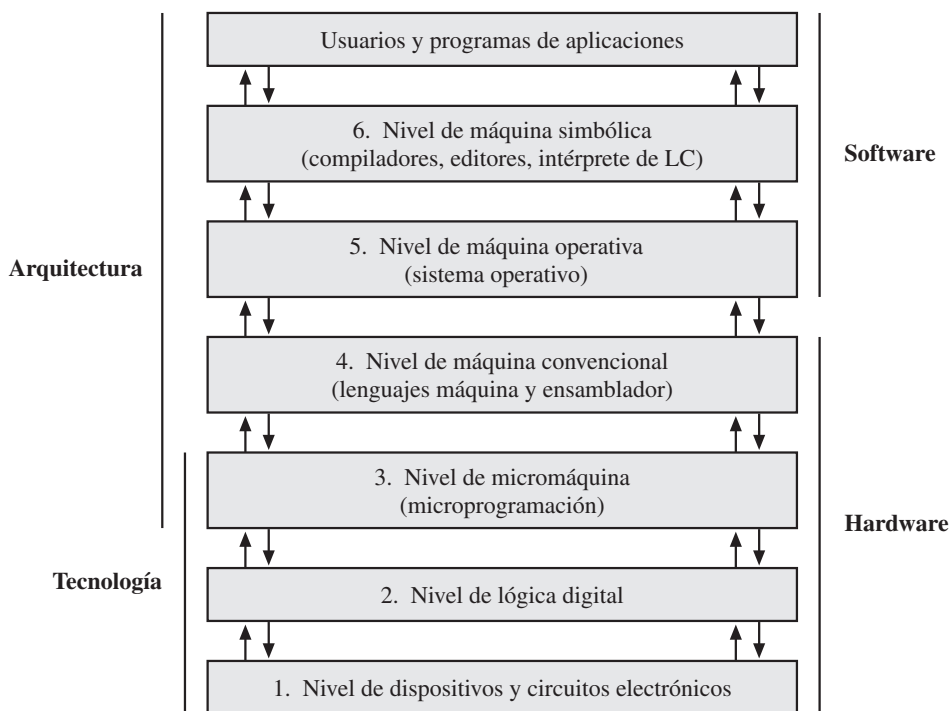
## 1.5. Niveles conceptuales de descripción de un computador

El **hardware** o **soporte físico** de un computador es la máquina en sí: el conjunto de circuitos electrónicos, cables, armarios, dispositivos electromecánicos, y otros elementos físicos que forman el computador.

El **software** o **soporte lógico** de un computador es el conjunto de programas (sistema operativo, utilidades, programas de los usuarios, etc.) ejecutables por el computador.

Para que un computador funcione es necesario utilizar programas; es decir, un computador con tan sólo sus elementos físicos no actúa: tan imprescindible es el hardware como el software.

Para facilitar el análisis o diseño de un computador se suele describir según distintos niveles conceptuales. La distinción entre niveles más sencilla es la que hay entre software y hardware. Una distinción más detallada (de nivel de mayor complejidad al de menor complejidad) es la que se muestra en la Figura 1.6.



**Figura 1.6.** Niveles conceptuales de descripción de un computador.

El estudio o diseño dentro de cada nivel se efectúa utilizando o *viendo* al computador según los elementos constructivos o primitivas proporcionadas por el nivel inmediato inferior. Así, para diseñar el sistema operativo se considera al computador en el nivel de máquina convencional; es decir, en teoría no sería necesario conocer niveles inferiores a éste (el de lógica digital, por ejemplo).

Los niveles 1, 2 y 3 corresponden a los de **tecnología de computadores**, y son de una gran importancia ya que el crecimiento del rendimiento de los computadores se ha debido en gran parte a mejoras introducidas en estos niveles. Los elementos del **nivel electrónico** son componentes tales como transistores, condensadores y resistencias; y la información se representa con valores de tensiones eléctricas (0/3.5 V, por ejemplo), corriente eléctricas (0,16 mA) o estados de magnetización (Norte, Sur), etc. La mayoría de las operaciones básicas de la máquina se describen en el **nivel de lógica digital**, y sus elementos básicos (puertas lógicas) pueden almacenar, manipular y transmitir datos representables en forma binaria (0, 1). El Capítulo 3 se dedica a sentar las bases tecnológicas de los computadores, que posibilitan la descripción de los mismos en los niveles 1 y 2.

En los Capítulos 4 y 5 se describirán algunos tópicos de la estructura de computadores dentro del nivel 3 (**nivel de microoperaciones**). Los módulos que utiliza este nivel son registros, multiplexores, bancos de registros, módulos de memoria RAM, etc., construidos con elementos del nivel 2 (puertas lógicas). Las señales son para realizar operaciones tales como carga de un registro, efectuar una operación concreta en la ALU, escribir en la memoria principal, etc.

El nivel 4, **nivel de lenguaje máquina**, es el genuino nivel de **arquitectura del computador**, ya que el diseño de un computador parte de la especificación de las instrucciones del lenguaje máquina. A este nivel se dedicará el Capítulo 4. El lenguaje ensamblador es igual al lenguaje máquina, salvo que utiliza una terminología distinta para describir los módulos o primitivas (las instrucciones) que se definen en este nivel.

El nivel 5, **sistema operativo**, puede considerarse como la interfaz entre el hardware y el software. Añade una capa para facilitar el uso del hardware y hacerlo lo más eficaz posible desde el punto de vista de los usuarios y de los programas de aplicación. A él se dedica el Capítulo 9.

El último nivel (nivel 6, o **nivel de máquina simbólica**) es el que realmente ven los programadores de aplicaciones y los usuarios, y está formado básicamente por los traductores o, en general, procesadores de lenguajes. Si está bien concebido, el usuario no tendrá necesidad de conocer nada de los niveles inferiores, y sus primitivas y lenguaje de utilización son simbólicos, debiendo ser lo más próximos posibles al hombre (Capítulo 12).

## 1.6. Aplicaciones de la informática

La **informática** tiene por objeto el tratamiento automático de la información (Sección 1.1), y hay pocas actividades humanas en las que no tenga incidencia, de forma directa o indirecta. Los computadores resultan útiles para aplicaciones que reúnen una o varias de las siguientes características:

1. Necesidad de un *gran volumen de datos*.
2. *Datos comunes*. Las bases de datos posibilitan que los datos incluidos en un computador puedan utilizarse en múltiples aplicaciones, sin necesidad de que estén físicamente repetidos.
3. *Repetitividad*. Tal vez una de las características más relevante de los lenguajes de programación es poder programar la ejecución de ciclos de instrucciones iterativamente.
4. *Distribución*. El origen y destino de la información no necesita estar ubicado en el computador central.
5. *Precisión*. Un computador puede realizar todas sus operaciones con una *precisión* controlada, obteniendo resultados consistentes con la precisión de los datos introducidos.
6. *Cálculos complejos*. Utilizando lenguajes de programación adecuados y rutinas de bibliotecas matemáticas, es posible efectuar cálculos sofisticados (resolución de modelos matemáticos atmosféricos para predicción del tiempo, por ejemplo).
7. Las operaciones que realiza un computador las efectúa a una *gran velocidad*, en comparación con los humanos.

**1. Procesamiento de datos administrativos:**

- Contabilidad. Control de caja.
- Procesamiento de pedidos. Facturación.
- Control de proveedores y clientes.
- Control de almacén.
- Control de producción y de productividad.
- Gestión de personal. Nóminas.
- Planificación y control de proyectos grandes y complejos.
- Programación lineal: búsqueda de soluciones óptimas; por ejemplo, minimización de costes de transporte.
- Investigación y prospección de mercado.
- Modelos financieros y para predicción (de bolsa, por ejemplo).
- Gestión bancaria (control de clientes, gestión de cajeros automáticos, etc.).
- Sistemas de gestión de terminales punto de ventas (cajas de abono en almacenes comerciales).
- Gestión bibliotecaria. Archivos automatizados de bibliotecas. Control de préstamos, localización de libros, etc. (ver también “Documentación científica y técnica” en el apartado “Ciencias sociales y del comportamiento”).
- Seguros (evaluación de riesgos, etc.).
- Sistema de reserva y expedición de billetes (compañías de transporte aéreas, ferrocarriles, etc.).

**2. Ciencias físicas e ingeniería:**

- Resolución de ecuaciones y de problemas matemáticos, en general (cálculo numérico o simbólico).
- Análisis de datos experimentales utilizando técnicas estadísticas.
- Simulación y evaluación de modelos (por ejemplo, los utilizados en predicción meteorológica).
- Realización de tablas matemáticas.

**3. Ciencias de la vida y médicas:**

- Investigación médica, biológica y farmacéutica (base de datos sobre el DNA, análisis de datos de experimentos sobre efectos de medicamentos, etc.).
- Ayuda al diagnóstico (sistemas expertos para diagnóstico médico, por ejemplo).
- Bases de datos con historias clínicas.
- Medicina preventiva (control de vacunación de niños, por ejemplo).
- Electromedicina (ver apartado de “Ingeniería con ayuda de computador”).

**4. Ciencias sociales y del comportamiento:**

- Análisis de datos (evaluación de encuestas, por ejemplo).
- Bases de datos jurídicas (incluyendo legislación, jurisprudencia, etc.).
- Aplicaciones en educación: **CAI** (*Computer Assisted Instruction*) o **CAL** (*Computer Aided Learning*).
- Evaluación automática de exámenes.
- Juegos con computador (videojuegos, ajedrez, etc.).
- Documentación científica y técnica.

**5. Arte y humanidades:**

- Composición de cuadros.
- Composición musical.
- Elaboración de publicaciones tales como libros, periódicos y revistas.
- Realización de escenas animadas para películas de cine, televisión, etc.
- Análisis automático de textos (determinación de frecuencias de uso de palabras, etc.).

**6. Ingeniería con ayuda de computador:**

- Diseño, fabricación y test con ayuda de computador: **CAD** (*Computer Aided Design*), **CAM** (*Computer Aided Manufacturing*) y **CADMAT** (*Computer Aided Design Manufacturing and Testing*).
- Cartografía.
- Minería.
- Informática industrial (control con computador).

**7. Computadores en otros campos o sistemas.****Tabla 1.1.** Campos de aplicación verticales de la informática.



Aquellas actividades que requieran o presenten alguna de las características anteriores, son candidatas a ser efectuadas con ayuda de computador.

En la Tabla 1.1 de la página anterior se relacionan algunos ejemplos de **aplicaciones verticales** de la informática. Damos el calificativo de *vertical* porque la clasificación se hace en base a actividades profesionales o de un determinado sector. Por otra parte, la Tabla 1.2 recoge aplicaciones que son de utilidad en muy diversos campos, y que denominamos **aplicaciones horizontales**.

**Internet:**

- Correo electrónico (e-mail).
- Boletines de noticias (news).
- Acceso remoto a otro computador (telnet).
- Guías para búsquedas de información (gopher).
- Charlas interactivas de usuarios en red (Chats).
- Web.
- Comercio electrónico (e-commerce).
- Enseñanza a través de Internet (e-learning), etc.

**Ofimática:**

- Procesador de textos.
- Hoja electrónica.
- Gestión de archivos o/y bases de datos.
- Programas para presentaciones.

**Inteligencia artificial:**

- Sistemas expertos.
- Visión por computador, etc.

**Informática gráfica.**

**Aplicaciones multimedia**, etc.

**Tabla 1.2.** Campos de aplicación horizontales de la informática.

## 1.7. Conclusión

En este capítulo se han introducido un conjunto de definiciones y conceptos básicos, los cuales se ampliarán a lo largo de todo el texto. También se ha tratado de dar una visión panorámica de los distintos aspectos que abarca la informática.

Consideramos, además, que se ha introducido la terminología elemental para abordar adecuadamente los distintos capítulos.

### Test



**T1.1.** En la informática la información hace referencia:

- a) Al conjunto de datos numéricos que procesa un computador.
- b) Al conjunto de programas y datos numéricos que procesa un computador.
- c) A la yuxtaposición de símbolos que representan una instrucción, orden, hechos, objetos o ideas, susceptibles de ser introducidos en un computador.
- d) Al conjunto de programas que procesa el computador.

**T1.2.** Un computador es una máquina concebida para:

- a) Codificar y almacenar información binaria.
- b) Efectuar operaciones aritméticas y lógicas bajo el control directo del usuario.
- c) Efectuar operaciones aritméticas y lógicas bajo el control de un programa de instrucciones.
- d) Codificar y digitalizar la información.



**T1.3.** Un microprocesador es:

- a) Un computador muy pequeño.
- b) Una CPU integrada en un chip.
- c) Un procesador ubicado en una placa madre.
- d) Un circuito integrado que contiene en su interior parte de la memoria caché.

**T1.4.** Una memoria de 8 Mpalabras, con palabras de 32 bits tiene:

- a) 32 MBytes.
- b) 8 MBytes.
- c) 16 MBytes.
- d) Ninguna de las contestaciones anteriores es correcta.

**T1.5.** 64 MBytes es equivalente a:

- a) 512 Kbits.
- b)  $256 \cdot 10^{10}$  bits.
- c) 512 Mbits.
- d) Ninguna de las contestaciones anteriores es correcta.

**T1.6.** Una unidad de disco con 16 EBytes de memoria tiene:

- a)  $16 \cdot 2^{50}$  Bytes.
- b)  $16 \cdot 2^{30}$  GBytes.
- c)  $16 \cdot 2^{30}$  TBytes.
- d) Ninguna de las contestaciones anteriores es correcta.

**T1.7.** 1 Kilobit *exactamente* es:

- a)  $2^{10}$  bits.
- b)  $10^3$  bits.
- c)  $10^{10}$  bits
- d)  $2^3$  bits.

**T1.8.** Las unidades centrales de una computadora están formadas por:

- a) Procesador, memoria principal y memoria masiva (disco duro).
- b) Unidad de control, ALU y memoria principal.
- c) Unidad de control y memoria principal.
- d) Unidad de control, memoria principal y disco duro.

**T1.9.** El procesador (CPU) de una computadora está formado por:

- a) La unidad de control.
- b) Unidad de control y la zona ROM de la memoria principal.
- c) Unidad de control y ALU.
- d) Unidad de control, ALU y memoria principal.

**T1.10.** La memoria ROM de un computador es una parte de la memoria principal:

- a) En la que, como los discos, la información no se pierde al desconectar el computador.
- b) A la que, a diferencia de una memoria RAM, no se puede acceder de forma aleatoria.
- c) En la que se graban programas del sistema operativo en el momento de arrancar el computador, y en la que posteriormente no puede escribir el usuario.
- d) A la que se accede sólo por bytes, incrementándose así su velocidad.

**T1.11.** El tiempo de ciclo de un computador cuyo reloj es de 500 MHz es:

- a) 500 microsegundos.
- b) 2 microsegundos.
- c) 2 milésimas de microsegundo.
- d) Ninguna de las contestaciones anteriores es correcta.

**T1.12.** El *ancho de un bus*:

- a) Es la longitud (medida en pulgadas o centímetros) transversal de la banda donde van embebidos los hilos conductores del bus.
- b) Representa la cantidad de información que se transfiere a través de él, dada usualmente en bytes/segundo.
- c) Es la longitud (medida en pulgadas o centímetros) total de la banda donde van embebidos los hilos conductores del bus, medida entre las unidades más lejanas que interconecta.
- d) Es el número de bits que transmite simultáneamente, en paralelo.

**T1.13.** El *ancho de banda de un bus*:

- a) Es la longitud (medida en pulgadas o centímetros) transversal de la banda donde van embebidos los hilos conductores del bus.
- b) Representa la cantidad de información que se transfiere a través de él, dada usualmente en bytes/segundo.
- c) Es el número de bits que transmite simultáneamente, en paralelo.
- d) Es la longitud (medida en pulgadas o centímetros) total de la banda donde van embebidos los hilos conductores del bus, medida entre las unidades más lejanas que interconecta.

**T1.14.** Podemos comparar el rendimientos de dos procesadores distintos en la ejecución de un programa determinado:

- a) Simplemente comparando sus frecuencias de reloj.
- b) Comparando el número de ciclos consumidos por el programa, en cada uno de los procesadores.
- c) Comparando el número que resulta de multiplicar el tiempo de ciclo por el número de ciclos consumidos por el programa, en cada uno de los procesadores.
- d) Debe utilizarse un *benchmark*.

**T1.15.** El *rendimiento* de un computador en la ejecución de un programa determinado es:

- a) El tiempo de ejecución de dicho programa.
- b) La inversa del tiempo de ejecución de dicho programa.
- c) Es el cociente entre resultados de salida y datos de entrada, expresado en tanto por ciento.
- d) El porcentaje total de resultados correctos que se obtiene ejecutando el programa con distintos datos de entrada.

**T1.16.** Para comparar la velocidad de dos computadores, se debe:

- a) Comparar las frecuencias de reloj, aunque los procesadores sean distintos; la máquina con mayor frecuencia es la más rápida.
- b) Hacer una media del número de ciclos que consumen las distintas instrucciones máquina del repertorio, y multipli-

carla por el período del reloj; la máquina con menor resultado será la más rápida.

- c) Comparar el número de instrucciones máquina; la máquina con menos instrucciones será la más rápida.
- d) Ejecutar un conjunto de programas de prueba (*benchmark*); la máquina con mayor valor de MIPS o MFLOPS será la más rápida.

**T1.17.** MIPS son las iniciales de:

- a) Millones de instrucciones por segundo.
- b) Memorias principal y secundaria.
- c) Memoria interna parcialmente secundaria.
- d) Múltiples interfaces para programas secundarios.

**T1.18.** MFLOPS son las iniciales en inglés de:

- a) MBytes en disquetes (*floppy discs*).
- b) Millones de instrucciones de coma flotante por segundo.
- c) Memoria en disquetes (*floppy discs*) y otros dispositivos de memoria secundaria.
- d) Megabytes ocupados por el sistema operativo.

**T1.19.**Cuál de las siguientes afirmaciones es correcta:

- a) En algunos computadores un programa puede ejecutarse sin necesidad de cargarlo en la memoria principal.
- b) Un programa, para que se ejecute, debe estar cargado en la memoria principal.
- c) Un programa, para que se ejecute, basta con que esté en el disco duro.
- d) Un programa, para que se ejecute, si está en lenguaje máquina, puede estar en cualquier unidad.

**T1.20.** Un computador:

- a) Puede tener más de un procesador central.
- b) Puede tener distintos tipos de periféricos pero no varios procesadores centrales.
- c) Puede disponer de varios procesadores centrales siempre que esté conectado a Internet.
- d) Pueden tener más de un procesador central, siempre que sea una estación de trabajo.

**T1.21.** Una estación de trabajo (*workstation*)

- a) Es un terminal remoto para introducción de trabajos en un computador (y obtención de resultados), a través de una red.
- b) Es un equipo utilizado en los grandes almacenes para el cobro de los productos, y que está conectado a una red

de computadores, tiene detector de barras impresas, etc. (a veces se le denomina “terminal punto de ventas”).

- c) Es, en general, un computador con velocidad, capacidad de memoria y prestaciones gráficas mejores que un PC, pero proyectado para ser usado por un único usuario.
- d) Es un equipo evolucionado del PC, para poder ser utilizado simultáneamente por varios usuarios.

**T1.22.** El lenguaje de control de un computador:

- a) Es el lenguaje asociado a la arquitectura del computador (depende del procesador central).
- b) Es un lenguaje para hacer uso de las posibilidades de su sistema operativo.
- c) Es el conjunto de términos técnicos que hacen referencia a la estructura y organización del computador.
- d) Es el conjunto de señales generadas por la unidad de control para monitorizar el funcionamiento de las distintas unidades del computador.

**T1.23.** En el contexto de la informática, CAD son las siglas inglesas de:

- a) Desarrollo y fabricación con ayuda de computador.
- b) Diseño con ayuda de computador.
- c) Visualización por computadora.
- d) Realización de gráficos e imágenes con ayuda de computadora.

**T1.24.** En el contexto de la informática, CAM son las siglas inglesas de:

- a) Fabricación con ayuda de computador.
- b) Diseño con ayuda de computador.
- c) Visualización por computadora.
- d) Realización de gráficos e imágenes con ayuda de computadora.

**T1.25.** Las aplicaciones multimedia tratan de:

- a) Ayudar a la producción en los medios de comunicación (edición de periódicos y revistas, programas de televisión, edición de programas radiofónicos, etc.).
- b) Poder transmitir información por redes heterogéneas (a través de medios distintos: cable, fibra óptica, atmósfera, vía satélite, etc.).
- c) Posibilitar grabar la información en distintos medios o soportes de información (discos magnéticos, discos ópticos, cintas magnéticas, etc.).
- d) Producir presentaciones con computador que combinan texto, gráficos, imágenes, sonido, etc.

## Problemas resueltos



### CONCEPTOS BÁSICOS. CODIFICACIÓN

- P1.1.** Codificar las letras del alfabeto utilizando caracteres numéricos. Escribir a continuación, con el código anterior, la frase “En un lugar de la Mancha...”.

## SOLUCIÓN

Un posible código puede ser el siguiente:

a	01
b	02
c	03
d	04
e	05
f	06
g	07
h	08
i	09
j	10
k	11
l	12
m	13
n	14
ñ	15
o	16
p	17
q	18
r	19

s	20
t	21
u	22
v	23
x	24
y	25
z	26
Espacio	00
“	27
.	28
Tilde (‘)	29
-	30
A	33
B	34
C	35
D	36
E	37
F	38
G	39

H	40
I	41
J	42
K	43
L	44
M	45
N	46
Ñ	47
O	48
P	49
Q	50
R	51
S	52
T	53
U	54
V	55
X	56
Y	57
Z	58

La codificación de la frase será así:

“En un lugar de la Mancha...”

27, 37, 14, 00, 22, 14, 00, 12, 22, 07, 01, 19, 00, 04, 05, 00, 12, 01, 00, 45, 01, 14, 03, 08, 01, 28, 28, 28, 27

(Separamos con una coma “,” cada carácter para comprobar con comodidad el código generado.)

## CONCEPTOS BÁSICOS: MEDIDAS DE CAPACIDAD

**P1.2.** ¿Cuántos bits hay en 32 KB?

## SOLUCIÓN

Suponemos que 1 byte = 8 bits.

$$32 \text{ KB} = 32 \cdot 1.024 \text{ B} \cdot 8 \text{ b/B} = 262.144 \text{ bits}$$

¿Y en 64 MB?

## SOLUCIÓN

$$64 \text{ MB} = 64 \cdot 2^{20} \text{ B} \cdot 8 \text{ b/B} = 536.870.912 \text{ bits}$$

¿Y en 4 GB?

## SOLUCIÓN

$$4 \text{ GB} = 4 \cdot 2^{30} \text{ B} \cdot 8 \text{ b/B} = 3.435.973.837 \cdot 10^{10} \text{ bits.}$$

**P1.3.** Un computador tiene 36 Kpalabras de memoria principal, y está estructurada en palabras de 32 bits. ¿Cuántos caracteres caben en dicha memoria?

## SOLUCIÓN

Suponemos que cada carácter ocupa 8 bits.

$$36 \text{ Mpalabras} = 36 \cdot 1.024 \cdot 1.024 \cdot 32 \text{ bits} = 1.207.959.552 \text{ bits}$$

$$N.^{\circ} \text{ caracteres} = (1.207.959.552 \text{ b}) / (8 \text{ b/car}) = 150.994.944 \text{ caracteres}$$

**P1.4.** Suponiendo que un computador cuya capacidad máxima de memoria es de 32 Mbytes, y está organizado en palabras de 32 bits:

- ¿Cuántos bits tienen en total?
- ¿Cuál es el ancho (número de hilos) de los buses de datos y de direcciones?
- Suponiendo que el ancho de banda de la memoria es de 200 MBytes/s, indicar la velocidad de transferencia que debe admitir cada hilo del bus de datos.

## SOLUCIÓN

$$a) \quad 32 \text{ MBytes} = \frac{32 \text{ Mbytes}}{4 \text{ bytes/palabra}} = 8 \text{ Mpalabras}$$

Como internamente, en los chips, cada palabra tiene: 32 bits + 1 bit (de paridad) = 33 bits, internamente habrá:

$$C_{\text{bits}} = 8 \text{ Mpalabras} \cdot 33 \text{ bits/palabra} = 8 \cdot 33 \text{ Mbits} = 264 \text{ Mbits} = 264 \cdot 2^{20} \text{ bits} = 276.824.064 \text{ bits}$$

- Como la longitud de la palabra es de 32 bits:  $L_{\text{datos}} = 32 \text{ bits}$  (o hilos).

Como hay que direccionar 8 Mpalabras =  $8 \cdot 2^{20} \text{ palabras} = 2^{23} \text{ palabras} \Rightarrow L_{\text{direcciones}} = 23 \text{ bits}$  (o hilos).

$$c) \quad 200 \text{ MBytes/s} = 200 \cdot 8 \text{ Mbits/s} = 1.600 \text{ Mbits/s}$$

Como la transferencia se realiza simultáneamente (en paralelo) a través del bus de datos, que tiene 32 hilos, la velocidad de transferencia por cada hilo será:

$$v_{\text{hilo}} = \frac{1.600 \text{ Mbits/s}}{32 \text{ hilos}} = 50 \text{ Mbits/(s} \cdot \text{hilo)} = 52.428.800 \text{ bits/(s} \cdot \text{hilo)}$$

## MEDIDAS DE PRESTACIONES

**P1.5.** Se dispone de una memoria de 4 GBytes, organizada en palabras de 32 bits. Suponiendo que cada hilo transmite a una velocidad de 150 Mbits/segundo, obtener:

- La anchura de los buses de datos y de dirección de la memoria.
- El tiempo necesario para transferir 1 MByte entre la memoria y el procesador.
- ¿Cuál debe ser la frecuencia de reloj mínima para poder leer una palabra de memoria en un ciclo?

## SOLUCIÓN

$$a) \quad 1 \text{ palabra} = 32 \text{ bits} = \frac{32 \text{ bits}}{8 \text{ bits/Byte}} = 4 \text{ Bytes}$$

$$4 \text{ GBytes} = \frac{4 \text{ GB}}{4 \text{ B/p}} = 1 \text{ Gp} = 2^{30} \text{ palabras}$$

$$A_{\text{direcciones}} = 30 \text{ hilos}$$

Como la palabra es de 32 bits:

$$A_{\text{datos}} = 32 \text{ hilos}$$

- Como hay 32 hilos y cada hilo transmite a 150 Mb/s, en 1 segundo se transmitirán:

$$150 \text{ Mp/s} = 150 \cdot 4 \text{ MB/s} = 600 \text{ MB/s}$$

1 MB tardará en transmitirse:

$$\frac{1 \text{ MB}}{600 \text{ MB/s}} = 1,66 \text{ ns}$$

c) Como en el bus de datos se transmite a una velocidad de 150 Mp/s, se requerirá un CPU de:

$$150 \cdot 2^{20} \text{ Hz} = 157,3 \cdot 10^6 \text{ Hz} = 157,3 \text{ MHz}$$

ya que  $1 \text{ MHz} = 10^6 \text{ Hz}$ .

**P1.6.** Supóngase que se tiene un programa en lenguaje máquina con las siguientes instrucciones:

```
LLI r0,00
LLI r1,04
ADDS rE,r1,r0
```

Este programa carga en el registro del procesador r0 el valor 0, en el registro r1 el valor 4, y suma los valores de r0 y r1, llevando el resultado al registro rE. El programa se ejecuta por dos procesadores distintos, A y B, cuyas frecuencias de reloj y número de ciclos que utilizan para ejecutar las instrucciones del programa son las que se dan en la siguiente tabla:

	Procesador A	Procesador B
Frecuencia de reloj	1 GHz	800 MHz
N.º de ciclos de instrucción LLI	6	3
N.º de ciclos de instrucción ADDS	7	4

- Obtener los tiempos de ejecución del programa en los procesadores A y B.
- Obtener los rendimientos (en MIPS) de cada uno de los procesadores para el programa.
- Discutir la influencia de la frecuencia de reloj y del número de ciclos por cada instrucción en los resultados anteriores.

#### SOLUCIÓN

- El tiempo de ejecución de un programa ( $t$ ) se puede obtener sumando lo que tarda la ejecución de cada una de sus instrucciones. Por otra parte, la duración de cada instrucción  $i$  es el número de sus ciclos ( $n_{ci}$ ) multiplicado por el período de reloj ( $T$ ); es decir:

$$t_A = \sum n_{ci} \cdot T = \frac{\sum n_{ci}}{F}$$

donde se ha tenido en cuenta que el período del reloj es el inverso de la frecuencia:  $T = 1/F$ .

El número de ciclos del programa en cada procesador será el que se da en la siguiente tabla:

	Procesador A	Procesador B
LLI r0,00	6	3
LLI r1,04	6	3
ADDS rE,r1,r0	7	4
N.º total de ciclos	19	10

Con lo que el tiempo de ejecución del programa en el procesador A ( $t_A$ ) y el B ( $t_B$ ) serán, respectivamente:

$$t_A = \frac{19}{10^9 \text{ c/s}} = 19 \text{ ns}$$

$$t_B = \frac{10}{8 \cdot 10^8 \text{ c/s}} = 12,5 \text{ ns}$$

- b) El procesador A tarda 19 ns en ejecutar las tres instrucciones del programa, por lo que, por término medio, en un segundo ejecutaría las siguientes instrucciones:

$$\text{rendimiento} = \frac{3 \text{ instrucciones}}{19 \cdot 10^{-9} \text{ s}} = 158 \cdot 10^6 \text{ instrucciones/s} = 158 \text{ MIPS}$$

En el caso del procesador B se tiene:

$$\text{rendimiento} = \frac{3 \text{ instrucciones}}{12,5 \cdot 10^{-9} \text{ s}} = 240 \cdot 10^6 \text{ instrucciones/s} = 240 \text{ MIPS}$$

- c) Se observa que el rendimiento es mejor en el procesador B que en el A (la ejecución del programa tarda menos en el primero). Obsérvese que, a pesar de que la frecuencia de reloj es mayor en el procesador A, su rendimiento es menor que el B. La expresión deducida en el apartado a) del problema pone claramente de manifiesto que los tiempos de ejecución serán menores cuanto mayores sean las frecuencias y menor el número de ciclos total del programa. El número de ciclos de un programa, a su vez, depende del número de instrucciones del programa y el número de ciclos de cada una de las instrucciones; en definitiva de la arquitectura del procesador.

Por otra parte hay que señalar que el rendimiento calculado ha sido para un programa en concreto. Para obtener resultados rigurosos sobre el rendimiento del procesador habría que utilizar conjuntos de programas de prueba normalizados (*benchmarks*).

- P1.7.** Se dispone de un programa que contiene un total de 200 instrucciones máquina, 150 de ellas pertenecientes a un bucle que se ejecuta 50 veces. La ejecución del programa consume un total de  $N_{TC} = 34.000$  ciclos de reloj:

- a) ¿Cuál es el número medio de ciclos por instrucción,  $N_{CI}$ ?  
 b) ¿Cuál es la frecuencia del reloj del procesador si tarda  $34 \mu\text{s}$  en ejecutarse el programa?

SOLUCIÓN

- a) El número total de instrucciones a ejecutar es:

$$N_I = 50 + 150 \cdot 50 = 7.550 \text{ instrucciones}$$

Entonces, el número medio de ciclos por instrucción será:

$$N_{CI} = \frac{N_{TC}}{N_I} = \frac{34.000}{7.550} = 4,5 \text{ ciclos/instrucción}$$

- b) De la expresión [1.2] se tiene que:

$$F = \frac{N_{TC}}{T_E} = \frac{34.000}{34 \cdot 10^{-6}} = 10^9 \text{ Hz} = 1 \text{ GHz}$$

- P1.8.** Un programa de prueba (en lenguaje de alto nivel) se ejecuta en un procesador obteniéndose una velocidad de 100 MIPS y tardando su ejecución 35 segundos. Obtener la frecuencia de reloj sabiendo que por término medio cada instrucción consume 6 ciclos de reloj.

SOLUCIÓN

La frecuencia de reloj es el número de ciclos que se ejecutan por segundo; con lo que, como sabemos, el número de instrucciones que se ejecutan por segundo (MIPS) se tendrá que:

$$F = 100 \cdot 10^6 \frac{\text{instrucción}}{\text{s}} \cdot 6 \frac{\text{ciclos}}{\text{instrucción}} = 600 \cdot 10^6 \frac{\text{ciclos}}{\text{s}} = 600 \text{ MHz}$$



## Problemas propuestos

### CONCEPTOS BÁSICOS. CODIFICACIÓN

**P1.9.** Suponiendo que las calificaciones que se dan en un determinado centro son Suspenso, Aprobado, Notable, Sobresaliente y Matrícula de Honor, ¿cuántos bits se necesitarán para codificar la calificación?

### MEDIDAS DE PRESTACIONES

**P1.10.** Suponga que un sistema se ha diseñado para admitir una capacidad máxima de memoria principal de 16 MBytes y su bus de E/S de datos es de 32 bits. Asumiendo que tanto el bus de datos como el de direcciones puede transmitir a una velocidad máxima de 800 MBytes/segundo, obtener:

- La anchura del bus de dirección de la memoria.
- La velocidad máxima alcanzable en cada línea del bus de datos.
- La velocidad máxima alcanzable por cada línea del bus de direcciones.
- El tiempo necesario para transferir 1 MByte entre la memoria y el procesador.
- ¿Cuál debe ser la frecuencia de reloj mínima para poder leer una palabra de memoria en un ciclo?

**P1.11.** Suponga que las instrucciones ADD, MUL, LOAD y STORE de un computador, cuyo reloj funciona a una frecuencia de 500 MHz, consumen 1, 10, 6 y 6 ciclos de reloj, respectivamente; obtener el tiempo que tardan en ejecutarse dichas instrucciones.

**P1.12.** En un catálogo se indica que la velocidad de un computador es de 2 MIPS. Estimar el tiempo aproximado que se tardaría en ejecutar un conjunto de 328.325 instrucciones. ¿Qué inexactitudes presenta dar la velocidad en MIPS?

**P1.13.** Se dispone de un programa que contiene un total de 500 instrucciones máquina, de las que 35 se encuentran en un bucle que se tiene que ejecutar 325 veces, y 102 en otro bucle que se ejecuta 76 veces. Sabiendo que el número medio de ciclos por instrucción se ha estimado en 5,3 y que la frecuencia del reloj es de 1,2 GHz:

- Obtener el rendimiento del procesador en la ejecución de ese programa.
- Si la arquitectura del procesador mejorase de forma tal que, para el mismo repertorio de instrucciones máquina el número medio de ciclos por instrucción se redujese en un 25 por 100, y la velocidad del procesador aumentase en un 6 por 100, ¿qué variación en el tiempo de ejecución del programa se obtendría?

**P1.14.** Un programa de prueba (en lenguaje de alto nivel) se ejecuta en dos procesadores (A y B) y se observa que en el A se obtienen 58 MFLOPS y en el B se logran 75 MFLOPS; sin embargo, el tiempo de ejecución de ese mismo programa es de

85 ms en el procesador A y 105 en el procesador B. ¿Es esto posible? Justifique su respuesta.

### TIPOS Y CARACTERÍSTICAS DE COMPUTADORES

**P1.15.** Consiga información de al menos cuatro computadores distintos, realice una tabla comparativa de los mismos incluyendo:

- Marca.
- Modelo.
- Microprocesador que utiliza.
- Longitud de palabra.
- Velocidad (frecuencia de reloj).
- Memoria principal, inicial y máxima.
- Periféricos de que dispone.
- Nombre del sistema operativo.
- Precio.
- Otros parámetros que considere de interés.

Realice un pequeño informe comparativo, e indique el tipo de computador de que se trata.

**P1.16.** Encuentre un ejemplo concreto y actual de cada uno de los tipos de computadores siguientes:

- Supercomputador.
- Macrocomputador.
- Servidor de red.
- Estación de trabajo.
- Computador personal.
- Computador móvil.

Indique las características más destacadas de cada uno de ellos. (*Sugerencia: hágalo utilizando Internet.*)

**P1.17.** Obtener a través de la dirección web de los computadores más potentes de la actualidad (TOP500) las siguientes características del computador más potente:

- Número de procesadores.
- Modelo de procesador.
- Capacidad de memoria.
- Capacidad de disco.
- Precio.
- Institución propietaria del computador.
- Fabricante.

### PROGRAMAS E INSTRUCCIONES

**P1.18.** Suponiendo un lenguaje máquina que tuviese 16 tipos de instrucciones, ¿cuántos programas diferentes de 10 instrucciones de distinto tipo podrían realizarse?

**P1.19.** Suponiendo que en un lenguaje máquina todas las instrucciones fuesen de 16 bits, ¿cuántas instrucciones distintas se podrían formar con dicho lenguaje?

### NIVELES DE DESCRIPCIÓN DE UN COMPUTADOR

**P1.20.** Indicar a qué niveles conceptuales de descripción de una computadora pertenecen los siguientes elementos:

- a)* La descripción del trazado del interior de un circuito integrado que contiene una ALU.
- b)* Un compilador de C.
- c)* Un programa en lenguaje máquina.
- d)* El diseño de un circuito para multiplicar utilizando puertas lógicas.
- e)* El controlador de un ratón, incluido en un DVD.





# Representación de la información en los computadores

Según se indicó en el capítulo anterior un computador es un sistema para procesar de forma automática la información. Resulta obvio, por tanto, que una primera cuestión que debemos estudiar es la forma de representar esa información en el computador, siendo éste el objetivo del presente capítulo.

En la actualidad las formas de información más relevantes que se utilizan son: textos, sonidos, imágenes y datos numéricos, y cada una de ellas presenta peculiaridades muy distintas. En definitiva, este capítulo pretende presentar los procesos que transforman la información externa al computador en patrones de bits fácilmente almacenables y procesables por los elementos internos del mismo.

El capítulo se completa con dos temas íntimamente relacionados con la representación de la información como son los métodos de detección de errores que se puedan producir en la transmisión o almacenamiento de la información, y las técnicas más básicas para compresión con objeto de que la información ocupe menos espacio en los dispositivos de almacenamiento y sea más rápida su transmisión.

Diversas cuestiones tratadas en este y en posteriores capítulos se basan en los sistemas de numeración aritméticos, por lo que para tener una referencia próxima a ellos se incluye el Apéndice 1, que recomendamos estudiar al lector no familiarizado con este tema antes de iniciar el presente capítulo.

## 2.1. Representación de textos

Podemos representar cualquier información escrita (texto) por medio de caracteres. Los caracteres que se utilizan en informática suelen agruparse en cinco categorías:

1. **Caracteres alfabéticos:** Son las letras mayúsculas y minúsculas del abecedario inglés:

*A, B, C, D, E,..., X, Y, Z, a, b, c, d, ..., x, y, z*

2. **Caracteres numéricos:** Están constituidos por las diez cifras decimales:

*0, 1, 2, 3, 4, 5, 6, 7, 8, 9*

3. **Caracteres especiales:** Son símbolos ortográficos y matemáticos no incluidos en los grupos anteriores, entre otros los siguientes:

) ( , \* / ; : + Ñ ñ = ! ? . " & > # < ] Ç [ SP

Con SP representamos el carácter o espacio en blanco, tal como el que separa dos palabras.

4. **Caracteres geométricos y gráficos:** Son símbolos o módulos con los que se pueden representar cuadros, formas geométricas o iconos elementales, como por ejemplo:

| ¬ ¯ ♠ ♣ ♥ ♦ ▒ ▓ ▔ ▕ ▖ ▗ ▘ ▙

5. **Caracteres de control:** Representan órdenes de control, como el carácter para pasar a la línea siguiente (*NL*), o para ir al comienzo de una línea (*CR*) o para sincronización de una transmisión (*SYN*) o para que se emita un pitido en un terminal (*BEL*), etc. Por ejemplo, cuando en un teclado, al estar creando un texto, pulsamos la tecla de nueva línea, automáticamente se insertan los caracteres de control *CR* y *NL*. Cuando en una impresora se reciben esos caracteres se generan las señales de control que provocan las acciones requeridas: saltar al comienzo (*CR*) de la línea siguiente (*NL*).

Al introducir un texto en un computador, a través del periférico correspondiente, los caracteres se codifican según un **código de entrada/salida** de forma que a cada carácter se le asocia una determinada combinación de  $n$  bits. Un **código de E/S** es sencillamente una correspondencia entre los conjuntos:

$$\alpha \equiv \{0, 1, 2, \dots, 9, A, B, \dots, Z, a, b, \dots, z, *, +, \%, \dots\} \rightarrow \beta \equiv \{0, 1\}^n \quad [2.1]$$

Los elementos del conjunto  $\alpha$  de caracteres, así como su número  $m$ , dependen del código de E/S utilizado por el programa de edición del texto y por el periférico que codifique o decodifique dicho texto.

Estamos suponiendo que utilizamos un número fijo,  $n$ , de bits para codificar los  $m$  símbolos de  $\alpha$ . El número mínimo de bits necesarios para codificar un conjunto de símbolos depende del cardinal de este conjunto, verificándose:

$$2^{n-1} < m \leq 2^n \quad \text{o} \quad n \geq \log_2 m = 3,32 \cdot \log(m), \quad \text{con } n \in \mathbb{N} \quad [2.2]$$

Obviamente  $n$  debe ser un número natural (entero positivo).

Los códigos más utilizados en la actualidad son el EBCDIC, ASCII y Unicode, por lo que a continuación pasamos a describirlos brevemente:

- **Código EBCDIC** (*Extended Binary Coded Decimal Interchange Code*)

Este código utiliza  $n = 8$  bits, de forma que permite codificar hasta  $m = 2^8 = 256$  símbolos distintos. En la Tabla A2.1 del Apéndice 2 se incluye el código correspondiente a cada carácter. Sumando el valor de la primera fila con el de la primera columna correspondientes a un carácter dado se obtiene el código en hexadecimal de dicho carácter, y sumando el de la segunda fila con el de la segunda columna se obtiene el código en decimal.

- **Código ASCII** (*American Standard Code for Information Interchange*)

El código ASCII básico utiliza 7 bits y hoy día es de los más usados. Se puede decir que la mayor parte de las transmisiones de datos entre dispositivos se realizan en esta codificación, y corresponde a la normalización ANSI X3.4-1968 o ISO 646. En la Tabla A2.2 del Apéndice 2 se incluye el código correspondiente a cada carácter. Sumando el valor de la primera fila con el de la primera columna corres-

pondientes a un carácter dado se obtiene el código en hexadecimal de dicho carácter, y sumando el de la segunda fila con el de la segunda columna, se obtiene el código en decimal. Usualmente se incluye un octavo bit para detectar posibles errores de transmisión o grabación (bit de paridad, Sección 2.5).

Existen numerosas versiones ampliadas del código ASCII básico, que utilizan 8 bits, aprovechando las combinaciones no usadas para representar símbolos adicionales. Entre ellas se encuentran los **códigos ISO 8859-n**, donde *n* es un número que identifica el juego de los nuevos caracteres introducidos, dependiendo de los lenguajes (Tabla 2.1). Por ejemplo, la norma **ISO 8859-1**, también denominada **ISO Latín1** (Tabla A2.3), se proyectó para América y Europa occidental, e incluye vocales con acentos, tildes, con signos diacríticos (esto es, signos ortográficos que dan a una letra un valor especial, como la diéresis usada en alemán y en español), y otras letras latinas no usadas en los países anglosajones. Corresponde a la *página de códigos* 819 de los PC. Según puede observarse en Tabla A2.3, en ISO Latín-1 quedan combinaciones libres, que pueden usarse para codificar otros caracteres obteniéndose así nuevos códigos no normalizados, pero compatibles con el ISO Latín1, como es la página de códigos 850 de los PC.

Denominación	Estándar	Área geográfica
Latín-1	ISO 8859-1	Oeste y Europa del Este.
Latín-2	ISO 8859-2	Europa Central y del Este.
Latín-3	ISO 8859-3	Europa Sur, maltés y esperanto.
Latín-4	ISO 8859-4	Europa Norte.
Alfabeto latín/cirílico	ISO 8859-5	Lenguajes eslavos.
Alfabeto latín/árabe	ISO 8859-6	Lenguajes arábigos.
Alfabeto latín/griego	ISO 8859-7	Griego moderno.
Alfabeto latín/hebraico	ISO 8859-8	Hebreo y Yiddish.
Latín-5	ISO 8859-9	Turco.
Latín-6	ISO 8859-10	Nórdico (Sámi, Inuit e islandés).
Alfabeto latín/Thai	ISO 8859-11	Lenguaje Thai.
Latín-7	ISO 8859-13	Báltico <i>Rim</i> .
Latín-8	ISO 8859-14	Céltico.
Latín-9 ( <i>alias Latín-0</i> )	ISO 8859-15	Latín-1 con ligeras modificaciones (símbolo €).

**Tabla 2.1.** Familia de la normalización ISO 8859, ampliaciones de ASCII.

### • Código Unicode

Los códigos de E/S citados anteriormente son insuficientes para representar todos los símbolos escritos que necesitan las aplicaciones actuales. En particular, los lenguajes escritos de diversas culturas orientales, como la china, japonesa y coreana, se basan en la utilización de ideogramas o símbolos que representan palabras, frases o ideas completas, siendo, por tanto, inoperantes los códigos de texto convencionales, que sólo codifican letras individuales. **Unicode** es un código de E/S propuesto por un consorcio de empresas y entidades que trata de hacer posible escribir aplicaciones que sean capaces de procesar texto en diversos sistemas de escritura. Está reconocido como estándar **ISO/IEC 10646**, y trata de ofrecer las siguientes propiedades:

- *Universalidad*, ya que persigue cubrir la mayoría de lenguajes escritos existentes en la actualidad.
- *Unicidad*, ya que a cada carácter se le asigna exactamente un único código.
- *Uniformidad*, ya que todos los símbolos se representan con un número fijo de bits, concretamente 16. En total se pueden codificar  $2^{16} = 65.536$  símbolos.

La Tabla 2.2 muestra un esquema de cómo se han asignado los códigos Unicode. Puede observarse que a los códigos ASCII Latín-1 (ISO 8859-1) se les ha reservado las primeras 256 posiciones (de la H'0000 a la H'00FF).

Zona	Rango Unicode		Se corresponde con	N.º
A	0000	0000 a 007F 0080 a 00FF	Latín Básico (00 a 7F; ASCII ANSI-X3.4) Suplemento Latín-1 (ISO 8859-1)	256
		0100 a 017F 0180 a 024F 0250 a 02AF 02BF a 02FF 0300 a 036F 0370 a 03FF 0400 a 04FF 0530 a 058F 0590 a 05FF 0600 a 06FF 0700 a 074F Etc.	Ampliación A de Latín Ampliación B del Latín Ampliación del Alfabeto Fonético Internacional (IPA) Espaciado de letras modificadoras Combinación de marcas diacríticas (tilde, acento grave, etc.) Griego Cirílico Armenio Hebreo Árabe Sirio Etc.	7.936
	3FFF	2000 a 3FFF	Símbolos generales y caracteres fonéticos chinos, japoneses y coreanos	8.192
I	4000 9FFF	Ideogramas		24.576
O	A000 DFFF	Pendiente de asignación		16.384
R	E000 FFFF	Caracteres locales y propios de los usuarios. Compatibilidad con otros códigos		8.192

**Tabla 2.2.** Esquema de asignación de códigos en Unicode.

## 2.2. Representación de sonidos

Las **aplicaciones multimedia**, sobre todo debido al desarrollo de la web, han adquirido una gran importancia. Estas aplicaciones procesan tanto textos como sonidos e imágenes. En esta sección vamos a describir cómo se representan dentro de un computador los sonidos, y en la sección siguiente las imágenes.

Una señal de sonido se capta por medio de un micrófono que produce una **señal analógica**, esto es, una señal que puede tomar cualquier valor dentro de un determinado intervalo continuo. Posteriormente la señal analógica es amplificada para encajarla dentro de dos valores límites, por ejemplo entre  $-5$  voltios y  $+5$  voltios. En un intervalo de tiempo continuo se tienen infinitos valores de la señal analógica, por lo que para poder almacenarla y procesarla se realiza un proceso de **muestreo**. El muestreo selecciona valores de la señal analógica a una frecuencia  $F_s$  determinada; así cada  $T_s = 1/F_s$  segundos se dispone de un valor de la señal. Simultáneamente al muestreo, las muestras se digitalizan (se transforman a binario) con un **convertor analógico/digital** (Sección 6.9). En definitiva, la señal de sonido queda representada por una secuencia de valores, por ejemplo de 8 bits, correspondiendo cada uno de ellos a una muestra analógica. A partir de las muestras digitales se puede recuperar la señal. En el muestreo intervienen fundamentalmente dos parámetros:

- La *frecuencia de muestreo*,  $F_s$ , que debe ser igual o superior a un determinado valor, que depende de la calidad del sonido a recuperar; en otras palabras, dentro de un intervalo de tiempo dado deben tomarse suficientes muestras para no perder la forma de la señal original.
- El *número de bits* (precisión) con el que se representa cada muestra y que debe ser el adecuado.

Obviamente, cuanto mayores son la frecuencia de muestreo y el número de bits por muestra, mayor será el volumen de los archivos que almacenan el sonido; por lo que ambos parámetros deben elegirse en función de la calidad requerida. No siempre se necesita la misma calidad; así una conversación telefónica no tiene por qué ser de una fidelidad tan buena como la de una audición de música en un equipo de alta fidelidad. En la Tabla 2.3 se especifican parámetros usuales para obtener distintas calidades de sonido.

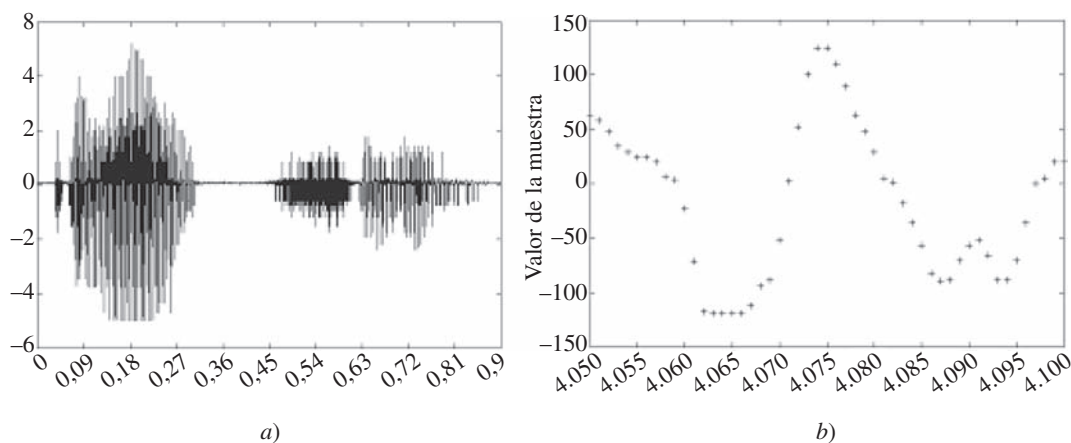
	N.º de bits/muestra	Frecuencia de muestreo ( $F_s$ , KHz)	Período de muestreo ( $T_s$ , $\mu$ segundos)
PCM teléfono	8	8	125
Calidad telefónica	8	11,025	90,7
Radio	8	22,05	45,4
CD	16 <sup>1</sup>	44,1	22,7

<sup>1</sup> Número de bits/muestra por canal; con sonido estereofónico hay que multiplicar por 2.

**Tabla 2.3.** Distintas calidades de señales de sonido.

### EJEMPLO 2.1

En la Figura 2.1a se muestra la señal producida por un micrófono al pronunciar la palabra “casa” que ocupa 0,9 segundos. Esta señal se ha muestreado a una frecuencia de  $F_s = 22,05$  KHz; de esta forma, en el intervalo de tiempo de 0 a 0,9 segundos, se obtienen 19.845 muestras. En la Figura 2.1b se representan las muestras 4.050 a 4.100, que corresponden al intervalo de tiempo que va de 0,184 a 0,186 segundos. La señal se digitaliza con 8 bits, y quedará, por lo tanto, almacenada en un archivo o tabla de unos 2 Kbytes. En el archivo, previamente a las muestras, se almacena una cabecera especificando el formato de los datos.



**Figura 2.1.** a) Señal analógica captada por un micrófono al pronunciar la palabra “casa”. b) Tramo de muestras comprendido entre 0,184 a 0,186 segundos.

## 2.3. Representación de imágenes

Las imágenes se adquieren por medio de periféricos especializados tales como escáneres, cámaras de video o cámaras fotográficas (Sección 6.8). Como todo tipo de información, una imagen se representa por patrones de bits, generados por el periférico correspondiente. Hay sistemas de codificación de imágenes muy diversos. En la Tabla 2.4 se describen algunos de ellos. Existen dos formas básicas de

representar las imágenes, conocidas como mapas de bits y mapas de vectores, respectivamente, y que a continuación se describen brevemente.

Tipo	Formato	Origen	Descripción
Mapa de bits	BMP ( <i>BitMap</i> )	Microsoft	Sencillo, imágenes de gran calidad pero ocupan mucho (no útil para web).
	TIFF ( <i>Tagged Image File Formats</i> )	Microsoft y Aldus	Prácticamente un estándar para imágenes fotográficas naturales de alta calidad. Ocupa mucho (no útil para web).
	JPEG ( <i>Joint Photographic Experts Group</i> )	Grupo JPEG	Calidad razonable para imágenes naturales. Incluye compresión. Usado en la web.
	GIF ( <i>Graphic Interchange Format</i> )	CompuServe	Muy adecuado para imágenes no naturales (colores planos (logotipos, banderas, dibujos animados...)). Muy usado en la web.
	PNG ( <i>Portable Network Graphics</i> )	Consortio www	Evolución mejorada de GIF. Muy buena calidad de colores. Incluye muy buena compresión.
Mapa de vectores	IGES ( <i>Initial Graphics Exchange Specification</i> )	ASME/ANSI	Estándar para intercambio de modelos y datos CAD (usable en AutoCAD, etc.).
	DXF ( <i>Document eXchange Format</i> )		Formato original del AutoCAD.
	PICT ( <i>PICTure</i> )	Apple Comp.	Imágenes vectoriales que pueden incluir objetos que son imágenes en mapa de bits.
	EPS ( <i>Encapsulated Postscript</i> )	Adobe Sys.	Ampliación para imágenes del lenguaje de impresión Poscript, con la que se pueden insertar imágenes en distintos formatos como TIFF, WMF, PICT o EPSI.
	TrueType	Apple Comp.	Alternativa de Apple y Microsoft para el EPS.

**Tabla 2.4.** Algunos formatos utilizados para representación de imágenes.

### 2.3.1. MAPA DE BITS

Una imagen está compuesta por infinitos puntos, y a cada uno de ellos se le asocia un atributo que puede ser su nivel de gris, en el caso de una imagen en blanco y negro, o su color, si la imagen es en color. Como no podemos almacenar y procesar los atributos de los infinitos puntos, los sistemas de captación consideran la imagen dividida en una fina retícula de celdas o **elementos de imagen o píxeles**, y a cada uno de ellos se le asigna como atributo el nivel de gris medio o el color medio de la celda correspondiente. En consecuencia para codificar y almacenar la imagen hay que tener en cuenta dos factores: el número de puntos a considerar (o **resolución**) en una superficie dada y la forma de representar (codificar) el atributo asociado a cada uno de ellos. En la Tabla 2.5 se indican las resoluciones usualmente utilizadas para representar imágenes. Un archivo en mapa de bits contiene una cabecera especificando el formato, las dimensiones de los píxeles y las características de la representación, y a continuación los atributos de los distintos puntos ordenados, por ejemplo de arriba abajo y de derecha a izquierda.

En el caso de imágenes en color, éste se descompone en tres colores básicos: rojo (R), verde (G) y azul (B), y la intensidad media de cada uno de ellos en cada celda se codifica por separado. Para conseguir una gran calidad de colores (calidad fotográfica), cada color básico debe codificarse con 8 bits; es decir, se necesitarán 3 bytes para codificar cada píxel, lo que da lugar a que se necesiten

		Resolución (horizontal × vertical)	Movimiento
Convencionales	Fax (A4)	(100, 200, 400) × (200, 300, 400) ei/pulgada	Estática
	Foto (8" × 11")	128, 400, 1.200 ei/pulgada	Estática
Televisión	Videoconferencia	176 × 144 ei/imagen	10 a 36 imágenes/s
	TV	720 × 480 ei/imagen	30 imágenes/s
	HDTV (TV alta definición)	1.920 × 1.080 ei/imagen	30 imágenes/s
Pantalla computador	VGA	640 × 480 ei	
	SVGA	800 × 600 ei	
	XGA	1.024 × 768 ei	

**Tabla 2.5.** Resoluciones usuales para codificar imágenes.

archivos de una gran capacidad para almacenar una imagen. El formato que hemos descrito es la base de las representaciones **BMP** (*Windows BitMaP*) y **TIFF** (*Tagged Image File Formats*); en la Sección 2.6 se describirá el fundamento de los formatos JPEG y GIF que consiguen una reducción muy notable del tamaño de los archivos.

### EJEMPLO 2.2

Obtener la capacidad de memoria que ocupará una imagen en color con una resolución XGA y con 256 niveles para representar cada color básico.

#### SOLUCIÓN

Para codificar el atributo (nivel de gris) se necesitan 3 Bytes, ya que cada color básico necesita 1 Byte ( $2^8 = 256$ ).

Por otra parte, de acuerdo con la Tabla 2.5, para obtener calidad XGA hay que almacenar el atributo de  $1.024 \times 768 = 768$  K elementos. Con lo que la capacidad total será:

$$C = 768 \cdot 3 = 2.304 \text{ KB} = 2,25 \text{ MBytes} \quad [2.3]$$

### 2.3.2. MAPAS DE VECTORES

Otros métodos de representar una imagen se fundamentan en descomponer ésta en una colección de objetos tales como líneas, polígonos y textos con sus respectivos atributos o detalles (grosor, color, etcétera) modelables por medio de vectores y ecuaciones matemáticas que determinan tanto su forma como su posición dentro de la imagen. Cuando se visualiza una imagen en una pantalla o impresora determinadas, un programa evalúa las ecuaciones y escala los vectores generando la imagen concreta a ver. Algunas características de este tipo de representación son las siguientes:

- Son adecuadas para gráficos de tipo geométrico y no para imágenes reales (tales como pinturas); en particular resulta muy adecuada en aplicaciones de diseño con ayuda de computador (CAD) en los que se utilizan imágenes compuestas de objetos geométricos que se visualizan y manipulan en dos y tres dimensiones.
- En comparación con la representación con mapas de bits, la representación con mapa de vectores genera usualmente archivos que ocupan menos espacio, y las imágenes son más fáciles de reescalar a cualquier tamaño y de procesar, ya que, por ejemplo, es más rápido mover un objeto o cambiar sus parámetros que recalculas las nuevas posiciones de miles de píxeles; por el contrario, la calidad y la fidelidad de la imagen, en comparación con la realidad, es peor.

En la Tabla 2.4 se incluyen los formatos más conocidos de representación vectorial de imágenes.



## 2.4. Representación de datos numéricos

Tradicionalmente los datos se introducen en un computador usando el lenguaje escrito, y por tanto se codifican, como cualquier tipo de texto, de acuerdo con un código de E/S, tal como el ASCII. Los códigos de E/S representan los datos numéricos como cualquier otra secuencia de caracteres. Si se va a realizar algún cálculo matemático la representación de los datos numéricos como textos es inapropiada. En efecto, como este tipo de codificación no se basa en los sistemas de numeración matemáticos, no podemos aplicar las tablas y reglas para operar con números representados en forma aritmética. La solución adoptada para representar datos numéricos es la siguiente. Cuando se introduce en el computador un número se codifica y almacena como un texto o cadena de caracteres cualquiera. Ahora bien, dentro de un programa, cada dato tiene asociado un **tipo de dato** determinado (ver Capítulo 10). El programador debe asociar a cada dato o variable el tipo adecuado, en consonancia con las operaciones que se realicen con él. Así, por ejemplo, y por lo que respecta a los datos numéricos, en el lenguaje C los principales tipos de datos aritméticos son los que se indican en la Tabla 2.6. El programador elige el tipo de dato más adecuado de acuerdo con los objetivos de la variable que define, y teniendo en cuenta que cuanto mayor es el rango y precisión del número, más ocupará la variable en la memoria (ver la columna *N.º de bits* de la tabla).

Tipo		N.º de bits	Rango de valores	Precisión (dígitos decimales)
Tipos enteros	Carácter	8	-128,127	3
	Carácter sin signo	8	0 a 255	3
	Entero corto	16	-32.768 a 32.767	3
	Entero corto sin signo	16	0 a 65.535	5
	Enumerado	16	-32.768 a 32.767	5
	Entero *	*	*	*
	Entero sin signo	*	*	*
	Entero largo	32	-2.147,484.648 a 2.147,484.648	10
	Entero largo sin signo	32	0 a 4.294,967.295	10
Tipos reales	Coma flotante	32	$\pm[3,4E-38$ a $3,4E38]$ , 0	7
	Coma flotante doble	64	$\pm[1,7E-308$ a $1,7E308]$ , 0	15
	Coma flotante doble largo	80	$\pm[3,4E-4932$ a $1,1E4932]$ , 0	19

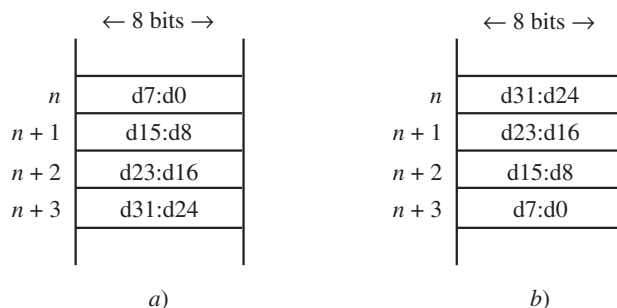
\* En máquinas de 16 bits igual a entero corto, y en máquinas de 32 bits a entero largo.

**Tabla 2.6.** Principales tipos de datos aritméticos utilizables en el lenguaje de programación C++ (compilador Borland para PC).

En la columna 3 de la Tabla 2.6 se observa que cada tipo de datos ocupa un número determinado de bits. En cualquier caso, el dato completo debe encajar en palabras de memoria, de forma que si el tamaño del dato es mayor que la longitud de palabra de memoria, aquél se trocea convenientemente, como indica el ejemplo de la Figura 2.2. Una vez dividido el patrón de bits que representa al dato en porciones que encajen exactamente en posiciones sucesivas de memoria, dichas porciones se almacenan en posiciones consecutivas de memoria; pero hay dos posibilidades de almacenamiento:

- Almacenar primero (en las posiciones más bajas de memoria) la parte menos significativa del dato (Figura 2.2a); este convenio se denomina **criterio del extremo menor**.
- Almacenar primero la parte más significativa del dato (Figura 2.2b); este es el **criterio del extremo mayor**.

Los mismos criterios se utilizan para almacenar cadenas de caracteres. Aunque seguir un criterio u otro es irrelevante, no hay un convenio que establezca cuál de los dos debe utilizarse, así hay computadores que siguen el primero y otros el segundo.



**Figura 2.2.** Formas de almacenar un dato numérico ( $d31:d0$ ) de 32 bits en una memoria direccionable por bytes siguiendo: a) el criterio del extremo menor y b) criterio del extremo mayor.

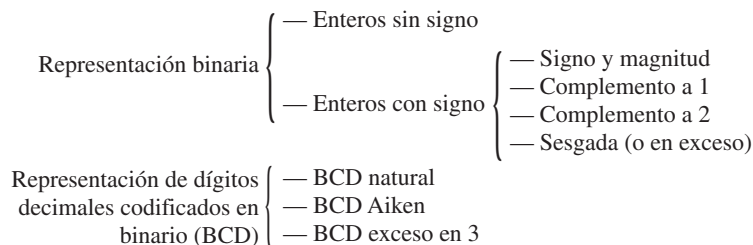
Una vez definidos los datos numéricos de un programa o de una aplicación, una rutina de la biblioteca del traductor del lenguaje de programación se encarga de transformar la cadena de caracteres que simboliza el número en su representación numérica.

Hay dos formas básicas de representar los datos numéricos: como números enteros o como números reales. Las Secciones 2.4.1 y 2.4.2 se dedican a describir los principios de estas representaciones. En todos los casos denominaremos  $N$  al dato,  $n$  al número total de bits dedicados a almacenarlo y los bits individuales del dato de forma que el bit **más significativo** (el de la izquierda) es el  $n - 1$ , y el **menos significativo** (el de la derecha) es el 0.

### 2.4.1. DATOS DE TIPO ENTERO

Se distinguen (Figura 2.3) dos formas básicas de representar en el interior del computador los datos de tipo entero: **representación binaria** y representación de **dígitos decimales codificados en binario** (o **representación BCD**). A su vez, dentro de la representación binaria se tienen dos situaciones, representación sin signo y representación con signo. En este último caso es usual considerar cuatro casos diferentes: signo y magnitud, complemento a 1, complemento a 2 y representación sesgada (o por exceso). En la representación BCD, a su vez, cada dígito decimal se puede codificar de tres formas distintas: natural, Aiken y exceso de 3; aquí únicamente describiremos la natural. Los lenguajes de programación admiten simultáneamente distintas representaciones para los números enteros (un ejemplo está en la Tabla 2.6), y el programador elige para cada variable que define la más adecuada. En la Tabla 2.7 se incluyen los valores representables en los distintos sistemas, utilizando  $n = 4$  bits para almacenar cada número, obsérvese que en algunas representaciones hay dos patrones para almacenar el cero. A continuación se describen las diferentes representaciones citadas:

- a) **Enteros sin signo:** En las  $n$  posiciones del dato se almacena el número en binario natural.
- b) **Enteros en signo y magnitud:** El signo se representa con el bit más significativo del dato (bit  $n - 1$ ). Este bit es 0 si el número es positivo y 1 si el número es negativo. El resto de los bits ( $n - 2$  a 0) representan el valor absoluto del número en binario natural.



**Figura 2.3.** Diferentes representaciones de datos enteros.

- c) **Enteros en complemento a uno:** El signo se representa de la misma forma que en el caso de signo y magnitud. El resto de los bits ( $n - 2$  a 0) representan, si el número es positivo ( $N > 0$ ) su valor absoluto en binario natural, y si no ( $N < 0$ ) su complemento a uno.
- d) **Enteros en complemento a dos:** El signo se representa de la misma forma que en el caso de signo y magnitud. El resto de los bits ( $n - 2$  a 0) representan, si el número es positivo ( $N > 0$ ) su valor absoluto en binario natural, y si no ( $N < 0$ ) su complemento a dos.
- e) **Representación sesgada:** Se le suma al número a representar,  $N$ , un sesgo  $S$ , de forma tal que el número resultante siempre es positivo, no siendo necesario reservar explícitamente un bit de signo. El dato almacenado es sencillamente el valor de  $N + S$  en binario natural. Usualmente se toma como sesgo  $S = 2^{n-1}$ . Esta notación también se le suele denominar **representación con exceso**. Obsérvese que en la representación sesgada los números positivos empiezan por 1 y los negativos por 0.

N.º decimal	Tipo de representación			
	Signo y magnitud	Complemento a 1	Complemento a 2	Sesgada
7	0111	0111	0111	1111
6	0110	0110	0110	1110
5	0101	0101	0101	1101
4	0100	0100	0100	1100
3	0011	0011	0011	1011
2	0010	0010	0010	1010
1	0001	0001	0001	1001
+0	0000	0000	0000	1000
-0	1000	1111	—	—
-1	1001	1110	1111	0111
-2	1010	1101	1110	0110
-3	1011	1100	1101	0101
-4	1100	1011	1100	0100
-5	1101	1010	1011	0011
-6	1110	1001	1010	0010
-7	1111	1000	1001	0001
-8	(11000)	—	1000	0000

**Tabla 2.7.** Alternativas usuales de representación de datos de tipo entero con signo en el supuesto de datos de  $n = 4$  bits.

La forma más utilizada en la actualidad para representar los números enteros con signo es la de complemento a dos. Ello se debe a la facilidad de efectuar las sumas y restas con este tipo de representación (ver Apéndice 1). La representación sesgada, como se indica en la Sección 2.4.2, se utiliza para representar los exponentes de los datos de tipo real.

En la Tabla 2.8 se incluyen los límites máximo y mínimo de los números enteros para distintas longitudes de palabra y tipos de representación.

### EJEMPLO 2.3

Obtener la representación interna del número  $N = -3675$  en los cuatro formas de representación vistas anteriormente, suponiendo que se tiene que almacenar en 2 Bytes.

#### SOLUCIÓN

1. *Transformamos a binario* el valor absoluto del número, para lo cual previamente lo pasamos a hexadecimal:

$$\begin{array}{r}
 3675 \mid 16 \\
 047 \quad 229 \mid 16 \\
 155 \quad 069 \quad 14 \\
 11 \quad 05
 \end{array}$$

Representación	Valor mínimo		Valor máximo	
	Patrón binario	Decimal	Patrón binario	Decimal
Sin signo	0000...0000	$N(\text{mín}) = 0$	1111...1111	$N(\text{máx}) = 2^n - 1$
Signo y magnitud	1   1111...1111	$N(\text{mín}) = -(2^{n-1} - 1)$	0   1111...1111	$N(\text{máx}) = 2^{n-1} - 1$
Complemento a 1	1   0000...0000	$N(\text{mín}) = -(2^{n-1} - 1)$	0   0000...0000	$N(\text{máx}) = 2^{n-1} - 1$
Complemento a 2	1   0000...0000	$N(\text{mín}) = -2^{n-1}$	1   0000...0000	$N(\text{máx}) = 2^{n-1} - 1$
Sesgado	0000...0000	$N(\text{mín}) = -2^{n-1}$	1111...1111	$N(\text{máx}) = 2^{n-1} - 1$

**Tabla 2.8.** Valores máximos y mínimos representables en distintos formatos de números enteros.

Consultando la Tabla A1.7:  $14 \rightarrow \text{E}$ ,  $5 \rightarrow 5$ ,  $11 \rightarrow \text{B}$ ; es decir<sup>1</sup>,

$$\text{D}'3675 = \text{H}'0\text{E}5\text{B} = \text{B}' 0000\ 1110\ 0101\ 1011$$

2. *Representación en signo y magnitud:*

*Signo:* como  $N < 0$ , será  $d(n-1) = 1$ , con lo que:

$$\mathbf{1000\ 1110\ 0101\ 1011}$$

o, abreviadamente: **8E5B**.

3. *Representación en complemento a 1.*

Como el número es negativo, hay que obtener el complemento a 1 de  $N$ , para lo cual podemos utilizar el valor hexadecimal de  $N$  (0E5B) obtenido anteriormente:

$$\begin{array}{r} \text{FFFF} \\ - \text{0E5B} \\ \hline \end{array}$$

$$\mathbf{F1A4} \quad \text{o, en forma ampliada:} \quad \mathbf{1111\ 0001\ 1010\ 0100}$$

Observar que en el resultado se obtiene directamente el signo:  $d(n-1) = 1$ .

4. *Representación en complemento a 2.*

Añadimos 1 al resultado anterior:

$$\begin{array}{r} \text{F1A4} \\ + \text{0001} \\ \hline \end{array}$$

$$\mathbf{F1A5} \quad \text{o, en forma ampliada:} \quad \mathbf{1111\ 0001\ 1010\ 0101}$$

5. *Representación sesgada.*

Tenemos que añadir al valor de  $N$  el sesgo, que en este caso será:

$$S = 2^{n-1} = 2^{15} = \text{D}'32.768 = \text{B}'1000\ 0000\ 0000\ 0000$$

Hagámoslo, por ejemplo, en binario (como  $N < 0$ , en realidad tenemos que efectuar  $S - |N|$ ):

$$\begin{array}{r} 1000\ 0000\ 0000\ 0000 \\ - \quad 0000\ 1110\ 0101\ 1011 \\ \hline 0111\ 0001\ 1010\ 0101 \end{array}$$

<sup>1</sup> D' quiere decir que el número que sigue es decimal, H' quiere decir que el número que sigue es hexadecimal y B' quiere decir que el número que sigue es binario.

Es decir, la representación interna del número será:

**0111 0001 1010 0101**; o abreviadamente: **71A5**

Mención aparte merece el formato de **datos enteros representados con dígitos decimales codificados en binario**, o, abreviadamente, **BCD**. En ocasiones los datos de tipo entero se representan internamente codificando aisladamente cada dígito decimal con cuatro dígitos binarios, según la Tabla 2.9. De esta forma, en un byte se pueden representar 2 dígitos decimales, denominándose esta representación **BCD empaquetada**, o bien un único dígito decimal, obteniéndose una representación **BCD desempaquetada**.

Esta forma de codificar es poco eficiente, puesto que de las  $2^4 = 16$  combinaciones posibles de 4 bits sólo se utilizan 10 (Tabla 2.9). No obstante, a veces se utiliza por la proximidad a nuestro sistema decimal y por la gran facilidad de codificar en BCD, sin más que considerar aisladamente cada dígito decimal según la Tabla 2.9. Los circuitos decodificadores y los algoritmos de transformación de código de E/S a BCD son muy sencillos.

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

**Tabla 2.9.** Dígitos decimales codificados en binario.

#### EJEMPLO 2.4

Representar en BCD el número decimal 98325.

**SOLUCIÓN**

Consultando la Tabla 2.9 se tiene:

$$\begin{array}{ccccc} 9 & 8 & 3 & 2 & 5 \\ 1001 & 1000 & 0011 & 0010 & 0101 \end{array} = \text{BCD}'1001\ 1000\ 0011\ 0010\ 0101$$

En la representación BCD de datos con signo se suelen utilizar cuatro bits para representar al signo. Puede ser, por ejemplo, 0000 para el signo positivo y 1001 para el signo negativo.

#### 2.4.2. DATOS DE TIPO REAL

La representación para datos de tipo real también denominada **notación exponencial** o **notación científica** o **notación en coma flotante**, se fundamenta en representar los números reales bajo la forma:

$$N = M \cdot B^E \quad [2.4]$$

donde es habitual denominar **mantisa** a  $M$ , **exponente** a  $E$  y **base** a  $B$ .

Un ejemplo de número en notación exponencial es el siguiente:

$$N = 0,0356445 = 356445 \cdot 10^{-7} = 3,56445 \cdot 10^{-2} \quad [2.5]$$

En este caso la base es 10. Obsérvese que si la mantisa está representada en el sistema de numeración de la base, el número queda inalterado si al desplazar la coma a la izquierda se suma al exponente el número de desplazamientos realizados; y si al desplazar la coma a la derecha se resta al exponente el número de desplazamientos realizados. Se denomina **forma normalizada** a la representación del número en la que la cifra más significativa se ubica en la posición cero (unidades). La última de las expresiones de [2.5] es la forma normalizada de ese número.

En la actualidad el sistema de representación que más se utiliza es el estándar IEEE 754, y que a continuación se va a analizar. En esta notación el número se considera en la forma de la expresión [2.4], estando el número normalizado y siendo  $B = 2$ , tanto  $M$  como  $N$  en el sistema de numeración 2. Por ejemplo, la forma de que parte la representación es la que se indica en el siguiente ejemplo:

$$N = 1,001001 \cdot 2^{-5} \quad [2.6]$$

donde, en este caso:  $M = 1,001001$ ;  $E = -5$  y  $B = 2$ .

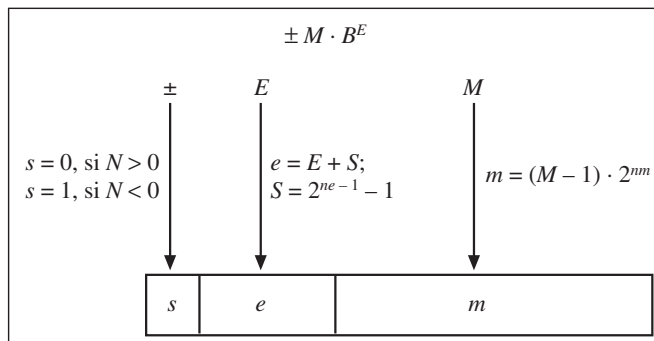
Para representar el número se utilizan  $n$  bits; de los cuales uno es para almacenar el signo,  $nm$  para almacenar la mantisa y  $ne$  para almacenar el exponente. La variante más utilizada de la notación IEEE 754 es la de **simple precisión**, en la que:

$$\begin{aligned} n &= 32 \text{ bits} \\ ne &= 8 \text{ bits} \\ nm &= 23 \text{ bits} \end{aligned}$$

El orden de almacenamiento, dentro del patrón de  $n$  bits, es signo, exponente y mantisa (Figura 2.4); y los valores que se almacenan son:

- **Signo** ( $s$ ), si  $N > 0$  se hace  $s = 0$ ; y si  $N < 0$  se hace  $s = 1$ .
- **Exponente** ( $e$ ), es el exponente sesgado:

$$e = E + S; \text{ siendo } S = 2^{ne-1} - 1$$



**Figura 2.4.** Esquema de representación de un número en notación IEEE754.

- **Mantisa** ( $m$ ) se almacena sólo la parte fraccionaria de la mantisa normalizada, lo que denotamos de la siguiente manera:

$$M = [1, m]; \text{ donde } m = (M - 1) \cdot 2^{nm}$$

En la Figura 2.4 se representan esquemáticamente las transformaciones entre el número de partida y los valores a incluir dentro del patrón de  $n$  bits.

### EJEMPLO 2.5

El número de la expresión [2.6] se representaría, en IEEE754, simple precisión, de la siguiente forma:

- **Signo:**  $s = 0$ , ya que  $N > 0$ .

- **Exponente:**

$$e = -5 + 127 = 122 = 0111\ 1010 ; \text{ ya que } S = 2^7 - 1 = 127$$

- **Mantisa:**  $m = 001001$ .

Con lo que el patrón de bits que representa al número  $N$  es:

0	0111 1010	001 0010 0000 0000 0000 0000
---	-----------	------------------------------

Con frecuencia se abrevia la escritura del número dándolo en hexadecimal:

3 D 1 2 0 0 0 0
-----------------

### EJEMPLO 2.6

Obtener la representación interna del número decimal  $N = -543,7 \cdot 10^{-17}$ , según la norma IEEE754 para datos en simple precisión.

#### SOLUCIÓN

Como se ha indicado en los párrafos anteriores, en la notación del IEEE754:  $n = 32$ ,  $ne = 8$ ,  $nm = 23$ ,  $B = 2$ , y la normalización se efectúa con el 1 más significativo en la posición de los enteros.

Para hacer la transformación el número original lo tenemos que poner en la forma:

$$N = -543,7 \cdot 10^{-17} = M \cdot 2^E \quad [2.7]$$

donde debemos normalizar la mantisa, pasar  $M$  y  $E$  a binario y sesgar el exponente. Para pasar  $M$  y  $E$  a binario es más cómodo utilizar la notación hexadecimal. También se simplifican los cálculos si antes de operar con el número original lo normalizamos.

a) *Transformación del número a la forma dada por la expresión [2.7].*

Primero normalizamos el número original:

$$N = -543,7 \cdot 10^{-17} = -5,437 \cdot 10^{-15} \quad [2.8]$$

Para pasar a la forma tenemos que tener en cuenta que:

$$10^{-15} = 2^{-x} \Rightarrow -15 \cdot \log(10) = x \cdot \log(2) \quad [2.9]$$

es decir:

$$x = -15 \cdot \frac{\log(10)}{\log(2)} = \frac{-15}{0,301029995} = -49,82892142 \quad [2.10]$$

Como el exponente de 2, en la expresión [2.7], debe ser entero, la parte fraccionaria del valor de  $x$  obtenido la debemos incluir en la mantisa inicial de  $N$ :

$$N = -5,437 \cdot 2^{-49,828921427} = -(5,437 \cdot 2^{-0,8282142}) \cdot 2^{-49} = -3,060758897 \cdot 2^{-49} \quad [2.11]$$

b) *Transformación de la mantisa a binario.*

Utilizamos la notación hexadecimal como código intermedio. Como tenemos que obtener 26 bits (24 de la mantisa y 2 de redondeo) debemos calcular en total  $26/4 = 6,5$  cifras hexadecimales (en realidad 7):

$$\begin{aligned}
3, &= 3, \rightarrow D'3 = H'6 \rightarrow 3, \\
0,060758897 \times 16 &= 0,972142348 \rightarrow D'0 = H'0 \rightarrow 3,0 \\
0,972142348 \times 16 &= 15,55427758 \rightarrow D'15 = H'F \rightarrow 3,0F \\
0,55427758 \times 16 &= 8,86844121 \rightarrow D'8 = H'8 \rightarrow 3,0F8 \\
0,86844121 \times 16 &= 13,89505935 \rightarrow D'13 = H'D \rightarrow 3,0F8D \\
0,89505935 \times 16 &= 14,32094966 \rightarrow D'14 = H'E \rightarrow 3,0F8DE \\
0,32094966 \times 16 &= 5,135194522 \rightarrow D'5 = H'5 \rightarrow 3,0F8DE5
\end{aligned}$$

Es decir:

$$\begin{aligned}
N &= -3,060758897 \cdot 2^{-49} = -3,0F8DE5 \cdot 2^{-49} = \\
&= -11,0000 \ 1111 \ 1000 \ 1101 \ 1110 \ 0101 \cdot 2^{-49} = \\
&= -1,100 \ 0011 \ 1110 \ 0011 \ 0111 \ 100101 \cdot 2^{-48}
\end{aligned} \quad [2.12]$$

donde hemos normalizado el número resultante. De la mantisa final únicamente se utilizarán 23 bits de la parte fraccionaria, con lo que los dos últimos (01) corresponden a los bits de guarda y redondeo, respectivamente. Al ser estos bits 01, el redondeo al más próximo se realiza truncando. En consecuencia los 23 bits de la mantisa a almacenar son:

$$m = 100 \ 0011 \ 1110 \ 0011 \ 0111 \ 1001 \quad [2.13]$$

c) *Exponente a almacenar.*

Ahora debemos sesgar el exponente (−48) y pasarlo a binario:

$$e = E + S = -48 + 127 = D'79 = H'4F = B'0100 \ 1111 \quad [2.14]$$

d) *Número empaquetado.*

El número final, teniendo en cuenta que el número es negativo, será:

<i>signo</i>	<i>exponente</i>	<i>mantisa</i>
1	010 0111 1	100 0011 1110 0011 0111 1001

o, abreviadamente:

**A7C3E379**

## Redondeos

Puede observarse que, por lo general, al transformar un número decimal a número binario se obtienen infinitas cifras binarias significativas, de las que sólo se pueden incluir en el patrón un número determinado que lo representa (24 en el caso de simple precisión). Para reducir al máximo este error de representación se consideran las dos cifras más significativas que no caben en el patrón (que se suelen denominar **bits de guarda**), y se sigue el siguiente procedimiento: si los dos bits de guarda más significativos son:

- 00 o 01, el número se redondea por defecto (se trunca).
- 11, se redondea por exceso (se suma 1 a la cifra menos significativa que se conserva en el patrón).
- 10, se efectúa un **redondeo al par**, consistente en si la cifra menos significativa que se conserva es 0, se trunca el número, y si es 1, se redondea por exceso (se le suma 1). Este sistema se llama redondeo al par porque siempre la cifra menos significativa que queda es 0.



**EJEMPLO 2.7**

Supóngase que las mantisas de los datos se almacenan en 5 bits ( $nm = 5$ ), indicar los números resultantes de efectuar el redondeo según el estándar IEEE754 de los números que se indican en la primera columna de la siguiente tabla.

Resultado en la ALU	Acción	Mantisa redondeada	Comentario
1,01101 00	Truncar	1,01101	Redondeo al más próximo.
1,01100 00	Truncar	1,01100	Redondeo al más próximo.
1,01101 01	Truncar	1,01101	Redondeo al más próximo.
1,01100 01	Truncar	1,01100	Redondeo al más próximo.
1,01101 10	Sumar 0,00001	1,01110	Redondeo al par: como el bit $-5$ es 1, se suma 1.
1,01100 10	Truncar	1,01100	Redondeo al par: como el bit $-5$ es 0, se trunca.
1,01101 11	Sumar 0,00001	1,01110	Redondeo al más próximo.
1,01100 11	Sumar 0,00001	1,01101	Redondeo al más próximo.

**Situaciones especiales**

- Cuando el campo del exponente toma su valor mínimo; es decir,  $e = 0$ , el 1 más significativo de la mantisa no se encuentra implícito, y entonces se almacena la **mantisa denormalizada** (con la parte entera cero, y el uno más significativo en la parte fraccionaria). En este caso el sesgo es:  $S = 2^{ne-1} - 2$ ; es decir, el valor del exponente correspondiente a los números denormalizados es:  $E = e - S = -2^{ne-1} + 2$  ( $-126$  en simple precisión).
- El número  $N = 0$  se representa con todos los bits del campo del exponente y del campo de la mantisa cero; esto es:  $e = 0$ ,  $m = 0$ .
- Si todos los bits del campo del exponente son unos (es decir, adquiere su valor máximo), el dato representa:
  - Si  $m = 0$ , más o menos infinito (el resultado de dividir por 0, por ejemplo).
  - Si  $m \neq 0$  no representa un número (es un código **NaN**, *No a Number*). Estos patrones de bits se utilizan para almacenar valores no válidos (resultados de operaciones tales como  $0 \cdot \infty$ ,  $\infty \cdot \infty$ , raíz cuadrada de un número negativo, etc.).

**Precisiones**

El estándar IEEE754 considera cuatro tamaños o precisiones posibles de datos (cuanto mayor sea  $nm$ , mayor es el número de cifras significativas y por tanto mayor es la precisión): simple precisión ( $n = 32$ ), simple ampliada, doble y doble ampliada; aunque el estándar sólo especifica completamente las precisiones sencilla y doble, según se indica en la Tabla 2.10.

	Precisión			
	Simple	Simple ampliada	Doble	Doble ampliada
$nm + 1$ (bits de precisión)	24	32	53	64
$E(\text{máx})$	127	1023	1023	16383
$E(\text{mín})$	-126	-1022	-1022	-16382
$S$ (sesgo del exponente)	127	(n.e.)	1023	(n.e.)

(n.e.: no especificado por el estándar)

**Tabla 2.10.** Tipos de precisión contemplados en el estándar IEEE754.

**Valores límite**

Teniendo en cuenta las situaciones especiales y el número de bits de cada campo, podemos obtener los números máximos y mínimos representables. En Figura 2.5 se indican los patrones correspondientes a los números mayores y menores, en precisión sencilla. También se indican sus correspondientes valores decimales.

**a) Infinito**

0	1111 1111	000 0000 0000 0000 0000 0000	$\infty$
---	-----------	------------------------------	----------

**b) Número mayor,  $N(máx)$** 

0	1111 1110	111 1111 1111 1111 1111 1111	$1,99999988 \cdot 2^{127} = 3,40 \cdot 10^{38}$
---	-----------	------------------------------	---

**c) Número menor normalizado,  $N(mín,nor)$** 

0	0000 0001	000 0000 0000 0000 0000 0000	$1 \cdot 2^{-126} = 1,18 \cdot 10^{-38}$
---	-----------	------------------------------	--

**d) Número menor denormalizado,  $N(mín,den)$** 

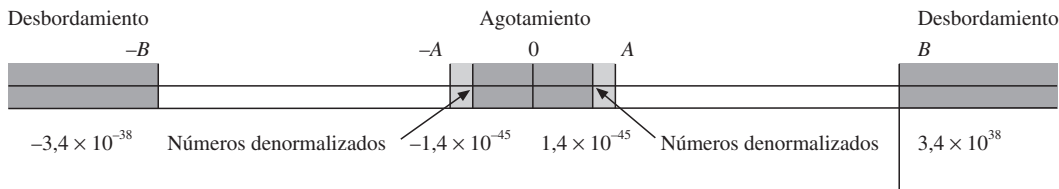
0	0000 0000	000 0000 0000 0000 0000 0001	$2^{-23} \cdot 2^{-126} = 2^{-149} = 1,40 \cdot 10^{-45}$
---	-----------	------------------------------	---

**e) Cero**

0	0000 0000	000 0000 0000 0000 0000 0000	0
---	-----------	------------------------------	---

**Figura 2.5.** Patrones para los límites de números positivos (mayores y menores) representables en IEEE754, precisión sencilla.

Los valores máximo  $N(máx)$  y mínimo  $N(mín)$  anteriores se tendrán tanto para números positivos como negativos (Figura 2.6). Obsérvese que los números fuera de los límites establecidos (zonas en gris en la figura) no son representables. Si, como resultado de una operación el número,  $N$ , tuviese que estar en la zona  $-N(mín,den) < N < N(mín,den)$  con  $N \neq 0$ , se dice que se ha producido un **agotamiento** (*underflow*). Si tuviese que estar en la zona  $N < -N(máx)$  y  $N > N(máx)$ , con  $N \neq \pm\infty$  se produciría un **desbordamiento** (*overflow*).



**Figura 2.6.** Límites de números reales representables en IEEE754.

## 2.5. Detección de errores

Según indicamos en la Sección 2.1 para representar  $m$  símbolos distintos necesitamos al menos  $n$  bits, siendo  $n$  el menor número natural que verifica la expresión [2.2]. A veces no se utilizan todas las com-

binaciones posibles de los  $n$  bits para codificar símbolos y cuantas menos combinaciones se desperdicien, es decir, no se utilicen para codificar símbolos, se dice que el código es más eficiente. La **eficiencia de un código** ( $\tau$ ) se define como el cociente entre el número de símbolos que se representan realmente,  $m$ , dividido por el número,  $m'$ , de símbolos que en total pueden representarse; es decir, con códigos binarios en que  $m' = 2^n$  se tiene:

$$\tau = \frac{m}{m'} = \frac{m}{2^n}, \text{ con } 0 \leq \tau \leq 1 \quad [2.15]$$

Obviamente cuanto más eficiente (*mejor aprovechado*) sea el código,  $\tau$  será mayor.

### EJEMPLO 2.8

Supongamos que utilizamos el código ASCII básico ( $n = 7$  bits) para representar 95 símbolos. ¿Qué eficiencia se obtiene con este código? ¿Y si se utilizase un bit más?

#### SOLUCIÓN

La eficiencia de esta codificación será:

$$\tau = \frac{m}{2^n} = \frac{95}{2^7} = 0,742$$

Si introdujésemos un bit adicional, la nueva eficiencia sería:

$$\tau = \frac{m}{2^n} = \frac{95}{2^8} = 0,371$$

Un código que es poco eficiente se dice que es redundante (más redundancias supone menos eficiencia), definiéndose la **redundancia** como:

$$R = (1 - \tau) \cdot 100\% \quad [2.16]$$

### EJEMPLO 2.9

Obtener la redundancia para los casos considerados en el Ejemplo 2.8. Aplicando la expresión [2.16] se tiene:

$$\begin{aligned} R &= (1 - 0,742) \cdot 100 = 25,8\% \\ R &= (1 - 0,371) \cdot 100 = 62,9\% \end{aligned} \quad [2.17]$$

A veces las redundancias se introducen deliberadamente para poder detectar posibles errores en la transmisión o grabación de información. Así, por ejemplo, si necesitamos transmitir 8 símbolos  $\{A, B, C, D, E, F, G \text{ y } H\}$  y se hace con un código sin redundancias, necesitaríamos  $n = 3$  bits, y un código posible puede ser el Código I de la Tabla 2.11. Si por algún error varía uno de los bits, obtenemos otro símbolo del alfabeto, que considerado por sí mismo (es decir, aisladamente) no puede ser detectado como erróneo. Si se utilizase un código redundante, tal como el Código II de la Tabla 2.11, existirían algunas posibilidades de detectar errores. Así, por ejemplo, si se transmite el símbolo  $H$ , esto es, 1111, y por un error en la transmisión cambiase el primer bit, esto es, se recibiese 0111, podría detectarse el error, ya que éste no corresponde a ninguno de los símbolos posibles.

La introducción de redundancias con objeto de detectar o corregir errores se efectúa de acuerdo con algún algoritmo predeterminado; de esta manera los códigos pueden comprobarse automática-

Alfabeto	Código I	Código II
A	000	1000
B	001	1001
C	010	1010
D	011	1011
E	100	1100
F	101	1101
G	110	1110
H	111	1111

**Tabla 2.11.** Introducción de redundancias en un código.

mente por circuitos adecuados. Uno de los algoritmos más conocidos añade al código inicial de cada carácter un nuevo bit denominado **bit de paridad**. Existen dos criterios para introducir este bit:

- **Paridad par:** Se añade un bit (0 o 1) de forma tal que el número total de unos de código que resulte sea par.
- **Paridad impar:** Se añade un bit (0 o 1) de forma tal que el número total de unos del código que resulte sea impar.

El ejemplo que se da a continuación aclara la cuestión.

### EJEMPLO 2.10

Insertar un bit de paridad en los datos que aparecen en la primera columna de las siguientes tablas.

<i>Mensaje o código inicial</i>	<i>Mensaje o código con bit de paridad (criterio par)</i>
111 0001	<b>0</b> 111 0001
011 1011	<b>1</b> 011 1011
110 1000	<b>1</b> 110 1000
101 0101	<b>0</b> 101 0101
	<i>Bit de paridad</i>
<i>Mensaje o código inicial</i>	<i>Mensaje o código con bit de paridad (criterio impar)</i>
100 0001	<b>1</b> 100 0001
010 1111	<b>0</b> 010 1111
110 1000	<b>0</b> 110 1000
111 0111	<b>1</b> 111 0111
	<i>Bit de paridad</i>

En la segunda columna de cada una de las tablas hemos incluido la solución.

El bit de paridad se introduce antes de transmitir o grabar la información (en la memoria principal, cinta o disco magnético). Por ruido o interferencias en la transmisión o defecto del soporte de información puede cambiar un bit (de 1 a 0 o de 0 a 1). Si en el receptor o al leer la información se comprueba la paridad del mensaje (esto es, se cuenta su número de unos), se detectaría el error, ya que el número de unos dejaría de ser par (en el caso de criterio par) o impar (en el caso de criterio impar).

## 2.6. Compresión de datos

Según se han ido incrementando los campos de aplicación de la informática, han ido aumentando las necesidades de capacidad de los archivos donde almacenar la información, y además, como conse-

cuencia, han crecido sustancialmente los tiempos necesarios para transmitir dichos archivos a través de redes. Este problema se presenta de forma notable en las aplicaciones multimedia, donde es frecuente utilizar combinadamente textos, sonidos e imágenes, tanto estáticas como dinámicas. Para reducir considerablemente el tamaño de los archivos se han desarrollado técnicas de **compresión de datos** que reducen las redundancias o recodifican la información. La información así comprimida puede almacenarse y transmitirse ocupando menos espacio y tiempo, respectivamente. Cuando, posteriormente, se desee procesar dicha información puede recuperarse en su forma original utilizando algoritmos de **descompresión de datos**.

Denominando  $C_a$  y  $C_d$  a las capacidades del archivo antes y después de comprimirlo, respectivamente, podemos definir el factor de compresión como:

$$f_c = \frac{C_a - C_d}{C_a} \cdot 100\% \quad [2.18]$$

También es frecuente indicar el factor de compresión en la forma:

*“compresión de  $C_d/C_a:1$ ”*

### EJEMPLO 2.11

Suponiendo que un archivo de una capacidad de 4 MB se comprime ocupando 2,5 MB, obtener su factor de compresión.

#### SOLUCIÓN

Aplicando la expresión (2.18) se tiene:

$$f_c = \frac{C_a - C_d}{C_a} \cdot 100\% = \frac{4 - 2,5}{2,5} \cdot 100 = 42,9\%$$

También se dice que se ha producido una compresión de 1,6:1.

**Codificación por longitud de secuencias** (o **RLE**, *Run Length Encoded*). Una forma sencilla de comprimir la información, en el caso de que en el archivo se repitan largas secuencias de ceros y unos, es sustituir cada una de éstas por el símbolo de la secuencia seguido por el número de veces que se repite en ella. Así, ocupa menos espacio almacenar: *uno 245 cero 94*, que los 339 bits que ocupa la secuencia original.

**Codificación relativa o incremental.** Con frecuencia en imágenes y señales de sonido, los valores de dos elementos de imagen consecutivos o de dos muestras de voz consecutivas en el tiempo difieren en una cantidad pequeña, en comparación con sus valores absolutos. Por ejemplo, las variaciones de color y de textura dentro de los objetos de una imagen suelen ser muy pequeñas, teniéndose sólo grandes variaciones en los contornos de dichos objetos. En este caso se logra una reducción considerable de almacenamiento si en lugar de almacenar los valores absolutos se almacenan los incrementos respecto al valor anterior.

**Codificación dependiente de la frecuencia.** Consiste en representar cada símbolo con un código binario cuya longitud sea inversamente proporcional a la frecuencia; de esta manera los símbolos que se repiten más en un archivo se representarán con menos bits que los símbolos que aparecen con menos frecuencia (ver Problemas P2.45 y P2.46).

**Codificación con diccionario adaptativo.** La idea básica consiste en realizar un diccionario con secuencias de bits, y sustituir en el archivo a comprimir dichas secuencias por su índice (número de or-

den) dentro del diccionario. El diccionario es adaptativo en el sentido de que según se va procesando la información para comprimirla, se va creando: dada una secuencia determinada, primero se consulta si ya está en el diccionario; si es así, se sustituye por el índice correspondiente; si no, la nueva secuencia se incluye en el diccionario y se sustituye en el archivo por el nuevo índice.

**Codificación Lempel-Ziv LZ77.** Es un caso particular de diccionario adaptativo, donde se busca si los siguientes caracteres a comprimir han aparecido previamente en una secuencia anterior; caso de ser así, esa cadena de caracteres se codifica con 3 números  $(m,n,s)$ , donde  $m$  representa el lugar hacia atrás donde se inicia la secuencia previa encontrada,  $n$  es la longitud de la secuencia y  $s$  es el siguiente carácter de la cadena comprimida. Se observa que en este caso realmente no se crea un diccionario de cadenas. El siguiente ejemplo pretende aclarar el proceso.

### EJEMPLO 2.12

Supóngase que se tiene la siguiente secuencia de bits:

0101111101000100001111101000001000011110111101000100000

Para realizar este ejercicio, primero vamos a ir identificando, en un punto dado, las mayores secuencias previas que coinciden con los próximos bits a comprimir, lo que se muestra a continuación (después de encontrar una secuencia, incluimos el próximo bit):

0	1	01	1	11	1	010	0	0100	0	011111010	0	000100001111	0	11110100010000	0
0,0,0	0,0,1	2,2,1	2,2,1	8,3,0	4,4,0	15,9,0	17,12,0	36,14,0							

Con lo que la secuencia comprimida es:

(0,0,0) (0,0,1) (2,2,1) (2,2,1) (8,3,0) (4,4,0) (15,9,0) (17,12,0) (36,14,0)

En el ejemplo, la información comprimida no parece muy inferior a la original; esto sencillamente se debe a que hemos puesto un ejemplo con tan sólo 50 bits y por tanto se generan secuencias muy cortas. Según aumenta el tamaño del archivo mayor probabilidad existe de encontrar secuencias previas más largas. En definitiva, la técnica consiste en encontrar la mayor secuencia previa que concuerde con los siguientes símbolos a comprimir del resto del archivo. Los conocidos programas **zip** y **unzip** para compresión y descompresión de datos utilizan procedimientos de este tipo.

**Compresión GIF (imágenes).** Con tres bytes (uno por cada color básico) se pueden obtener  $2^{24} = 16.777.216$  mezclas de colores distintos. Esas mezclas proporcionan una gama de colores extraordinariamente rica con **calidad fotográfica**. Sin embargo, podemos obtener una imagen de gran calidad sin necesidad de utilizar un número excesivo de dichas mezclas; así, usualmente con 256 mezclas distintas elegidas convenientemente de entre las  $2^{24}$  posibles se obtiene una calidad más que aceptable. El conjunto de mezclas seleccionadas se denomina **paleta de color**, y cada una de ellas se puede codificar con 1 byte. El fundamento del sistema de **compresión GIF** (*Graphic Interchange Format*) consiste en almacenar como atributo de cada elemento de imagen una de las combinaciones de la paleta (1 byte) en lugar de la mezcla concreta representada por 24 bits.

**Compresión JPEG (imágenes).** En la televisión en color, en lugar de transmitir las intensidades de los tres colores básicos ( $X_R$ ,  $X_G$ ,  $X_B$ ) de cada píxel se utilizan tres valores transformados, denominados **luminancia** (o brillo,  $X_Y$ ) y **crominancias** ( $X_I$ ,  $X_Q$ ), relacionadas con ( $X_R$ ,  $X_G$  y  $X_B$ ), como sigue:

$$\begin{aligned} X_Y &= 0,30 \cdot X_R + 0,59 \cdot X_G + 0,11 \cdot X_B \\ X_I &= 0,60 \cdot X_R - 0,28 \cdot X_G - 0,32 \cdot X_B \\ X_Q &= 0,21 \cdot X_R - 0,52 \cdot X_G + 0,31 \cdot X_B \end{aligned} \quad [2.19]$$

En los receptores de color, a partir de los tres valores anteriores se recuperan los valores de las intensidades  $X_R$ ,  $X_G$ ,  $X_B$  de los colores básicos que se visualizan directamente. El valor de la luminancia era el único utilizado por los antiguos televisores de blanco y negro.

El sistema de compresión desarrollado por el **JPEG** (*Joint Photographic Experts Group*) se fundamenta en el hecho de que el ojo humano es más sensible a los cambios espaciales del brillo que del color (crominancias), de forma que se almacena el brillo de cada punto de imagen, pero sólo la media de las dos crominancias obtenida sobre cuatro o más píxeles adyacentes. De esta forma, por ejemplo, en vez de almacenar por cada 4 puntos  $4 \cdot 3 = 12$  valores se almacena  $4 + 2 = 6$  valores, obteniéndose así una reducción del 50 por 100 en la capacidad necesaria para almacenar la imagen. En la compresión JPEG, el sistema descrito anteriormente se suele utilizar combinadamente con los otros métodos básicos de compresión descritos anteriormente.

**Compresión MPEG (imágenes).** El **MPEG** (*Motion Picture Experts Group*) ha definido un estándar ISO para imágenes en movimiento. Se fundamenta en el sistema JPEG, pero además una imagen dentro de una secuencia se representa con tan sólo los *cambios* con respecto a la imagen anterior.

**Compresión MP3 (sonido).** Según se puso de manifiesto en el Sección 2.2 también los archivos de sonido ocupan gran capacidad: en principio la grabación de un minuto de música con calidad CD ocupa unos 10 MBytes. Una de las técnicas más usadas para compresión de sonido se denomina MP3 (MPEG-1 Nivel 3 de audio) y es un estándar ISO. La idea básica consiste en descomponer mediante filtros la señal de audio original en diversos (32, por ejemplo) canales de frecuencias y codificar la información de cada canal con un número de bits dependiente de la amplitud de la señal. Con este sistema se obtienen porcentajes de compresión del 92 por 100 (compresión 12:1). Las señales de voz o música se graban o transmiten en formato MP3, ocupando un espacio reducido, y el sistema de reproducción de sonido contiene un procesador (circuito integrado) que descomprime la señal en tiempo real para poder escucharla adecuadamente.

## 2.7. Conclusión

En este capítulo se han abordado las técnicas de representación de diversas formas de información: textos (Sección 2.1), sonidos (Sección 2.2), imágenes (Sección 2.3) y datos numéricos (Sección 2.4), tanto enteros como reales.

Directamente relacionados con la representación de la información se encuentran los conceptos de detección automática de errores en almacenamiento o transmisión, y de compresión de datos. Las dos últimas secciones de este capítulo han hecho una introducción a ellos.



### Test

**T2.1.** Para codificar 1.320 símbolos distintos se necesitan, como mínimo:

- a) 11 bits.
- b) 10 bits.
- c) 12 bits.
- d) 1.320 bits.

**T2.2.** El código EBCDIC utiliza para codificar cada símbolo:

- a) 4 bits.
- b) 6 bits.

- c) 7 bits.
- d) 8 bits.

**T2.3.** El código ASCII básico (ANSI-X3.4) utiliza para codificar cada símbolo:

- a) 16 bits.
- b) 6 bits.
- c) 7 bits.
- d) 8 bits.

**T2.4.** La familia de normalizaciones ISO 8859 (ampliaciones de ASCII) utiliza para codificar cada símbolo:

- a) 16 bits.
- b) 6 bits.
- c) 7 bits.
- d) 8 bits.

**T2.5.** El sistema Unicode utiliza para codificar cada símbolo:

- a) 16 bits.
- b) 6 bits.
- c) 7 bits.
- d) 8 bits.

**T2.6.** En Unicode los caracteres ASCII, Latín-1, se codifican con los códigos comprendidos entre:

- a) 1000 0000 y 1000 00FF.
- b) 0000 0000 y 0000 00FF.
- c) 0000 2000 y 0000 20FF.
- d) Están pendientes de asignar.

**T2.7.** Para almacenar 3 minutos de una señal de sonido con calidad de audio (8 bits/muestra; 22,05 KHz) se necesitan:

- a) 3,79 Mbytes.
- b) 30,28 Mbytes.
- c) 3,96 Mbytes.
- d) Ninguna de las contestaciones anteriores es correcta.

**T2.8.** ¿Cuál de las siguientes afirmaciones es correcta?

- a) GIF es un formato para codificar imágenes en forma de mapa de bits.
- b) JPEG es un formato para codificar imágenes en forma de mapa de vectores.
- c) TIFF es un formato para codificar imágenes en forma de mapa de vectores.
- d) DXF es un formato para codificar imágenes en forma de mapa de bits.

**T2.9.** ¿Cuál de las siguientes afirmaciones es correcta?

- a) EPS es un formato para codificar imágenes en forma de mapa de bits.
- b) JPEG es un formato para codificar imágenes en forma de mapa de vectores.
- c) TIFF es un formato para codificar imágenes en forma de mapa de bits.
- d) DXF es un formato para codificar imágenes en forma de mapa de bits.

**T2.10.** La resolución de un monitor SVGA es de:

- a)  $640 \times 480$  elementos de imagen.
- b)  $720 \times 480$  elementos de imagen.
- c)  $800 \times 600$  elementos de imagen.
- d)  $1.024 \times 768$  elementos de imagen.

**T2.11.** La resolución de un monitor XGA es de:

- a)  $640 \times 480$  elementos de imagen.
- b)  $720 \times 480$  elementos de imagen.

- c)  $800 \times 600$  elementos de imagen.
- d)  $1.024 \times 768$  elementos de imagen.

**T2.12.** La capacidad de memoria que ocupará una imagen sin comprimir con 16 niveles de color y resolución de  $800 \times 600$  elementos de imagen, aproximadamente es:

- a) 234,375 KBytes.
- b) 468,75 KBytes.
- c) 937,5 KBytes.
- d) 7,32 MBytes.

**T2.13.** La representación interna de un determinado dato es: A73C DF05, si el sistema utiliza el criterio del extremo menor, la memoria se direcciona por palabras de 16 bits y el dato se almacena a partir de la posición 37AB FFFF, el dato queda memorizado de la siguiente forma:

- a) C37A en la posición 37AB FFFF; y 50FD en la posición 37AC 0000.
- b) A73C en la posición 37AB FFFF; y DF05 en la posición 37AC 0000.
- c) 50FD en la posición 37AB FFFF; y C37A en la posición 37AC 0000.
- d) DF05 en la posición 37AB FFFF; y A73C en la posición 37AC 0000.

**T2.14.** La representación interna de un determinado dato es: A73C DF05, si el sistema utiliza el criterio del extremo mayor, la memoria se direcciona por palabras de 16 bits y el dato se almacena a partir de la posición 37AB FFFF, el dato queda memorizado de la siguiente forma:

- a) C37A en la posición 37AB FFFF; y 50FD en la posición 37AC 0000.
- b) A73C en la posición 37AB FFFF; y DF05 en la posición 37AC 0000.
- c) 50FD en la posición 37AB FFFF; y C37A en la posición 37AC 0000.
- d) DF05 en la posición 37AB FFFF; y A73C en la posición 37AC 0000.

**T2.15.** En Borland C++ se define un tipo entero de datos denominado "entero largo sin signo" que ocupa 32 bits. El número mayor de este tipo almacenable será:

- a) 32.767.
- b) 65.535.
- c)  $2.147.484.648$ .
- d)  $4.294.967.295$ .

**T2.16.** El número decimal  $-25$  en representación como dato de tipo entero, con signo y magnitud, con  $n = 8$  bits es:

- a) H'67.
- b) H'99.
- c) H'E6.
- d) H'E7.

**T2.17.** El número decimal  $-25$  en representación como dato de tipo entero, complemento a 1, con  $n = 8$  bits es:

- a) H'67.
- b) H'99.



- c) H'E6.
- d) H'E7.

**T2.18.** El número decimal -25 en representación como dato de tipo entero, complemento a 2, con  $n = 8$  bits es:

- a) H'67.
- b) H'99.
- c) H'E6.
- d) H'E7.

**T2.19.** El número decimal -25 en representación como dato de tipo entero, sesgado, con  $n = 8$  bits es:

- a) H'67.
- b) H'99.
- c) H'E6.
- d) H'E7.

**T2.20.** La representación de un número en el interior de un computador es A9, suponiendo que corresponde a un dato de tipo entero, sin signo, su valor decimal es:

- a) 169.
- b) -41.
- c) -86.
- d) -87.

**T2.21.** La representación de un número en el interior de un computador es A9, suponiendo que corresponde a un dato de tipo entero, en signo y magnitud, su valor decimal es:

- a) 169.
- b) -41.
- c) -86.
- d) -87.

**T2.22.** La representación de un número en el interior de un computador es A9, suponiendo que corresponde a un dato de tipo entero, complemento a 1, su valor decimal es:

- a) 169.
- b) -41.
- c) -86.
- d) -87.

**T2.23.** La representación de un número en el interior de un computador es A9, suponiendo que corresponde a un dato de tipo entero, en complemento a 2, su valor decimal es:

- a) 169.
- b) -41.
- c) -86.
- d) -87.

**T2.24.** La representación de un número en el interior de un computador es A9, suponiendo que corresponde a un dato de tipo entero, sesgado, su valor decimal es:

- a) 41.
- b) -41.
- c) -86.
- d) -87.

**T2.25.** La representación de un número en el interior de un computador es 1001 0111, suponiendo que corresponde a un dato de tipo entero, BCD, su valor decimal es:

- a) 151.
- b) -23.
- c) -104.
- d) 97.

**T2.26.** El número decimal 4325 en hexadecimal es:

- a) 0100 0011 0010 0101.
- b) 10E5.
- c) 0001 0000 1110 0101.
- d) Ninguna de las contestaciones anteriores es correcta.

**T2.27.** El número decimal 4325 en BCD es:

- a) 0100 0011 0010 0101.
- b) 10E5.
- c) 0001 0000 1110 0101.
- d) Ninguna de las contestaciones anteriores es correcta.

**T2.28.** ¿Cuál de las siguientes afirmaciones es correcta?

- a) En la representación de tipo entero pueden representarse datos enteros por pequeños que éstos sean (no pueden existir agotamiento, *underflow*).
- b) Con un mismo número de bits pueden almacenarse datos con mayor precisión con una representación de tipo real que de tipo entero.
- c) El redondeo de la normalización IEEE se hace siempre de forma tal que el bit menos significativo sea 0 ("redondeo al par").
- d) En la representación IEEE 754 de números reales el exponente se almacena como entero sin signo.

**T2.29.** Suponiendo que se utilizasen 4 bits para almacenar la parte fraccionaria de la mantisa normalizada, y el redondeo se hiciese como el estándar IEEE 754, ¿cuál es el resultado correcto de redondear el número 1,100110?

- a) 1,1001.
- b) 1,1010.
- c) 1,1000.
- d) Ninguna de las respuestas anteriores es correcta.

**T2.30.** Suponiendo que se utilizasen 4 bits para almacenar la parte fraccionaria de la mantisa normalizada, y el redondeo se hiciese como el estándar IEEE 754, ¿cuál es el resultado correcto de redondear el número 1,100111?

- a) 1,1001.
- b) 1,1010.
- c) 1,1000.
- d) Ninguna de las respuestas anteriores es correcta.

**T2.31.** Suponiendo que se utilizasen 4 bits para almacenar la parte fraccionaria de la mantisa normalizada, y el redondeo se hiciese como el estándar IEEE 754, ¿cuál es el resultado correcto de redondear el número 1,100101?

- a) 1,1001.
- b) 1,1010.

- c) 1,1000.  
d) Ninguna de las respuestas anteriores es correcta.

**T2.32.** El patrón de bits de un dato de tipo real en simple precisión, normalización IEEE754, es: FFFF FFFF; el dato representado es:

- a) Un valor indeterminado.  
b) Más infinito.  
c) Menos infinito.  
d)  $-1,1111\dots111)_2 \cdot 2^{128}$ .

**T2.33.** El patrón de bits de un dato de tipo real en simple precisión, normalización IEEE754, es: FF80 0000; el dato representado es:

- a) Un valor indeterminado.  
b) Más infinito.  
c) Menos infinito.  
d)  $-2^{128}$ .

**T2.34.** El patrón de bits de un dato de tipo real en simple precisión, normalización IEEE754, es: 0000 0001<sub>H</sub>; el dato representado es:

- a)  $2^{-23}$ .  
b) Cero.  
c)  $2^{-32}$ .  
d)  $2^{-149}$ .

**T2.35.** ¿Cuál sería la eficiencia que se obtendría codificando los símbolos EBCDIC si se codificasen con el sistema Unicode?

- a) 0,0039.  
b) 99,6 por 100.  
c) 256.  
d) 0,4 por 100.

**T2.36.** Los datos siguientes contienen un bit de paridad, según el criterio impar. ¿Cuál de ellos es incorrecto?

- a) H'EA1.  
b) H'EA5.

- c) H'FA1.  
d) H'EA0.

**T2.37.** Cada uno de los datos que se dan en las siguientes respuestas contienen un bit de paridad, y sólo uno de ellos es correcto. ¿Cuál de ellos es?

- a) 1001 0111.  
b) 0110 1110.  
c) 0110 1110.  
d) 0110 1100.

**T2.38.** La secuencia generada por el algoritmo de compresión LZ77 para la sucesión de bits: 100010011000010010 es:

- a) (0,0,1)(0,0,0)(1,1,0)(3,3,0)(2,1,1)(1,1,0).  
b) (0,0,1)(0,0,0)(1,1,0)(4,3,1)(8,4,0)(9,4,0).  
c) (0,0,1)(1,0,0)(2,1,1)(3,2,1)(4,4,3)(7,3,1)(8,8,4)(12,9,5)(17,2,1).  
d) (0,0,1)(0,0,0)(0,0,0)(4,2,0)(3,1,1)(4,2,0)(1,1,1)(3,3,0).

**T2.39.** La descompresión de la siguiente secuencia generada por el algoritmo LZ77:

(0,0,0)(0,0,1)(1,1,0)(4,3,1)(8,7,0)(15,9,0)

es:

- a) 01001000100100000100100000100100.  
b) 0110111000000001111111110.  
c) 01100111011001101100111010.  
d) Ninguna de las contestaciones anteriores es correcta.

**T2.40.** Un fichero contiene los atributos de color (RGB) de cada uno de los píxeles de una imagen utilizando 256 niveles para cada color básico. Si se utiliza una paleta de 4.096 colores para realizar una compresión GIF, ¿qué factor de compresión se obtendrá?:

- a) 16 por 100.  
b) 33,33 por 100.  
c) 50 por 100.  
d) 6,2 por 100.

## Problemas resueltos



### REPRESENTACIÓN DE TEXTOS

**P2.1.** ¿Cuántos bits se necesitarían como mínimo para codificar un conjunto de 108 caracteres?

SOLUCIÓN

$$2^n \geq 108 \rightarrow n \cdot \log 2 \geq \log 108 \rightarrow n \geq 6,75$$

Luego necesitamos  $n = 7$  bits. En efecto,  $2^7 = 128 > 108$  y con 6 bits sería insuficiente ya que  $2^6 = 64 < 108$ .

**P2.2.** Codificar en EBCDIC el siguiente texto: JUAN LLOR C/ PRIM.

SOLUCIÓN

De acuerdo con la tabla de códigos EBCDIC se tiene que:

Carácter	EBCDIC	
	Hexadecimal	Binario
J	D1	11010001
U	E4	11100100
A	C1	11000001
N	D5	11010101
SP	40	01000000
L	D3	11010011
L	D3	11010011
O	D6	11010110
R	D9	11011001
SP	40	01000000
C	C3	11000011
/	61	01100001
P	D7	11010111
R	D9	11011001
I	C9	11001001
M	D5	11010101

donde SP representa al espacio en blanco.

**P2.3.** Transformar a códigos ASCII (ANSI-X3.4), Latín-1, EBCDIC y Unicode los siguientes mensajes:

- a) 37,45.  
b) 7784 BJC.

SOLUCIÓN

Consultando las tablas del Apéndice 2, y teniendo en cuenta que ASCII (ANSI-X3.4) utiliza 7 bits, Latín-1 8 bits, EBCDIC 8 bits y Unicode 16 bits, se tiene:

- a) 37,45

ASCII: 

3	7	,	4	5
33	37	2C	34	35

 → 0110011 0110111 0101100 0110100 0110101

EBCDIC: 

3	7	,	4	5
F3	F7	6B	F4	F5

 → 11110011 11110111 0110111 11110100 11110101

Latín-1: 

3	7	,	4	5
33	37	2C	34	35

 → 00110011 00110111 00101100 00110100 00110101

Unicode: 

3	7	,	4	5
0033	0037	002C	0034	0035

  
→ 0000000000110011 0000000000110111 0000000000101100 0000000000110100 0000000000110101

## REPRESENTACIÓN DE SONIDOS

**P2.4.** Obtener la capacidad necesaria para almacenar 1 minuto de una señal de audio estereofónico con calidad CD.

SOLUCIÓN

Según se indica en la Tabla 2.3, el número de muestras a captar en un minuto es:

$$N = F_s \cdot t = 44.100 \frac{\text{muestras}}{\text{segundo}} \cdot 60 \text{ segundos} = 2,646.000 \text{ muestras}$$

Como por cada muestra se representa con 16 bits = 2 Bytes, el número de bytes a almacenar por canal será:

$$C_{\text{canal}} = 2,646.000 \text{ muestras} \cdot 2 \frac{\text{Bytes}}{\text{muestra}} = 5,292.000 \text{ Bytes} \approx 5 \frac{\text{MB}}{\text{canal}}$$

Pero, al ser una grabación en estereofónico, se necesita el doble de muestras:

$$C_{\text{total}} \approx 5 \frac{\text{MB}}{\text{canal}} \cdot 2 \text{ canales} = 10 \text{ MB}$$

## REPRESENTACIÓN DE IMÁGENES

**P2.5.** Obtener la capacidad de memoria que ocupará una imagen en blanco y negro con una resolución de  $640 \times 350$  elementos de imagen y con 16 niveles de grises.

## SOLUCIÓN

Para codificar el atributo (nivel de gris) se necesitan 4 bits, ya que  $2^4 = 16$ , y en total hay que almacenar el atributo de  $640 \cdot 350 = 22.400$  elementos. Con lo que la capacidad total será:

$$C = 22.400 \cdot 4 = 896.000 \text{ bits} = 109,375 \text{ KBytes}$$

## REPRESENTACIÓN DE DATOS NÚMERICOS

**P2.6.** Suponiendo un computador con longitud de palabra  $n = 8$  bits y que utiliza representación en signo y magnitud, ¿cómo se representarían internamente los siguientes números enteros: 65; -37; +84; -21?

## SOLUCIÓN

$$\begin{array}{ll} 65 \rightarrow 01000001 & -37 \rightarrow 11011011 \\ 84 \rightarrow 01010100 & -21 \rightarrow 11101011 \end{array}$$

**P2.7.** Suponiendo un computador con longitud de palabra  $n = 8$  bits y que utiliza representación en complemento a 2, ¿cómo se representarían internamente los siguientes números enteros: 65; -37; +84; -21?

## SOLUCIÓN

$$\begin{array}{ll} 65 \rightarrow 01000001 & -37 \rightarrow 11011011 \\ 84 \rightarrow 01010100 & -21 \rightarrow 11101011 \end{array}$$

**P2.8.** ¿Cuáles serían los números decimales enteros correspondientes a los números:

$$1001 \ 1000; \quad 0111 \ 1001$$

suponiendo las representaciones que se indican a continuación?:

- a) Sin signo.
- b) Signo y magnitud.
- c) Complemento a 1.
- d) Complemento a 2.
- e) Sesgada.
- f) BCD.

## SOLUCIÓN

a) Sin signo:

$$\begin{array}{l} 1001 \ 1000)_2 = 98)_{16} = 9 \times 16^1 + 8 \times 16^0 = 152)_{10} \\ 0111 \ 1001)_2 = 79)_{16} = 7 \times 16^1 + 9 \times 16^0 = 121)_{10} \end{array}$$

b) Signo y magnitud.

$$\begin{array}{l} \text{El primer bit es 1, signo negativo} \\ 0001 \ 1000)_2 = -18)_{16} = 1 \times 16^1 + 8 \times 16^0 = -24)_{10} \\ \text{El primer bit es 0, signo positivo} \\ 0111 \ 1001)_2 = 79)_{16} = 7 \times 16^1 + 9 \times 16^0 = 121)_{10} \end{array}$$

c) Complemento a 1.

$$\begin{array}{l} \text{Como empieza por 1 es negativo} \\ 0001 \ 1000)_2 = -(0110 \ 0111) = -67)_{16} = -6 \times 16^1 + 7 \times 16^0 = -103)_{10} \\ \text{El primer bit es 0, signo positivo} \\ 0111 \ 1001)_2 = 79)_{16} = 7 \times 16^1 + 9 \times 16^0 = 121)_{10} \end{array}$$

d) Complemento a 2.

$$\begin{array}{l} \text{Como empieza por 1 es negativo} \\ 0001 \ 1000)_2 = -(0110 \ 0111 + 1) = -(0110 \ 1000) = -68)_{16} = -6 \times 16^1 + 8 \times 16^0 = -104)_{10} \\ \text{El primer bit es 0, signo positivo} \\ 0111 \ 1001)_2 = 79)_{16} = 7 \times 16^1 + 9 \times 16^0 = 121)_{10} \end{array}$$

e) Sesgada.

$$N_s = S + N = 1000\ 0000 + N = 1001\ 1000$$

$$N = 1001\ 1000 - 1000\ 0000 = 0001\ 1000_2 = 18_{10} = 1 \times 16^1 + 8 \times 16^0 = 24_{10}$$

$$N_s = S + N = 1000\ 0000 + N = 0111\ 1001$$

$$N = 0111\ 1001 - 1000\ 0000 = -0001\ 1001_2 = -(1000\ 0000 - 0111\ 1001)_{16} = -0000\ 1011_{16} = -0B_{16} = 0 \times 16^1 + 11 \times 16^0 = -11_{10}$$

$$\begin{array}{r} 1000\ 0000 \\ - 0111\ 1001 \\ \hline 0000\ 1011 \end{array}$$

**P2.9.** Obtener la representación interna en simple precisión según las especificaciones IEEE754 del número:

$$N = 37,485 \times 10^{-17}$$

(redondear y dar el resultado en hexadecimal empaquetado).

SOLUCIÓN

$$N = 37,485 \times 10^{-17} = 0,37485 \times 10^{-15}$$

$$10^{-15} = 2^x \Rightarrow$$

$$x = \frac{15}{\log_{10} 2} = 49,828921$$

$$\begin{aligned} N &= (0,37485 \times 2^{-0,828921}) \times 2^{-49} = 0,2110217 \times 2^{-49} = (0,360585D0E)_H \times 2^{-49} = \\ &= 0,0011\ 0110\ 0000\ 0101\ 1000\ 0101\ 1101\ 0000\ 1110 \times 2^{-49} = \\ &= 1,1011\ 0000\ 0101\ 1000\ 0001\ 1101\ 0000\ 1110 \times 2^{-52} \end{aligned}$$

(23) +0 ←

Como los bits de guarda son 10 y la última cifra significativa a almacenar es 0 la mantisa se trunca. El exponente a almacenar será:

$$e = 127 - 52 = 75_{10} = 113_8 = 0100\ 1011$$

$$s = 0 \text{ ya que } N > 0$$

0010	0101	1101	1000	0010	1100	0000	1110
2	5	D	8	2	C	0	E

Es decir, el resultado es: 25D81617

**P2.10.** Obtener la representación interna del número  $3754,8976 \times 10^8$  en notación IEEE754.

SOLUCIÓN

Haciendo la transformación  $10^8 = 2^y$  tenemos que:

$$y = 26,57542476$$

De donde  $N = 5595,233444 \times 2^{26}$ . Pasando a binario  $N$ :

$$N = 1010111011011,001110111100 \times 2^{26}$$

Normalizando  $N$ , nos queda:

$$N = 1,010111011011001110111100 \times 2^{38}$$

Calculamos el exponente a almacenar:

$$\begin{aligned} e &= E + S = 38 + 127 = 165 \quad (S = 2^{ne-1}) \\ e &= 10100101_2 \end{aligned}$$

Por tanto la representación interna sería:

$$0\ 10100101\ 010111011011001110111100$$

O bien, en hexadecimal,  $N \rightarrow 52AED9DE$ .

- P2.11.** En la representación IEEE754 simple, obtener los números mayores (finitos) y más próximos a cero que pueden representarse. Calcular el valor en decimal.

**SOLUCIÓN**

La mayor mantisa que se puede obtener es:

$$M_{\text{máx}} = 1.1\dots\dots 1 = (1 - 2^{-23}) = 2 - 2^{-23}$$

El mayor exponente almacenado representando un número normalizado es:

$$e = 1111\ 1110)_2 = 2^9 - 2 = 256 - 2 = 254$$

luego el mayor número es:

$$a = N_{\text{mayor}} = (1 - 2^{-23}) \times 2^{254-252} = (2 - 2^{-23}) \cdot 2^{127} \approx 3,4 \cdot 10^{38}$$

El número positivo más próximo a 0 (es decir, el de menor valor absoluto) será:

- a) Sin normalizar:

$$\begin{aligned} M_{\text{menor}} &= 0.0\dots\dots 1 = 2^{-23} \\ e_{\text{menor}} &= 0000\ 0000)_2 = 0_{10} \\ N_{\text{menor}} &= 2^{-23} \cdot 2^{0-126} = 2^{-149} \approx 1,4 \cdot 10^{-45} \end{aligned}$$

- b) Normalizado:

$$\begin{aligned} M'_{\text{menor}} &= 1.0\dots\dots 0 = 1_{10} \\ e'_{\text{menor}} &= 0000\ 0001)_2 = 1_{10} \\ N'_{\text{menor}} &= 1 \cdot 2^{1-127} = 2^{-126} \approx 1,18 \cdot 10^{-38} \end{aligned}$$

Una representación de estos valores puede verse en la Figura 2.6.

- P2.12.** Para el número  $N = -37$  obtener su representación interna con  $n = 32$  bits, como:

- a) Dato de tipo entero en complemento a 2.  
b) Dato de tipo real IEEE 754.

**SOLUCIÓN**

- a)  $N = -37$

$$\begin{array}{r|l} 37 & 16 \\ \hline 5 & 2 \end{array}$$

$$37)_{10} = 25)_{10} = 0010\ 0101)_2$$

$$C_1(N) = 111\dots 1011010$$

$$C_2(N) = 11\dots 11011011 = \text{FFFF FFDB}$$

Solución (en HEX): FFFF FFDB

- b)  $N = -100101 = -1,00101 \times 2^5$   
 $E = 127 + 5 = 132$

$$\begin{array}{r|l} 132 & 16 \\ \hline 4 & 8 \end{array}$$

$$132)_{10} = 84)_{10} = 10000\ 0100)_2$$

$$\begin{array}{cccccc} 1100 & 0010 & 0001 & 0100 & \dots 0 \\ \text{C} & 2 & 1 & 4 & \end{array}$$

Solución (en HEX): C214 0000

- P2.13.** Un computador utiliza Unicode como código de entrada/salida. Un programador teclea la siguiente sentencia:

$$A = -37;$$

- a) ¿Cómo se almacena dicha sentencia (código que se genera)?  
b) ¿Qué valor binario de  $A$  se obtendría al compilar el programa si el programador hubiese definido  $A$  como un entero largo (32 bits, complemento a dos)?  
c) ¿Qué valor binario de  $A$  se obtendría al compilar el programa si el programador hubiese definido  $A$  real coma flotante (simple precisión, IEEE754)?

## SOLUCIÓN

- a) Las 256 primeras combinaciones Unicode corresponden a los caracteres Latín-1, anteponiéndoles 8 ceros. Consultando una tabla ASCII/Latín-1, tenemos que: A es; = es; - es; 3 es y 7 es; con lo que en Unicode los códigos que se generan en binario son los que se indican en la tercera fila de la tabla que se da a continuación.

	A	=	-	3	7
	0041	003D	002D	0033	0037
Unicode →	0000 0000 0100 0001	0000 0000 0011 1101	0000 0000 0010 1101	0000 0000 0011 0011	0000 0000 0011 0011

- b)  $N = -37$

$$\begin{array}{r} 37 \overline{) 16} \\ 5 \quad 2 \end{array}$$

$$37)_{10} = 25)_{10} = 0010 \ 0101)_2$$

$$C_1(N) = 111 \dots 1011010$$

$$C_2(N) = 11 \dots 11011011 = \text{FFFF FFDB}$$

Solución (en HEX): FFFF FFDB

- c)  $N = -100101 = -1,00101 \times 2^5$   
 $E = 127 + 5 = 132$

$$\begin{array}{r} 132 \overline{) 16} \\ 4 \quad 8 \end{array}$$

$$132)_{10} = 84)_{10} = 10000 \ 0100)_2$$

$$\begin{array}{|c|c|c|c|c|} \hline 1100 & 0010 & 0001 & 0100 & \dots 0 \\ \hline C & 2 & 1 & 4 & \\ \hline \end{array}$$

Solución (en HEX): C214 0000.

**P2.14.** Sea el número  $A = -4$ . Obtener la representación que se obtendría del mismo como:

- a) Entero con  $n = 8$  bits, en complemento a 2.  
b) Entero con  $n = 8$  bits, en signo y magnitud.  
c) Entero con  $n = 8$  bits, sesgado.  
d) Real, IEEE754, en simple precisión.

## SOLUCIÓN

- a) Entero con  $n = 8$  bits, en complemento a 2.

El valor  $-4)_{10}$  es igual a  $2^2)_{10}$ , lo que en binario se expresa como  $00000100)_2$ . Para negarlo usando un complemento a 2 en un primer paso hay que aplicarle un complemento a 1 y luego sumar 1, con lo que conseguimos:

$$00000100)_2 = 11111011)_{C1} = 11111100)_{C2} = \text{FC})_{16}$$

- b) Entero con  $n = 8$  bits, en signo y magnitud.

La representación en signo y magnitud utiliza el bit más significativo para indicar el signo (0 para los positivos y 1 para los negativos) y el resto se los bits para representar el valor absoluto del número en binario natural, por tanto, el número  $-4)_{10}$  se representa como:

$$1 \ 0000100)_{SM} = 84)_{16}$$

- c) Entero con  $n = 8$  bits, sesgado.

La notación sesgada se basa en añadir un valor o sesgo a los valores de forma que el menor valor representable nota como  $00000000$  y el mayor como  $11111111$ . El sesgo para valores de 8 bits es  $2^7)_{10} = 10000000)_2$ , por tanto, el número representado en notación sesgada es:

$$\begin{array}{r} 10000000 \\ - 00000100 \\ \hline 01111100 = 7C_{16} \end{array}$$

d) Real, IEEE754, en simple precisión.

La notación IEEE754 representa a los números reales usando 32 bits con el siguiente formato:

$$\pm 1.m \times 2^e,$$

utilizando un bit para el signo (0 para los positivos y 1 para los negativos), 8 bits para el exponente (sesgado con  $S = 2^7 - 1$ ), y 23 bits para la mantisa. Así que lo primero que hay que hacer es colocar el número  $-4$ , con el formato anterior:

$$-4_{10} = -100_2 \times 2^0 = -1.0_2 \times 2^2$$

por tanto, los campos de signo, exponente y mantisa son los siguientes:

$$\begin{array}{ll} S &= 1 \\ E &= 2 + 2^7 - 1 = 00000010 + 01111111 = 10000001 \\ M &= 00000000000000000000000 \end{array}$$

con lo que el número representado en notación IEEE754 queda como

$$1\ 10000001\ 00000000000000000000000 = (C0800000)_{16}$$

**P2.15.** Dado el patrón de bits:

81C0 0000

qué valor representa suponiendo de que se tratase de un número:

- Entero sin signo.
- Entero, en signo y magnitud.
- Entero en complemento a uno.
- Entero en complemento a dos.
- Entero sesgado.
- Real en la normalización IEEE754.

#### SOLUCIÓN

Primero pasamos el patrón de bits de hexadecimal a binario:

	31	28	27	24	23	20	19	16	15	12	11	8	7	4	3	0
81C0 0000 →	1000		0001		1100		0000		0000		0000		0000		0000	

a) Entero sin signo:

$$N = 2^{22} + 2^{23} + 2^{24} + 2^{31} = 2,1768438 \times 10^9$$

b) Entero, en signo y magnitud:

$$N = -(2^{22} + 2^{23} + 2^{24}) = -29,360.128$$

c) Entero en complemento a uno.

El número representado es:

$$N = -(111\ 1110\ 0011\ 1111\ 1111\ 1111\ 1111\ 1111)_2$$

Para pasarlo a decimal tenemos que sumar los pesos de las posiciones en las que hay un uno. Podemos realizar menos cálculos teniendo en cuenta que  $N$  lo podemos expresar por medio de la resta de los siguientes números ( $A-B$ ):

$$\begin{array}{r} 10000\ 0000\ 0100\ 0000\ 0000\ 0000\ 0000\ 0000 \\ - 00000\ 0010\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001 \\ \hline 00111\ 1110\ 0011\ 1111\ 1111\ 1111\ 1111\ 1111 \end{array} \begin{array}{l} = A = 2^{31} + 2^{22} \\ = B = 2^{25} + 1 \\ = N = A - B \end{array}$$



Es decir:

$$N = -[(2^{31} + 2^{22}) - (2^{25} + 1)] = -2,1181235 \times 10^9$$

- a) Entero en complemento a dos:

$$C_2(N) = C_1(N) + 1; \text{ con lo que: } N = -2,1181235 \times 10^9 - 1 = -3,1181235 \times 10^9$$

- b) Entero sesgado.

Por definición, el número sesgado es:

$$N_s = N + S = N + 2^{n-1} = N + 2^{31}$$

donde  $n$  es el número de bits con el que se representa el número.

En consecuencia:

$$\begin{aligned} N = N_s - 2^{31} &= (1000\ 0001\ 1100\ 0000\ 0000\ 0000\ 0000\ 0000)_2 - 2^{31} = \\ &= 2,1768438 \times 10^9 - 2,1474836 \times 2^9 = \\ &= 2,1181235 \times 10^9 \end{aligned}$$

donde el valor decimal de  $N$  ya lo habíamos calculado en la solución del apartado a).

- c) Real en la normalización IEEE754.

El significado de los bits del patrón es el siguiente:

$$\begin{array}{ll} \text{signo:} & s = 1 \text{ (número negativo)} \\ \text{exponente sesgado:} & e = (0000\ 0011)_2 = 3 \\ \text{mantisa:} & M = (1,1)_2 = 1,5 \end{array}$$

El exponente almacenado es:  $E = e - S = 3 - 127 = -124$ .

Con lo que:

$$N = (-1)^s \cdot M \cdot 2^E = -1,5 \cdot 2^{-124} = -7,0529661 \times 10^{-38}$$

## DETECCIÓN DE ERRORES

- P2.16.** ¿Cuál será la eficiencia y redundancia de un sistema que utiliza 7 bits para codificar 108 caracteres?  
¿Cuáles serían éstas, si se utilizasen dos bits más de los necesarios?

### SOLUCIÓN

- a) Se utilizan 7 bits para codificar  $m = 108$  caracteres, pero con  $n = 7$  bits se pueden codificar  $m' = 2^7 = 128$ , caracteres con lo que la eficiencia es:

$$\tau = \frac{m}{m'} = \frac{108}{128} = 0,84$$

La redundancia del código es:

$$R = (1 - 0,84) \times 100 = 16\%$$

- b) Con 2 bits más,  $n = 9$  bits, el número de caracteres a codificar sigue siendo el mismo,  $m = 108$ , y el número de bits que se pueden codificar ahora es  $m' = 2^n = 2^9 = 512$ , con lo que:

$$\tau = \frac{108}{512} = 0,21$$

$$R = (1 - 0,21) \times 100 = 79\%$$

Obviamente al utilizar más bits de los necesarios disminuye la eficiencia y aumenta la redundancia.

- P2.17.** Obtener el bit de paridad, según el criterio impar, de los siguientes mensajes:

10010; 10000; 101111; 1010

## SOLUCIÓN

Mensaje inicial	Mensaje con bit de paridad (criterio impar)	
10010	1	10010
10000	0	10000
101111	0	101111
1010	1	1010
	bit de paridad ↑	

**P2.18.** Se desean codificar 1,328.345 elementos diferentes.

- Calcular el número mínimo de bits necesarios para realizar la codificación.
- Obtener la redundancia y eficiencia del código generado.

## SOLUCIÓN

- Para poder codificar 1328345 elementos necesitaremos un número de hilos capaz de representar esos elementos:

$$2^n \geq 1328345$$

Así pues:

$$n = \log_2 1328345 = 3,32 \cdot \log(1328345) = 20,34$$

Por tanto, el número de bits necesarios son 21.

- La eficiencia será:

$$\tau = \frac{n.^{\circ} \text{ de símbolos}}{2^{n.^{\circ} \text{ de bits}}} = \frac{1328345}{2^{21}} = 0,633$$

Redundancia:

$$R = (1 - \tau) \cdot 100 = (1 - 0,633) \cdot 100 = 36,659\%$$

## COMPRESIÓN DE DATOS

**P2.19.** En un archivo comprimido por medio de una codificación Lempel-Ziv LZ77 se tiene la siguiente secuencia:

(0,0,0) (0,0,1) (2,2,0) (1,1,0) (4,3,1) (11,11,1) (17,16,0)

¿Qué sucesión de bits se obtendrá al descomprimir esta secuencia (dar el resultado en hexadecimal)?

## SOLUCIÓN

Secuencia comprimida →	0,0,0;	0,0,1;	2,2,0;	1,1,0;	4,3,1;	11,11,1;	17,16,0
Secuencia descomprimida →	0	1	010	00	1001	010100010011	01001010100010010

Es decir, la secuencia en binario y en hexadecimal es:

$$0101\ 0001\ 0010\ 1010\ 0010\ 0110\ 1001\ 0101\ 0001\ 0010 \rightarrow 512A269512$$

**P2.20.** Obtener los códigos Lempel-Ziv LZ77 para la siguiente cadena de bits: 90089188C.

## SOLUCIÓN

La secuencia de bits es:

$$90089188C = 1001\ 0000\ 0000\ 1000\ 1001\ 0001\ 1000\ 1000\ 1100$$

Primero hacemos un grupo con el primer uno (grupo 0,0,1) de la cadena, luego con el primer cero (grupo 0,0,0); y después vamos buscando secuencias que hayan aparecido anteriormente en la cadena, y a la pareja (posición de la secuencia encontrada, longitud de la secuencia) le añadimos el siguiente bit:

Secuencia original →	1	0	01	000	0000	010001	00100011	00010001	100
Secuencia comprimida →	0,0,1	0,0,0	1,1,1	3,2,0	3,3,0	9,5,1	7,7,1	12,10,1	5,2,0

Es decir, la cadena comprimida es:

(0,0,1) (0,0,0) (1,1,1) (3,2,0) (3,3,0) (9,5,1) (7,7,1) (12,10,1) (5,2,0)



## Problemas propuestos

### REPRESENTACIÓN DE TEXTOS

**P2.21.** Una empresa dispone de 3.524 empleados.

- ¿Cuántas cifras decimales se necesitarían para dar un código numérico identificar a cada uno de los empleados?
- ¿Cuántos caracteres se necesitarían para dar un código con cifras decimales y letras (no incluir *ch*, *ll* ni *ñ*) que identifique a cada uno de los empleados?
- ¿Cuántos bits se necesitarían para dar un código identificativo de cada empleado?
- ¿Cuántos empleados más pueden darse de alta en la empresa, sin necesidad de ampliar el número de bits para cada código?

**P2.22.** Teniendo en cuenta que en la Unión Europea las matrículas de coches se forman con 4 cifras decimales y tres letras del alfabeto, excluyendo las vocales y las letras *ch*, *ll* y *ñ*, indicar cuál es el número máximo de coches que pueden estar matriculados en un momento dado por este sistema.

**P2.23.** A qué texto corresponde el siguiente patrón de bits:

004D006100F100610020006C006C006F0076006500720061

suponiendo que estuviese codificado en:

- EBCDIC.
- Latín-1 ampliado.
- Unicode.

### REPRESENTACIÓN DE SONIDOS

**P2.24.** ¿Qué tiempo de música en calidad CD estéreo y sin comprimir se puede almacenar en un CD-ROM de 650 MB? ¿Y en un DVD de 4,7 GB?

**P2.25.** En un sistema de audio tetrafónico se muestrean las señales de sonido a una frecuencia de 44,1 KHz, y cada muestra se representa por uno entre 256 niveles posibles. Obtener:

- El ancho de banda mínimo que necesita el bus que lleva la información de sonido (suponer que la información de los cuatro canales se transmite por un único bus).
- La capacidad de memoria necesaria para almacenar una hora de audio.

### REPRESENTACIÓN DE IMÁGENES

**P2.26.** Suponiendo que en una pantalla se representan caracteres Unicode en una distribución de 80 columnas por 25 filas, y que cada carácter puede tener uno entre 256 colores, obtener el número de bytes total para almacenar una imagen de pantalla.

**P2.27.** Una pantalla de televisión está compuesta de  $720 \times 480$  píxeles, y cada uno de ellos tiene un color elegido en una paleta compuesta de 256 colores. Por otra parte la imagen se actualiza a una velocidad de 30 imágenes/segundo, para dar sensación de movimiento. Determinar:

- La capacidad de memoria necesaria para almacenar una imagen.
- El ancho de banda mínimo que necesita el bus que lleva la información de imagen a la pantalla.

**P2.28.** Cuántas imágenes BMP (sin compresión) caben en un CD de 600 MB, suponiendo:

- Resolución VGA.
- Resolución SVGA.
- Resolución XGA.

**P2.29.** Cuántas imágenes BMP (sin compresión) caben en un DVD de 4,7 GB, suponiendo:

- Resolución VGA.
- Resolución SVGA.
- Resolución XGA.

**P2.30.** Qué tiempo se puede almacenar en un CD de 600 MB:

- a) De imágenes de televisión (sin compresión).
- b) De imágenes de HDTV (sin compresión).

**P2.31.** Qué tiempo se puede almacenar en un DVD de 17 GB:

- a) De imágenes de televisión.
- b) De imágenes de HDTV.

## REPRESENTACIÓN DE DATOS NÚMERICOS

**P2.32.** Suponiendo un computador con longitud de palabra  $n = 8$  bits y que utiliza representación en complemento a 1, ¿cómo se representarían internamente los siguientes números enteros: 65; -37; +84; -21?

**P2.33.** Suponiendo un computador con longitud de palabra  $n = 8$  bits y que utiliza representación sesgada, ¿cómo se representarían internamente los siguientes números enteros: 65; -37; +84; -21?

**P2.34.** Suponiendo un computador de longitud de palabra de 16 bits y que utiliza el criterio del extremo menor, ¿cómo se almacenaría en memoria el dato 5,454.327 al utilizar las representaciones que se indican a continuación?:

- a) Sin signo.
- b) Signo y magnitud.
- c) Complemento a 1.
- d) Complemento a 2.
- e) Sesgada.
- f) BCD.

**P2.35.** Suponiendo un computador que utiliza el estándar IEEE754, obtener las representaciones internas en simple precisión de los siguientes números:

425372,27;  $-37 \cdot 48$ ;  $-25,38 \cdot 10^{-17}$ ; 7;  $1,5 \cdot 10^{-40}$

**P2.36.** En un programa se define el número 3 como número entero ( $J = 3$ ), y como número real en simple precisión ( $X = 3,0$ ). Suponiendo que en los dos casos los datos se almacenan con 32 bits, y que los enteros se representan en complemento a dos y los reales en la notación IEEE754, obtener las representaciones para los valores de  $J$  y  $X$ . ¿Cuál de las dos representaciones es más precisa?

**P2.37.** ¿Qué valores decimales corresponden a los siguientes datos de tipo real, simple precisión IEEE754?

- a) 1000 0111 1111 1111 1111 1111 0011 0011.
- b) 0111 0000 0000 0000 0000 0000 1110 0100.

**P2.38.** En IEEE754, precisión sencilla, se tienen los datos que se indican a continuación. ¿A que números decimales corresponden?

- a) BC7E 0000.
- b) 3754 3000.
- c) 00A0 0000.
- d) 8000 0004.

- e) FF80 0000.
- f) 7FF8 0700.

**P2.39.** Un computador de longitud de palabra de memoria de 16 bits, criterio extremo mayor, utiliza el código Unicode para entrada/salida de los datos. Suponiendo que con el editor de C se introduce el dato  $X = 3,2$ :

- a) ¿Cómo queda inicialmente almacenado ese dato en la memoria?
- b) Suponiendo que  $X$  se define como dato en simple precisión y se utiliza la notación IEEE754, ¿cómo lo almacenará en memoria el compilador de C?

**P2.40.** En un computador que utiliza el código ASCII Latin-1 se teclea el siguiente dato: 183.27E-17.

- a) ¿Qué códigos se generan?
- b) El dato anterior se utiliza en un programa como número real en precisión sencilla. ¿Qué representación interna se obtendrá suponiendo que el computador utiliza la notación IEEE754? Indicar el resultado en hexadecimal.

**P2.41.** Obtener los números reales máximos y mínimos representables en IEEE754, doble precisión.

## DETECCIÓN DE ERRORES

**P2.42.** En el interior de un chip de memoria se leen los datos que se dan a continuación, que incluyen un bit de paridad (criterio par):

```
1 1001 0010 0011 1001;
0 1110 0011 0011 1111;
0 1111 0011 1100;
0 1011 0000 1111 0101
```

Indicar cuáles de los datos anteriores se han grabado o leído erróneamente.

**P2.43.** Un computador recibe de un terminal los siguientes caracteres ASCII, que contienen un bit de paridad (criterio impar):

7A; 5C; 47; CA; 7C; C8

Indicar cuáles de estos caracteres deben ser rechazados por ser erróneos.

## COMPRESIÓN DE DATOS

**P2.44.** Dado el mensaje: BCCACBBCCACCA:

- a) Codificarlo utilizando el siguiente código:  $A \rightarrow 00$ ,  $B \rightarrow 01$ ,  $C \rightarrow 10$ .
- b) Codificarlo utilizando el siguiente código, dependiente de la frecuencia:  $A \rightarrow 11$ ,  $B \rightarrow 10$ ,  $C \rightarrow 0$ .
- c) Obtener el factor de compresión obtenido con el procedimiento b) con respecto al a).

**P2.45.** Decodificar el mensaje que se da a continuación, usando el siguiente código, dependiente de la frecuencia:  $A \rightarrow 10$ ,  $B \rightarrow 11$ ,  $C \rightarrow 0$

1011001011011001100100110010

(Obsérvese que no es necesario ningún carácter separador entre los códigos de cada letra, no produciéndose ninguna ambigüedad. Este tipo de codificación se denomina codificación de Huffman.)

**P2.46.** Un sistema utiliza ASCII de 7 bits para representar los textos, añadiendo al final de cada código un bit de paridad (criterio par).

- a) Indicar en hexadecimal la representación de la cadena: Granada 11.
- b) Comprimir los bits de la cadena resultante según el algoritmo Lempel-Ziv LZ77.

**P2.47.** ¿Qué tamaño ocuparía un archivo de imagen de 2,5 MBytes si se utilizase un algoritmo básico compresión JPEG?

**P2.48.** ¿Qué tamaño ocuparía un archivo de sonido de 2,5 MBytes si se utilizase un algoritmo básico compresión MP3?

**P2.49.** ¿Qué factor de compresión habría que utilizar para grabar *Rigoletto* de Giuseppe Verdi en un disquete de 1,44 MBytes?; ¿y en un CD de 600 MBytes?, ¿y en dos CD? (*Ayuda:* La versión a grabar de la ópera indicada tiene una duración de 1 hora 55 minutos.)

# El nivel de lógica digital

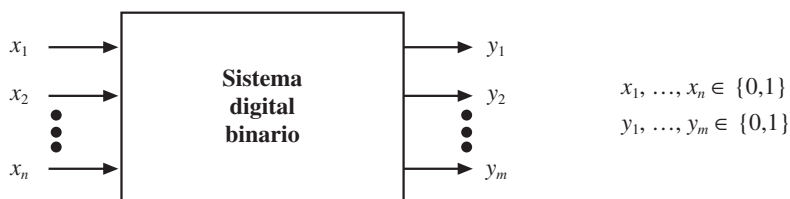
Las unidades funcionales centrales de un computador (unidad de control, unidad de tratamiento o camino de datos y memoria principal) están constituidos por módulos tales como codificadores, decodificadores, registros, multiplexores, circuitos aritméticos y puertas lógicas. El estudio y diseño de estos elementos se realiza dentro de un nivel conceptual denominado **nivel de lógica digital** que constituye el núcleo de la tecnología de computadores. El nivel de lógica digital, en definitiva, estudia los elementos necesarios para el diseño de procesadores y otros subsistemas del nivel de micro-máquina (Sección 1.5), que los utilizaremos en los próximos capítulos para describir la estructura de procesadores como el CODE-2.

Los sistemas digitales, objeto de este capítulo, se caracterizan por utilizar (transformar o memorizar) señales discretas; es decir, señales que sólo pueden tomar valores discretos, de entre una serie de niveles preestablecidos. En Electrónica tradicionalmente se han utilizado más los **sistemas analógicos**, en los que las variables y señales pueden tomar cualquier valor comprendido dentro de un rango determinado. Debido a una serie de cualidades inherentes a los sistemas digitales, como son el funcionamiento con una precisión mayor y controlada, los computadores se implementan como sistemas digitales y poco a poco los equipos de telecomunicaciones y distintos productos de la electrónica de consumo están siendo sustituidos por realizaciones digitales.

## 3.1. Sistemas combinacionales

Un **sistema digital** es cualquier dispositivo o equipo de procesamiento, almacenamiento o transmisión de información en el que ésta se da mediante magnitudes físicas que sólo pueden tomar valores discretos. Un **sistema binario** es un sistema digital en el que sólo se utilizan dos valores discretos. Los dos valores discretos, que suelen corresponder a magnitudes físicas tales como tensión o corriente eléctrica, polos de magnetización (Norte o Sur), dos niveles de intensidad de luz, etc., se representan con las letras *L* (Low) y *H* (High), o como *cero* (0) y *uno* (1). En este capítulo utilizaremos 0 y 1.

En la Figura 3.1 se representa esquemáticamente un sistema digital binario. Como todo sistema tiene unas entradas y unas salidas; en la figura se incluyen  $n$  entradas,  $x_1, x_2, \dots, x_n$ , y  $m$  salidas,  $y_1, y_2, \dots, y_m$ . Según la definición de sistema binario, los valores tanto de las entradas como de las salidas sólo pueden tomar valores del conjunto  $\{0,1\}$ . El sistema funciona de forma tal que para cada combinación de entradas genera una combinación de valores de salida. Al ser el sistema binario, el número de combinaciones distintas de entrada será  $2^n$ .



**Figura 3.1.** Esquema simplificado de un sistema binario.

Existen dos tipos de sistemas: combinacionales, que son objeto de estudio de la presente sección (Sección 3.1), y secuenciales, que serán analizados en la Sección 3.2. Un **sistema combinacional** es aquel en el que los valores lógicos de las salidas, en un instante dado están determinados por los valores que en ese momento tengan las entradas, salvo un pequeño retraso inherente al propio circuito o sistema.

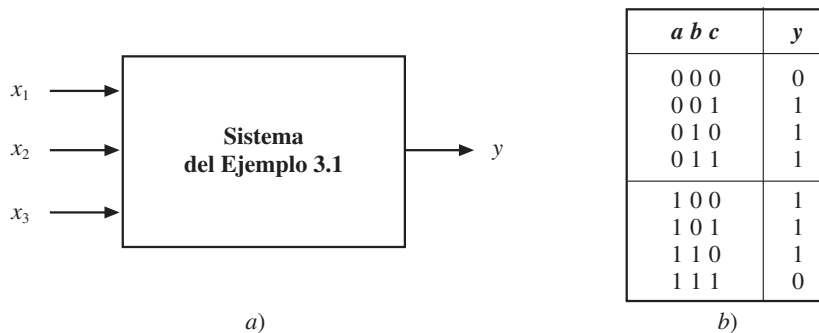
Un sistema combinacional se puede representar por medio de una **tabla verdad**, que consiste en una tabla que describe por extenso el funcionamiento del sistema; es decir, proporciona los valores de salida para las  $2^n$  combinaciones posibles de entrada.

### EJEMPLO 3.1

Describir por medio de una tabla verdad un sistema con tres entradas y una salida, que actúa de forma que su salida es 1 si y sólo si dos de sus entradas son iguales.

#### SOLUCIÓN

La solución se muestra en la Figura 3.2. En la tabla verdad se listan todas las combinaciones de entrada, y si en éstas hay dos y sólo dos valores cero o uno, la salida ( $y$ ) debe ser 1; en cualquier otro caso la salida debe ser 0.



**Figura 3.2.** Sistema del Ejemplo 3.2: a) esquema simplificado y b) tabla verdad.

Obsérvese en la tabla de la Figura 3.2 que las  $2^n = 2^3 = 8$  combinaciones posibles se forman poniendo en la primera columna  $2^{n/2} = 4$  ceros y cuatro unos; en la siguiente columna alternativamente la mitad de ceros y unos que en la columna anterior (2, en el ejemplo), y así sucesivamente. Siempre en la última columna se tendrá *cero, uno, cero, uno*, etc.

Los sistemas combinacionales, al igual que los secuenciales, que se estudiarán en la Sección 3.2, están fundamentados en el álgebra de conmutación, que analizaremos brevemente en la Sección 3.1.1. El álgebra de conmutación permite representar una función en forma algebraica, y a partir de ella diseñar el circuito que sintetiza dicha función. Para que el circuito sea del menor coste posible, interesa minimizar al máximo la función de conmutación a sintetizar; por ello resulta de sumo interés utilizar técni-

cas para minimizar funciones (Sección 3.1.2). La Sección 3.1.3 se dedica a describir cómo a partir de una descripción en extenso de una función (tabla verdad) se puede obtener la función algebraica correspondiente. A partir de las funciones de conmutación, como se indicará en la Sección 3.1.4, se pueden implementar sistemas combinacionales utilizando puertas lógicas como elementos o primitivas constitutivos. La Sección 3.1.5 describirá bloques o subsistemas combinacionales que se utilizan para implementar las distintas unidades que integran un sistema digital en general, y un computador en particular.

### 3.1.1. PRINCIPIOS DEL ÁLGEBRA DE CONMUTACIÓN

El **Álgebra de Boole**, como otras álgebras, se caracteriza por un conjunto **B** de  $k \geq 2$  elementos, unas operaciones de composición interna que se aplican a los elementos de **B** y un conjunto de axiomas o principios. Las propiedades del álgebra están constituidas por los propios axiomas junto con las leyes que se deducen de los axiomas y de las demostraciones o teoremas.

Desde el punto de vista del diseño de sistemas digitales, las principales propiedades que interesa aplicar son las que se muestran en la Tabla 3.1. Aquí (como en las referencias bibliográficas [12] y [17]) se han considerado como postulados las siete primeras propiedades, pudiéndose demostrar, por medio de teoremas, las restantes. Se utilizan tres operadores:

- El producto lógico que se denota con un punto ( $\cdot$ ); a veces entre dos variables se omite (así  $a \cdot b = ab$ ) siempre que no se produzcan ambigüedades.
- La suma lógica que se representa con el signo más (+).
- La complementación que se representa con una barra encima de la variable o expresión, o con una tilde; así,  $\bar{b}$  es lo mismo que  $b'$ , y se dice “*complemento de b*”.

Hay dos alternativas para demostrar estos teoremas: por extenso y algebraicamente. Las demostraciones por extenso se fundamentan en que dos expresiones booleanas son iguales, si para todas las combinaciones de sus variables se obtienen los mismos resultados; es decir, si realizando la tabla verdad para ambas expresiones, se obtienen en la columna de resultado los mismos valores.

			$\exists B \mid x, y \in B, x \neq y$	
Axiomas	P1	Elementos		
	P2	Leyes de composición interna	$x + y \in B \quad \forall x, y \in B$	$x \cdot y \in B \quad \forall x, y \in B$
	P3	Elementos neutros únicos	$x + 0 = 0 + x = x$	$x \cdot 1 = 1 \cdot x = x$
	P4	Conmutatividad	$x + y = y + x$	$x \cdot y = y \cdot x$
	P5	Distributividad	$x + (y \cdot z) = (x + y) \cdot (x + z)$	$x \cdot (y + z) = (x \cdot y) + (x \cdot z)$
	P6	Asociatividad	$x + (y + z) = (x + y) + z = x + y + z$	$x \cdot (y \cdot z) = (x \cdot y) \cdot z = x \cdot y$
	P7	Elemento simétrico	$x + x' = 1$	$x \cdot x' = 0$
Teoremas	P8	Idempotencia	$x + x = x$	$x \cdot x = x$
	P9	Absorción de una variable	$x + 1 = 1$	$x \cdot 0 = 0$
	P10	Complementación	$1' = 0$	$0' = 1$
	P11	Ley de De Morgan	$(x + y)' = x' \cdot y'$	$(x \cdot y)' = x' + y'$
	P12	Complementos sucesivos	$(x')' = x$	
	P13	Absorción de dos operadores	$x + x \cdot y = x$	$x \cdot (x + y) = x$
	P14	Absorción de un operador	$x + x' y = x + y$	$x \cdot (x' + y) = x \cdot y$

**Tabla 3.1.** Axiomas y teoremas básicos del Álgebra de Boole.



**EJEMPLO 3.2**

Demostrar algebraicamente, a partir de las propiedades P1 a P7, que se verifica  $a + 1 = 1$ .

**SOLUCIÓN**

A continuación se da una demostración; entre paréntesis se indica la propiedad que se aplica para establecer cada una de las igualdades.

$$\begin{aligned}
 a + 1 &= (a + 1) \cdot 1 && \text{(P3)} \\
 &= (a + 1) \cdot (a + a') && \text{(P7)} \\
 &= a + (1 \cdot a') && \text{(P5)} \\
 &= a + a' && \text{(P3)} \\
 &= 1 && \text{(P7)}
 \end{aligned}$$

con lo que queda demostrado.

**EJEMPLO 3.3**

Demostrar en extenso que se verifica la Ley de De Morgan:  $(a + b)' = a' \cdot b'$ .

**SOLUCIÓN**

En la Tabla 3.2 se muestran los valores de las expresiones  $(a + b)'$  y  $a' \cdot b'$  para todas las combinaciones posibles de las variables  $a$  y  $b$ ; y se comprueba, comparando las columnas 3 y 6, que son iguales, con lo que queda demostrado. En la tabla se incluye, después de los valores, la propiedad de la Tabla 3.1 que se ha utilizado para dar cada valor.

$a \ b$	$a + b$	$(a + b)'$	$a'$	$b'$	$a' \cdot b'$
0 0	0 (P3)	1 (P10)	1 (P10)	1 (P10)	1 (P8)
0 1	1 (P3)	0 (P10)	1 (P10)	0 (P10)	0 (P3)
1 0	1 (P3)	0 (P10)	0 (P10)	1 (P10)	0 (P3)
1 1	1 (P9)	0 (P10)	0 (P10)	0 (P10)	0 (P8)

**Tabla 3.2.** Demostración de que  $(a + b)' = a' \cdot b'$ .

El diseño de sistemas combinacionales está fundamentado en el **Álgebra de Conmutación**, que es un caso particular del Álgebra de Boole en la que el número de elementos del conjunto  $B$  es 2, y que representaremos aquí por 0 y 1. En definitiva, una **variable de conmutación** es un símbolo que puede tomar, en un momento dado, uno entre dos valores (0 o 1), y una **función de conmutación** de  $n$  variables,  $f(x_1, \dots, x_n)$ , es una aplicación del conjunto  $C = \{0,1\}^n$  en el conjunto  $B = \{0,1\}$ , donde  $\{0,1\}^n$  representa el producto cartesiano, es decir:

$$\begin{aligned}
 C &= \{0,1\}^n = \\
 &= \{00\dots00, 00\dots01, 00\dots10, 00\dots10, 00\dots11, \dots, 11\dots00, 11\dots01, 11\dots10, 00\dots10, 11\dots11\}
 \end{aligned}$$

De las propiedades indicadas en la Tabla 3.1 se deducen las tablas para las operaciones básicas:

- **Complementación**, también denominada **inversión** o **negación**, o **función NO** o **función NOT**, denominada así porque el valor de la variable se invierte (niega o complementa):

$$\text{si } x \text{ es } 0, x' \text{ es } 1; \text{ y si } x \text{ es } 1, x' \text{ es } 0$$

- La **suma lógica**, también denominada **función O** o **función OR**; este nombre se debe a que:

$$a + b \text{ es } 1, \text{ si y sólo si } a \text{ es } 1 \text{ o } b \text{ es } 1$$

- El **producto lógico**, o **función Y** o **función AND**, denominada así debido a que:

$$a \cdot b \text{ es } 1, \text{ si y sólo si } a \text{ es } 1 \text{ y } b \text{ es } 1.$$

Otras funciones, muy utilizadas que se pueden expresar en función de las anteriores son:

- **Función NOR u OR complementada:**

$$(x + y)' \text{ es } 1 \text{ si y sólo si } x \text{ e } y \text{ son } 0$$

- **Función NAND o AND complementada:**

$$(x \cdot y)' \text{ es } 0 \text{ si y sólo si } x \text{ e } y \text{ son } 1$$

- **Función exclusive-OR o XOR:**

$x \oplus y$  es 1 si y sólo si  $x$  e  $y$  son distintos, o, para más de dos variables si el número de entradas 1 es impar.

- **Función exclusive-OR complementada o EXNOR:**

$(x \oplus y)'$  es 1 si y sólo si  $x$  es igual a  $y$ , o, para más de dos variables si el número de entradas 1 es par.

Las tablas correspondientes a estas funciones se incluyen en la Tabla 3.3.

		NOT	OR	AND	NOR	NAND	EXOR	EXNOR
$x$	$y$	$x'$	$x + y$	$x \cdot y$	$(x + y)'$	$(x \cdot y)'$	$x \oplus y$	$(x \oplus y)'$
0	0	1	0	0	1	1	0	1
0	1	1	1	0	0	1	1	0
1	0	0	1	0	0	1	1	0
1	1	0	1	1	0	0	0	1

**Tabla 3.3.** Operaciones básicas del álgebra de conmutación.

### 3.1.2. MINIMIZACIÓN ALGEBRAICA DE FUNCIONES

El comportamiento de una función se suele describir de una o varias de las siguientes formas:

- Descripción en lenguaje natural, tal y como se ha realizado en el enunciado del Ejemplo 3.1.
- Tabla verdad.
- Descripción algebraica.

Las tres formas son muy útiles para el diseño de sistemas digitales, y es conveniente conocer los procedimientos para pasar de una a otra. Una función de conmutación, como se estudiará en la Sección 3.1.3, se puede diseñar directamente a partir de su tabla verdad o de su expresión algebraica, dependiendo del tipo de circuitos que se utilicen para su implementación. A partir de la descripción algebraica es muy fácil pasar a la descripción de tabla de verdad o de lenguaje natural. Ofrece más dificultad pasar de la descripción en forma de tabla verdad a la descripción algebraica, para lo cual hay que aplicar el Teorema de Shannon, que se analizará en la próxima sección (Sección 3.1.3).

**EJEMPLO 3.4**

Describir mediante una tabla verdad la función  $z = a'b' + c$ .

**SOLUCIÓN**

Es una función de tres variables  $a$ ,  $b$  y  $c$ . Construimos la tabla a partir de todas las combinaciones posibles de las tres variables, y aplicamos los valores que se obtienen de las operaciones básicas (Tabla 3.3), cosa que hacemos en la Tabla 3.4. La tabla verdad de la función  $z$  es la que se muestra en la Tabla 3.5.

$a$	$b$	$c$	$a'$	$b'$	$a' \cdot b'$	$a'b' + c$
0	0	0	1	1	1	1
0	0	1	1	1	1	1
0	1	0	1	0	0	0
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	1	0	1	0	1
1	1	0	0	0	0	0
1	1	1	0	0	0	1

**Tabla 3.4**

$a$	$b$	$c$	$z$
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

**Tabla 3.5**

Los circuitos que sintetizan funciones de conmutación suelen utilizar módulos que implementan los operadores básicos. Evidentemente cuantos menos operadores (complementaciones, sumas y productos lógicos) y cuantas menos variables intervengan en una función, menos componentes tendrá el sistema que la sintetiza, y será más barato y más sencillo de implementar y mantener. Interesa, por lo tanto, simplificar o minimizar las funciones al máximo antes de implementarlas. La minimización se puede realizar mediante manipulación algebraica, como se indica en el siguiente ejemplo.

**EJEMPLO 3.5**

Simplifica la siguiente función de conmutación:  $z = ab + abc + a'cd + a'c'd + a'bcd$ .

**SOLUCIÓN**

$$\begin{aligned}
 z &= ab + abc + a'cd + a'c'd + a'bcd' = \\
 &= ab + abc + a'cd + a'c'd + abc + a'bcd' \quad \text{(la expresión no varía si repetimos alguno de sus términos, en este caso } abc) \\
 &= ab(1 + c) + a'd(c + c') + bc(a + a'd') = \quad \text{(P5)} \\
 &= ab + a'd + bc(a + d') = \quad \text{(P9, P3, P7, P14)} \\
 &= ab + a'd + abc + bcd' = \quad \text{(P5)} \\
 &= ab(1 + c) + a'd + bcd' = \quad \text{(P8, P5)} \\
 &= ab + a'd + bcd' \quad \text{(P13)}
 \end{aligned}$$

Obviamente, será más económico implementar la expresión  $ab + a'd + bcd'$  que la  $ab + abc + a'cd + a'c'd + a'bcd'$ ; y ambas sintetizan la misma función.

La simplificación por medio de manipulación algebraica no se realiza aplicando reglas precisas y depende notablemente de la intuición y experiencia del diseñador. Para solventar este problema se han ideado procedimientos sistemáticos como el de los Mapas de Karnaugh y el método de Quine McClus-

key, cuya explicación va más allá de los objetivos de este libro. Existen numerosas herramientas software para el diseño de circuitos digitales que suelen incluir programas para minimización de funciones.

### 3.1.3. FORMAS NORMALIZADAS

Con frecuencia para realizar la implementación de un sistema digital es necesario obtener la descripción algebraica de la función que lo representa. El **Teorema de Shannon** proporciona un método sistemático para hacerlo. Previamente vamos a introducir el concepto de término mínimo.

Si se tiene una función de  $n$  variables, un **término mínimo** (o **minterm**) es cualquier producto lógico en el que aparezcan las  $n$  variables de la función, complementadas o no. A cada combinación de entrada de una función le podemos asociar un término mínimo, en el que aparecerán todas las variables: sin complementar, las que correspondan a entradas 1, y complementadas, las que aparezcan con entrada 0.

#### EJEMPLO 3.6

Obtener los términos máximos posibles de una función de 4 variables.

##### SOLUCIÓN

En la segunda columna de la Tabla 3.6 se muestran los 16 términos mínimos posibles.

El **Teorema de Shannon** establece que la descripción algebraica de una función puede obtenerse a partir de su tabla verdad, sumando los términos mínimos correspondientes a las combinaciones de las entradas para las que la función es 1.

#### EJEMPLO 3.7

Obtener la expresión algebraica de la función dada en la Tabla 3.7.

$a$ $b$ $c$ $d$	Término mínimo
0 0 0 0	$a'b'c'd'$
0 0 0 1	$a'b'c'd$
0 0 1 0	$a'b'cd'$
0 0 1 1	$a'b'cd$
0 1 0 0	$a'bc'd'$
0 1 0 1	$a'bc'd$
0 1 1 0	$a'bcd'$
0 1 1 1	$a'bcd$
1 0 0 0	$ab'c'd'$
1 0 0 1	$ab'c'd$
1 0 1 0	$ab'cd'$
1 0 1 1	$ab'cd$
1 1 0 0	$abc'd'$
1 1 0 1	$abc'd$
1 1 1 0	$abcd'$
1 1 1 1	$abcd$

**Tabla 3.6.** Términos mínimos asociados a las distintas combinaciones de entrada para una función de  $n = 4$  variables.

$a$ $b$ $c$	$z$
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	0
1 1 1	1

**Tabla 3.7.** Función del Ejemplo 3.7.

## SOLUCIÓN

Para mejor comprender el procedimiento, en la Tabla 3.8 se repite la 3.7, incluyendo los términos mínimos correspondientes a cada combinación de entrada.

<i>a b c</i>	Término máximo	<i>z</i>
0 0 0	<i>a'b'c'</i>	0
0 0 1	<i>a'b'c</i>	1
0 1 0	<i>a'bc'</i>	0
0 1 1	<i>a'bc</i>	1
1 0 0	<i>ab'c'</i>	0
1 0 1	<i>ab'c</i>	1
1 1 0	<i>abc'</i>	0
1 1 1	<i>abc</i>	1

**Tabla 3.8.** Términos mínimos asociados a las distintas combinaciones de entrada.

De acuerdo con la regla de aplicación del teorema de Shannon debemos seleccionar los términos mínimos correspondientes a las combinaciones para las que  $z = 1$ ; es decir:

- $z = 1$  para la combinación  $abc = 001$ , luego seleccionamos como sumando el término mínimo: *a'b'c*.
- $z = 1$  para la combinación  $abc = 011$ , luego seleccionamos como sumando el término mínimo: *a'bc*.
- $z = 1$  para la combinación  $abc = 101$ , luego seleccionamos como sumando el término mínimo: *ab'c*.
- $z = 1$  para la combinación  $abc = 111$ , luego seleccionamos como sumando el término mínimo: *abc*.

Con lo que la función  $z$  será:

$$z = a'b'c + a'bc + ab'c + abc$$

De forma descriptiva podemos comprobar la validez de la regla utilizada. En efecto, el significado del contenido de la Tabla 3.8 es el siguiente:

$$\begin{aligned} z \text{ es } 1 & \quad \text{si } a = 0 \text{ Y } b = 0 \text{ Y } c = 1, 0 \\ & \quad \text{si } a = 0 \text{ Y } b = 1 \text{ Y } c = 1, 0 \\ & \quad \text{si } a = 1 \text{ Y } b = 0 \text{ Y } c = 1, 0 \\ & \quad \text{si } a = 1 \text{ Y } b = 1 \text{ Y } c = 1 \end{aligned}$$

donde hemos destacado con letra gótica las conjunciones  $y$  (Y) y  $o$  (O).

Ahora bien, decir que  $x = 0$  es lo mismo que decir  $x' = 1$ , con lo que la descripción anterior se puede realizar así:

$$\begin{aligned} z \text{ es } 1 & \quad \text{si } a' = 1 \text{ Y } b' = 1 \text{ Y } c = 1, 0 \\ & \quad \text{si } a' = 1 \text{ Y } b = 1 \text{ Y } c = 1, 0 \\ & \quad \text{si } a = 1 \text{ Y } b' = 1 \text{ Y } c = 1, 0 \\ & \quad \text{si } a = 1 \text{ Y } b = 1 \text{ Y } c = 1 \end{aligned}$$

Por otra parte, teniendo en cuenta que afirmar que, por ejemplo,  $x' = 1 \text{ Y } y' = 1 \text{ Y } z = 1$ , es lo mismo que decir:  $x'y'z = 1$ , la tabla anterior queda:

$$\begin{aligned} z \text{ es } 1 & \quad \text{si } a'b'c = 1, 0 \\ & \quad \text{si } a'bc = 1, 0 \\ & \quad \text{si } ab'c = 1, 0 \\ & \quad \text{si } abc = 1 \end{aligned}$$

Aplicando el significado del operador  $O$ , resulta:

$$z \text{ es } 1 \quad \text{si } a'b'c + a'bc + ab'c + abc = 1$$

Con lo que:

$$z = a'b'c + a'bc + ab'c + abc$$

### EJEMPLO 3.8

Simplificar la función  $z$  del ejemplo anterior (Ejemplo 3.7).

SOLUCIÓN

$$\begin{aligned} z &= a'b'c + a'bc + ab'c + abc = \\ &= a'c(b' + b) + ac(b' + b) = \\ &= a'c + ac = \\ &= c(a' + a) = \\ &= c \end{aligned}$$

Del Ejemplo 3.8 deducimos que una misma función tiene distintas formas de representación algebraica; así, las formas:

$$\begin{aligned} z_1 &= a'b'c + a'bc + ab'c + abc = \\ z_2 &= a'c(b' + b) + ac(b' + b) = \\ z_3 &= a'c + ac = \\ z_4 &= c(a' + a) = \\ z_5 &= c \end{aligned}$$

son expresiones de la misma función (todas ellas tienen la misma tabla verdad). Por tanto, una función no tiene una expresión algebraica única, no pudiendo por simple inspección determinar si dos funciones son iguales. Se denomina **forma canónica** a la descripción que se obtiene a partir de la aplicación del Teorema de Shanon; es decir, aquella expresión formada por suma de productos en los que intervienen todas las variables de entrada, complementadas o no, y sin repetirse. Por lo tanto, la forma canónica de la función  $z$  del Ejemplo 3.8 es:  $z_1 = a'b'c + a'bc + ab'c + abc$ . La forma canónica de cualquier función es única, con lo que podemos saber si dos funciones son iguales comparando sus formas canónicas, si éstas son iguales aquéllas lo serán también.

## 3.1.4. IMPLEMENTACIÓN DE FUNCIONES. PUERTAS LÓGICAS

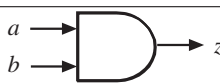
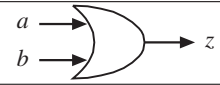
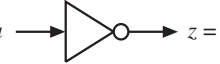




Para construir un sistema digital se disponen de circuitos que sintetizan operaciones básicas; en otras palabras, el diseñador para hacer un diseño debe seleccionar los elementos o circuitos más adecuados dentro de una biblioteca de componentes disponibles, y determinar las interconexiones entre ellos. En la Figura 3.3 se muestra una biblioteca de componentes constituida por 7 elementos. El circuito NOT tiene una sola entrada, y los demás pueden tener varias entradas, aunque, por simplificar, en la figura sólo se han incluido dos.

Como existen circuitos específicos para la síntesis de los operadores básicos (suma, producto y complementación) resulta inmediato diseñar cualquier función, según se pone de manifiesto en el siguiente ejemplo.

### EJEMPLO 3.9

Diseñar un circuito para sintetizar la función:

$$z = a'b'c' + ab'c + abc' + ab'c$$

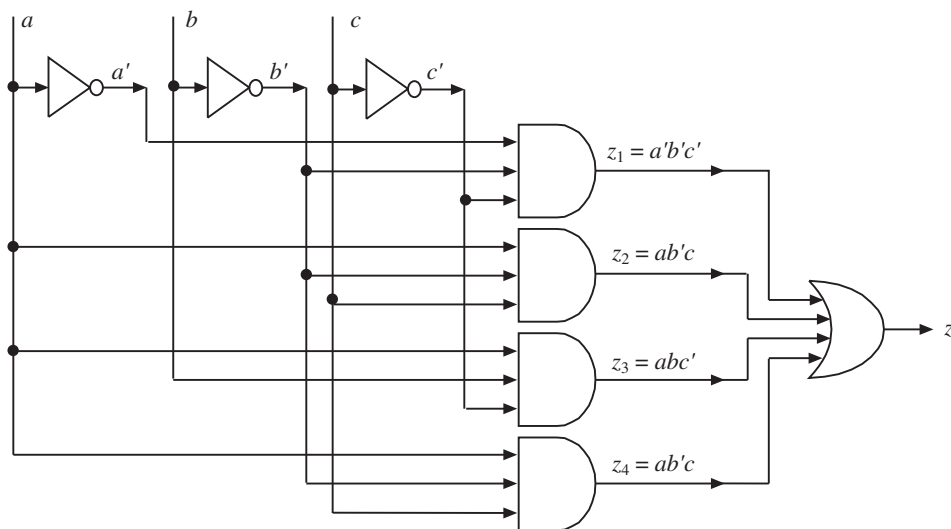
Denominación	Símbolo	Operación	Representación algebraica
AND		$z = 1$ si y sólo si todas las entradas son 1.	$z = a \cdot b$
OR		$z = 0$ si y sólo si todas las entradas son 0.	$z = a + b$
NOT		$z = 1$ si y sólo si $a = 0$ .	$z = a'$
NOR		$z = 0$ si y sólo si todas las entradas son 1.	$z = (a \cdot b)'$
NAND		$z = 1$ si y sólo si todas las entradas son 0.	$z = (a + b)'$
XOR		$z = 1$ si y sólo si el n.º de entradas uno es impar.	$z = a \oplus b$
EXNOR		$z = 1$ si y sólo si el n.º de entradas uno es par.	$z = (a \oplus b)'$

**Figura 3.3.** Biblioteca de puertas lógicas para diseñar sistemas combinacionales.

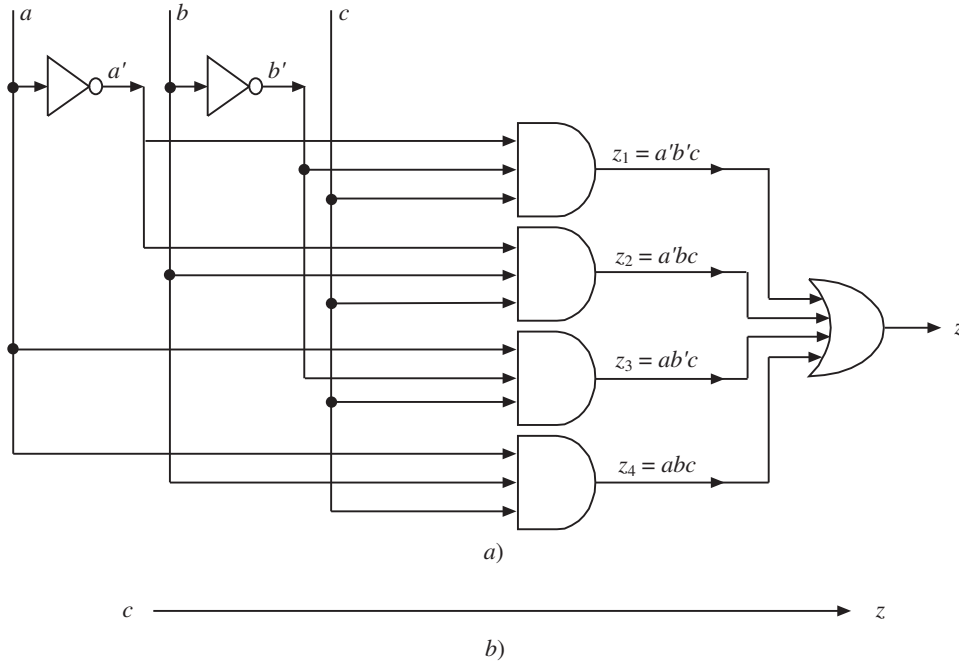
### SOLUCIÓN

Podemos sintetizar esta función con 3 inversores (para generar las variables  $a'$ ,  $b'$  y  $c'$ ), 4 puertas NAND de 3 entradas y un circuito OR de 4 entradas. El esquema del circuito es el que se muestra en la Figura 3.4.

Ahora podemos comprender mejor la importancia de minimización de funciones. En el Ejemplo 3.8 utilizamos la función  $z = a'b'c' + ab'c + abc' + abc$ ; y su implementación sería la que se indica en la Figura 3.5a. Ahora bien, simplificando esa función se llega a la conclusión de que es la mis-



**Figura 3.4.** Sistema que sintetiza la función  $z = a'b'c' + ab'c + abc' + ab'c$ .



**Figura 3.5.** Dos formas de implementar la función  $z = a'b'c + a'bc + ab'c + abc$ .

ma que  $z = c$ , con lo que se puede sintetizar sencillamente utilizando el valor de  $c$ , como muestra la Figura 3.5b. Con este ejemplo se pone claramente de manifiesto la importancia de la minimización.

Los diseños de las Figuras 3.4 y 3.5a, se conocen como **implementaciones de dos niveles** ya que, salvo los inversores de entrada, las señales de entrada tienen que pasar por dos puertas (una AND y otra OR). Se han desarrollado técnicas para distintas implementaciones. Una de ellas consiste en utilizar tan sólo puertas NAND. Se fundamenta en que, aplicando la ley de De Morgan (P11, Tabla 3.1), se verifica que:

$$a + b = (a' \cdot b')' \quad (\text{P11 y P12})$$

con lo que la operación OR, de la etapa de salida, se puede realizar con una puerta NAND, sin más que complementar las entradas. Como las entradas son resultados de productos lógicos que debemos complementarlos, en la etapa de entrada podemos sustituir las puertas AND por NAND. También la complementación se puede realizar con una puerta NAND, sin más que aplicar en todas sus entradas la señal a complementar; ya que se verifica:

$$a' = (a \cdot a)' \quad (\text{P8})$$

### EJEMPLO 3.10

Diseñar un circuito para sintetizar la siguiente función utilizando sólo puertas NAND:

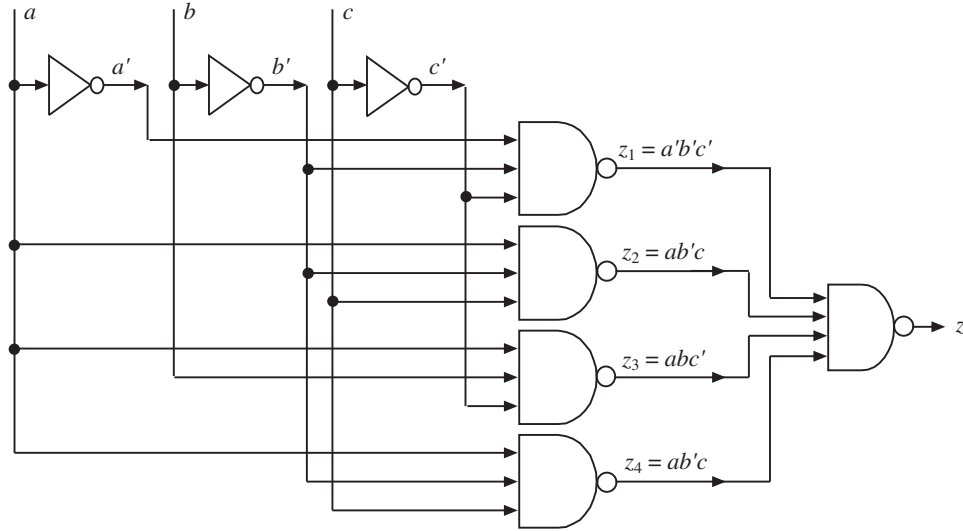
$$z = a'b'c' + ab'c + abc' + ab'c$$

### SOLUCIÓN

El resultado se muestra en la Figura 3.6, y se fundamenta en la siguiente igualdad:

$$a'b'c' + ab'c + abc' + ab'c = [(a'b'c')' \cdot (ab'c)' \cdot (abc')' \cdot (ab'c)']'$$





**Figura 3.6.** Síntesis con puertas NAND de la función  $z = a'b'c' + ab'c + abc' + ab'c$ .

Para el diseño de funciones de conmutación con puertas lógicas existen circuitos integrados, que, por ejemplo, incluyen en el mismo chip:

- 5 inversores (chip 7404).
- 4 puertas de 2 entradas (NAND: 7400; AND: 7408; OR: 7432, XOR: 7486, NOR: 7402).
- 3 puertas de 3 entradas (NAND: 7410; AND: 7411 y NOR: 7427).
- 2 puertas de 4 entradas (NAND: 7420; AND: 7421).
- 1 puerta de 8 entradas (NAND: 7430).

### 3.1.5. BLOQUES COMBINACIONALES BÁSICOS

Esta sección se dedica al análisis de algunos circuitos combinacionales de uso muy extendido, y de los que se dispone comercialmente en forma de circuitos integrados. Estos elementos pueden incluirse en la biblioteca de componentes o primitivas de que dispone el diseñador para construir sistemas digitales. Los bloques que se van a describir son los siguientes:

- Sumador binario.
- Decodificador.
- Codificador y codificador de prioridad.
- Multiplexor.
- Demultiplexor.
- Puerta tri-estado.
- Dispositivos lógicos programables: ROM, PLA y PAL.
- Unidad aritmético-lógica (ALU).

#### 3.1.5.1. Sumador binario

Para realizar una suma binaria (Apéndice 1) utilizamos el mismo algoritmo que para la suma decimal, pero con la tabla de sumar del sistema de numeración base 2. Analicemos con un ejemplo el proceso de la suma aritmética de dos números.

**EJEMPLO 3.11**

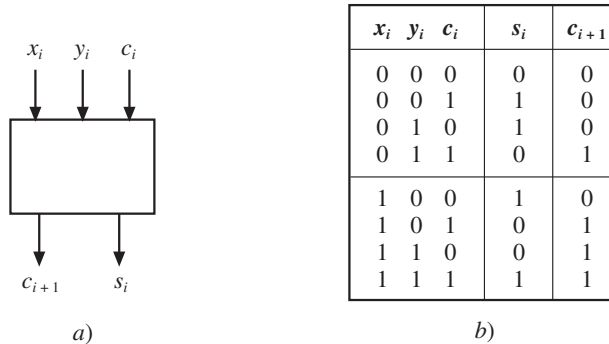
Sumar los números binarios  $X = 1001\ 0101$  e  $Y = 0111\ 0101$

**SOLUCIÓN**

La suma es la siguiente:

$$\begin{array}{r}
 \text{Acarreos} \rightarrow \begin{array}{cccccccccc} c_8 & c_7 & c_6 & c_5 & c_4 & c_3 & c_2 & c_1 & c_0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \end{array} \\
 \begin{array}{r} 1.^{\text{er}} \text{ sumando (X)} \rightarrow \\ 2.^{\text{o}} \text{ sumando (Y)} \rightarrow \end{array} \begin{array}{cccccccccc} & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ + & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \end{array} \\
 \text{Resultado (S)} \rightarrow \begin{array}{cccccccccc} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{array}
 \end{array}$$

Puede observarse que en cada bit,  $i$ , realmente hay que hacer la suma de tres valores:  $c_i$  (que es el acarreo que procede de la etapa anterior,  $i - 1$ ),  $x_i$  o cifra del primer sumando de la posición  $i$ , e  $y_i$  o cifra del segundo sumando que se encuentra en la posición  $i$ . Como resultado de la operación por cada bit se obtienen dos valores: el bit del resultado de la suma,  $s_i$ , y el acarreo para la etapa siguiente,  $c_{i+1}$ . En definitiva, cada etapa se puede implementar con un circuito de tres entradas ( $c_i$ ,  $x_i$ ,  $y_i$ ) y dos salidas ( $s_i$ ,  $c_{i+1}$ ), tal y como se representa en la Figura 3.7a. En la Figura 3.7b se muestra la tabla verdad para las funciones  $s_i$ ,  $c_{i+1}$ .



**Figura 3.7.** Sumador binario: a) esquema del sistema y b) tabla verdad.

A partir de la tabla verdad de la Figura 3.7b se pueden obtener los circuitos que sintetizan las funciones  $s_i$ ,  $c_{i+1}$ . En efecto, aplicando el teorema de Shannon y simplificando, se tiene:

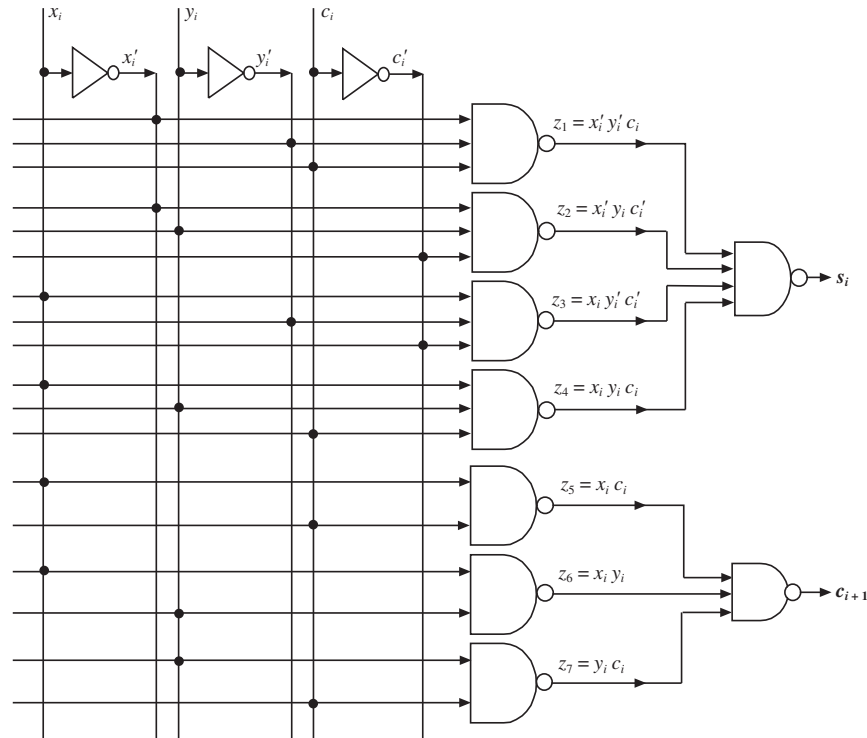
$$\begin{aligned}
 s_i &= x_i' \cdot y_i' \cdot c_i + x_i' \cdot y_i \cdot c_i' + x_i \cdot y_i' \cdot c_i' + x_i \cdot y_i \cdot c_i \\
 c_{i+1} &= x_i' \cdot y_i \cdot c_i + x_i \cdot y_i' \cdot c_i + x_i \cdot y_i \cdot c_i' + x_i \cdot y_i \cdot c_i \\
 &= x_i' \cdot y_i \cdot c_i + x_i \cdot y_i' \cdot c_i + x_i \cdot y_i \cdot c_i' + x_i \cdot y_i \cdot c_i + x_i \cdot y_i \cdot c_i + x_i \cdot y_i \cdot c_i \\
 &= [x_i' \cdot y_i \cdot c_i + x_i \cdot y_i \cdot c_i] + [x_i \cdot y_i' \cdot c_i + x_i \cdot y_i \cdot c_i] + [x_i \cdot y_i \cdot c_i' + x_i \cdot y_i \cdot c_i] = \\
 &= y_i \cdot c_i + x_i \cdot c_i + x_i \cdot y_i
 \end{aligned}$$

Con lo que el circuito que sintetiza una etapa del sumador es el de la Figura 3.8.

Si se desea obtener un sumador de  $n$  etapas no hay más que conectar adecuadamente  $n$  sumadores de una etapa, tal como se muestra en el siguiente ejemplo.

**EJEMPLO 3.12**

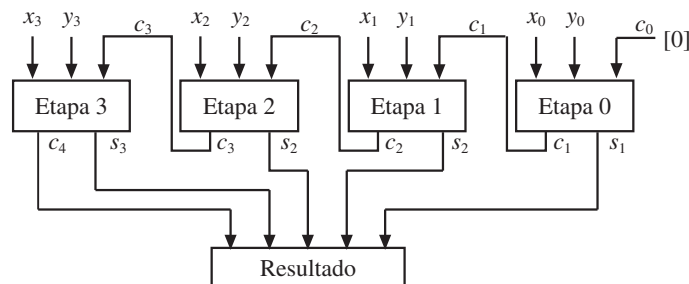
Diseñar un sumador de 4 bits, utilizando sumadores de una etapa.



**Figura 3.8.** Circuito que sintetiza una etapa de un sumador binario.

### SOLUCIÓN

Tenemos que utilizar 4 sumadores de una etapa, y conectarlos de forma que el acarreo de orden 0 sea un 0, y que el acarreo de salida de cada etapa se lleve al acarreo de entrada de la siguiente. El resultado se muestra en la Figura 3.9.

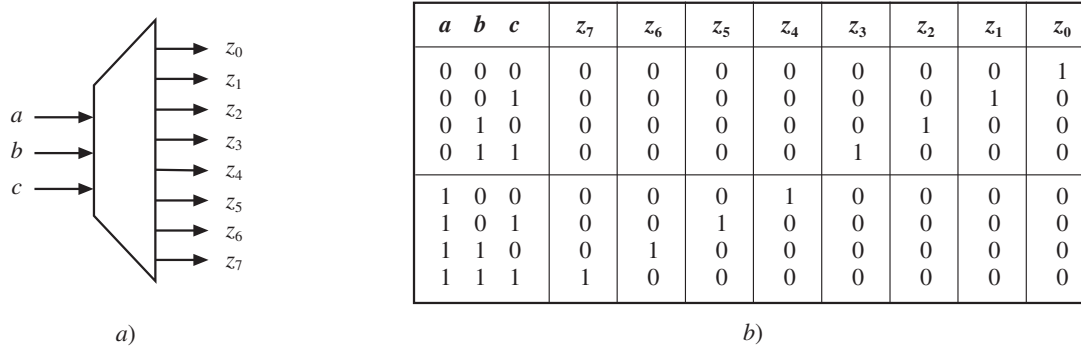


**Figura 3.9.** Sumador de 4 etapas. Cada etapa es un circuito como el de la Figura 3.8.

Un problema que presenta el sumador anterior es que, como se comentó en el inicio de esta sección (Sección 3.1), cada etapa invierte un determinado tiempo (**tiempo de retardo,  $p$** ) en producir la salida correcta, por lo que para obtener la suma completa será necesario que se vayan propagando los acarreos desde la etapa 0 a la etapa  $n - 1$ ; es decir, que la respuesta no es válida hasta  $n \cdot p$  segundos después de haber dado en las entradas los sumandos. En otras palabras, el tiempo de respuesta del circuito depende del número de bits del sumador. Hay otros circuitos sumadores que resuelven este problema.

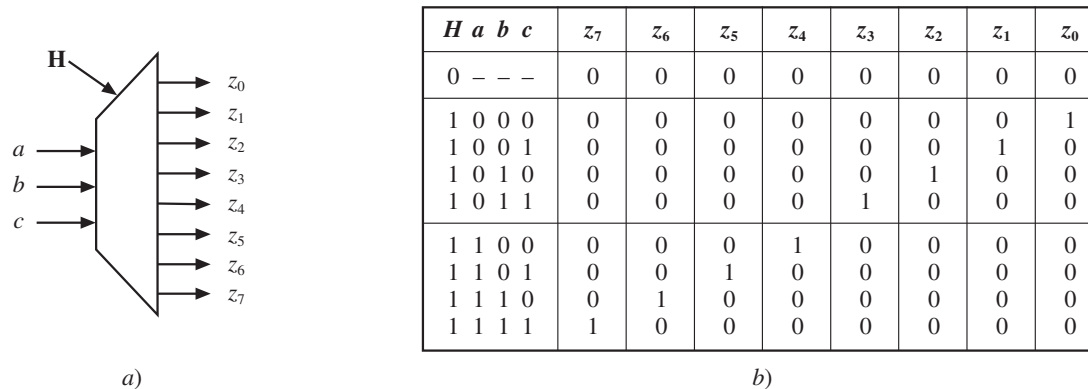
### 3.1.5.2. Decodificador

Un decodificador binario es un circuito combinacional con  $n$  entradas y  $m = 2^n$  salidas, y actúa de forma tal que para combinación de entrada se activa sólo una de sus salidas; es decir, su funcionamiento es el que se indica en la Figura 3.10, para  $n = 3$ .



**Figura 3.10.** Decodificador de 3 en 8: a) símbolo y b) tabla verdad.

Existen decodificadores que disponen de una entrada adicional denominada de **habilitación** (*enable*). Si la señal de habilitación es 0, todas las salidas son 0, independientemente de las otras entradas; cuando la señal de habilitación es 1, actúa como un decodificador normal (Figura 3.11). Las entradas adicionales a un circuito, como la de habilitación que acabamos de comentar, que determinan distintas opciones del sistema, se denominan **señales de control**; en concreto **H** es una señal de control.



**Figura 3.11.** Decodificador de 3 en 8, con señal de habilitación (**H**): a) símbolo y b) tabla verdad (el signo —: significa 0 o 1, es indiferente).

### 3.1.5.3. Codificador y codificador de prioridad

Un **codificador binario** es un circuito combinacional con  $m$  entradas y  $n$  salidas, siendo  $m = 2^n$ , y actúa de forma inversa al decodificador; es decir, cuando una de sus entradas es 1 aparece en la salida una combinación que identifica unívocamente la entrada activada. Únicamente una de sus entradas puede ser 1. En la Figura 3.12 se muestra el símbolo y la tabla verdad de un codificador binario de 8 a 3.

Una variante de los codificadores anteriormente citados son los codificadores de prioridad. En un **codificador de prioridad** pueden estar a 1 varias de las entradas, y la salida identifica la entrada de orden más alta que esté activa. También es normal que se disponga de una salida adicional, **sin petición** (*not request* o NR) que se pone a 1 cuando ninguna de las entradas es 1. La Tabla 3.9 muestra la tabla verdad de un codificador de prioridad de 8 entradas.

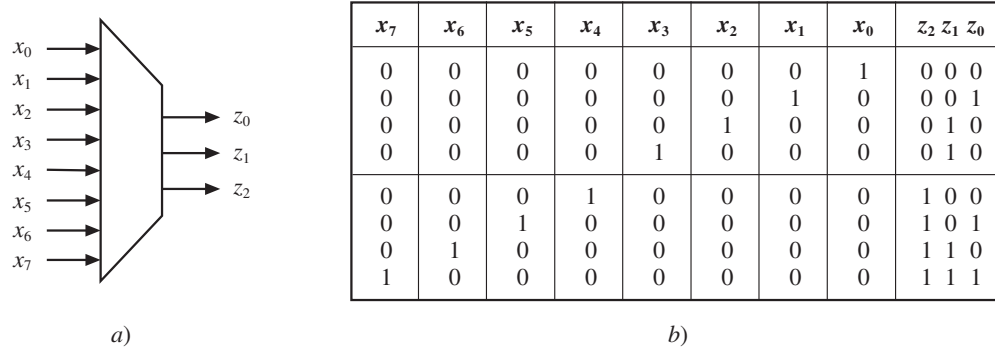


Figura 3.12. Decodificador de 3 en 8: a) símbolo y b) tabla verdad.

$x_7$	$x_6$	$x_5$	$x_4$	$x_3$	$x_2$	$x_1$	$x_0$	$z_2$	$z_1$	$z_0$	$NR$
0	0	0	0	0	0	0	0	*	*	*	1
0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	1	0	0	0	0	1	1	0
0	0	0	1	0	0	0	0	1	0	0	0
0	0	1	0	0	0	0	0	1	0	1	0
0	1	0	0	0	0	0	0	1	1	0	0
1	0	0	0	0	0	0	0	1	1	1	0

Tabla 3.9. Tabla verdad de un codificador con prioridad (el signo \* indica que no son válidas las salidas).

### 3.1.5.4. Multiplexor

Un **multiplexor** es un circuito combinacional que dispone de  $m$  entradas de datos, 1 salida y  $n$  (con  $m = 2^n$ ) entradas de control. El circuito actúa de tal forma que a la salida aparece la señal que hubiese en ese momento en la entrada seleccionada por las señales de control. Las entradas de control de un multiplexor se las suele denominar **entradas de selección**.

#### EJEMPLO 3.13

Un multiplexor de 8 entradas ( $x_0, \dots, x_7$ ) tendrá tres señales de selección ( $s_2, s_1, s_0$ ) y una salida ( $z$ ). Si las señales de selección tienen el valor  $s_2 = 1$ ,  $s_1 = 1$  y  $s_0 = 0$ , se selecciona la entrada  $x_6$  (ya que  $(110)_2 = 6_{10}$ ), con lo que en la salida aparecerá el valor que haya en  $x_6$ ; es decir:  $z = x_6$ .

En la Figura 3.13 se muestra el símbolo utilizado para representar un multiplexor, y una tabla que describe su comportamiento.

### 3.1.5.5. Demultiplexor

Un demultiplexor, como su nombre indica, realiza la función inversa de un multiplexor. En efecto, dispone de una entrada ( $x$ ),  $m$  salidas y  $n$  señales de selección. El circuito actúa de forma que la señal de la entrada aparece en la salida seleccionada por las señales de control.

En la Figura 3.14 se muestra el símbolo utilizado para representar un demultiplexor, y una tabla que describe su comportamiento.

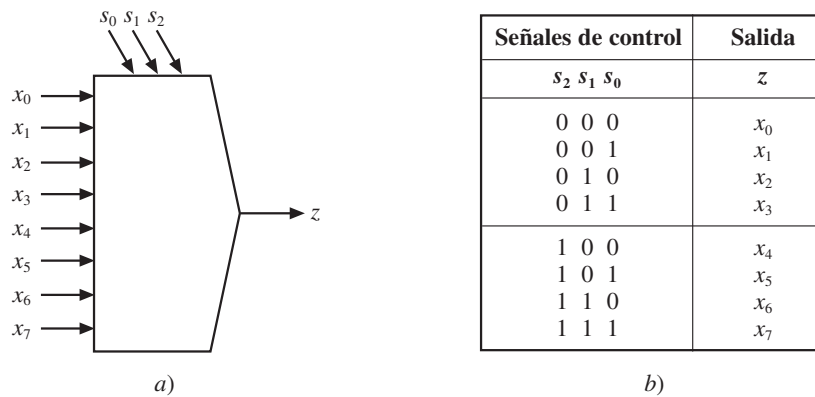


Figura 3.13. Multiplexor de 8 entradas: a) símbolo y b) tabla de comportamiento.

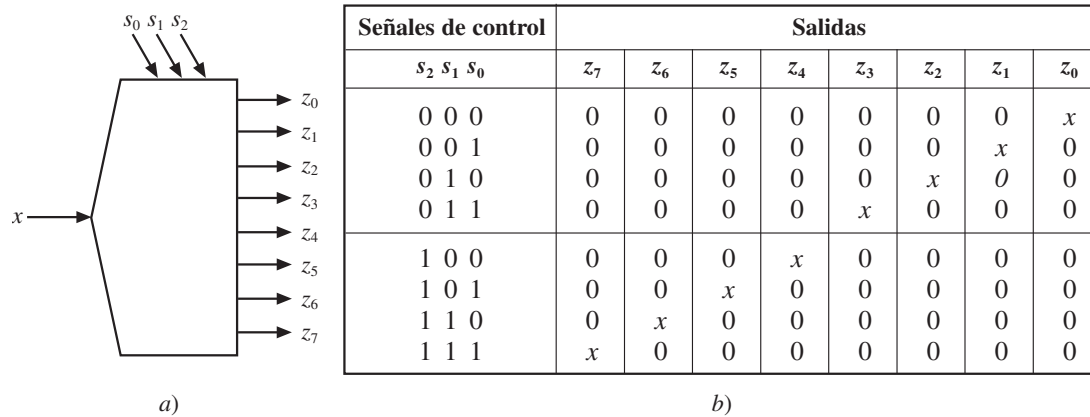


Figura 3.14. Demultiplexor de 8 salidas: a) símbolo y b) tabla de comportamiento.

### 3.1.5.6. Puerta tri-estado

Una puerta lógica, o cualquier circuito digital, en general, se dice que es **tri-estado**, cuando su salida puede tomar los valores 0, 1 y de alta impedancia. El estado de alta impedancia, que aquí denotamos con la letra  $D$  (de “desconectar”), hace como si la salida estuviese en circuito abierto; es decir, como si estuviese eléctricamente desconectada. Los circuitos tri-estado disponen de una señal de control (habilitación o, abreviadamente,  $EN$ ), de forma que si es 0, la salida está en estado de alta impedancia, y si es 1, está en estado de baja impedancia, apareciendo en su salida el valor 0 o 1, según corresponda.

Un **buffer tri-estado** es un circuito que, cuando está en estado de baja impedancia, en la salida aparece la entrada que hubiese en ese momento (Figura 3.15).

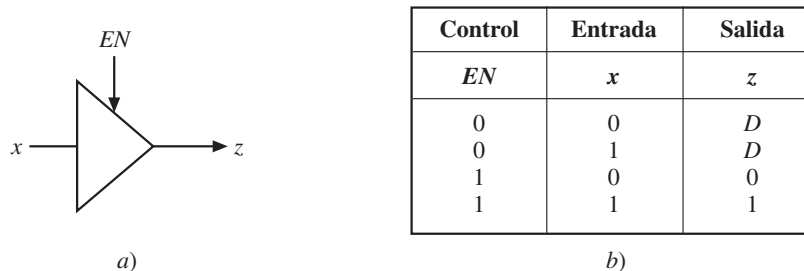
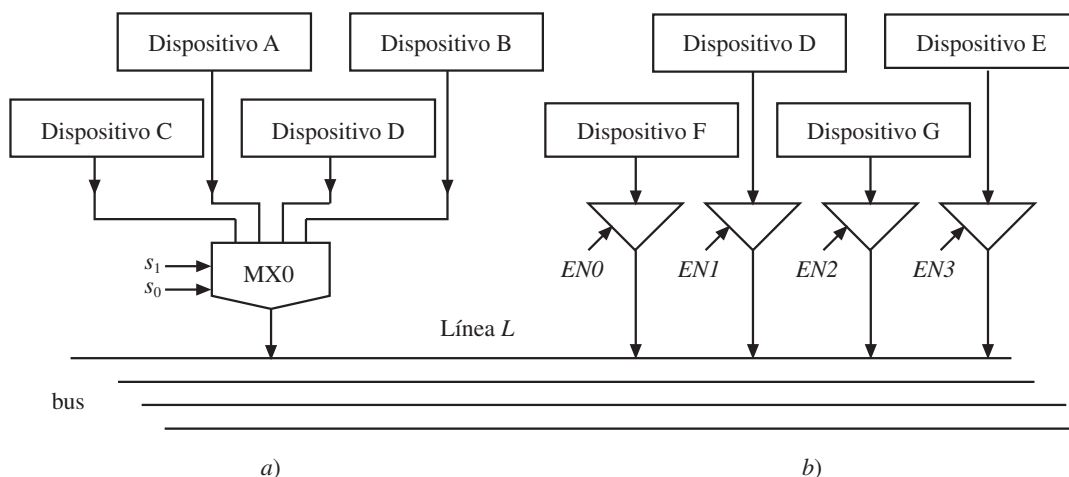


Figura 3.15. Buffer tri-estado: a) símbolo y b) tabla de comportamiento.

Las puertas tri-estado son de gran utilidad si deseamos conectar las salidas de varios circuitos a una misma línea eléctrica; por ejemplo, una línea de un bus. Supongamos que deseamos conectar las salidas de 4 circuitos o dispositivos a una misma línea conductora,  $L$ . Las 4 salidas no las podemos conectar directamente a  $L$ , ya que si no produciríamos un cortocircuito; por ejemplo, si la salida de A fuese 1 (3,5 voltios) y la de B, 0 (0 voltios), estaríamos aplicando a un conductor simultáneamente dos niveles de tensión: 3,5 voltios y 0 voltios, lo que produciría una corriente extremadamente elevada (un cortocircuito). Hay dos alternativas para solucionar el problema: la primera es utilizar un multiplexor (Figura 3.16a), que se encargue de seleccionar en un momento dado la salida que se conecta al bus. La otra es utilizar circuitos tri-estado, o *buffers* tri-estado (Figura 3.16b); teniendo la precaución de no habilitar en más de un buffer el estado de baja impedancia (en un instante dado sólo una de las tres señales  $EN0$ ,  $EN1$ ,  $EN2$  o  $EN3$  puede ser 1).



**Figura 3.16.** Conexión de varios dispositivos a una línea de un bus: a) los dispositivos A, B, C y D con un multiplexor; b) los dispositivos D, E, F y G con *buffers* tri-estado.

### 3.1.5.7. Dispositivos lógicos programables: ROM, PLA y PAL

Existen alternativas de diseño distintas a las vistas hasta ahora. Una de ellas es la de utilizar **dispositivos lógicos programables sencillos** o **SPLDs** (*Simple Programmable Logic Devices*). Entre los SPLD más utilizados se encuentran las ROM, PLA y PAL.

Todos estos circuitos se caracterizan porque tienen una estructura regular que es particularizada por el usuario, para una aplicación concreta. Tal como salen de fábrica no realizan ninguna función, y es el diseñador quien los programa o configura para una tarea concreta. Estos circuitos tienen un número  $n$  de entradas ( $e_1, e_2, \dots, e_n$ ) y generan simultáneamente un conjunto  $p$  de funciones de salida ( $s_1, s_2, \dots, s_p$ ).

La idea fundamental es que cualquier función de conmutación se puede obtener como suma de términos productos, mínimos o no. Recordemos que los términos mínimos están formados por productos de todas las variables, complementadas o no, de las que depende de la función. Los SPLD se componen de dos partes, una de entrada que se denomina plano AND, que genera todos los términos mínimos posibles de la función; es decir, para una función de  $n$  variables, generará sus  $2^n$  términos mínimos. Por otra parte la salida del dispositivo se denomina plano OR, que contiene un total de  $m$  puertas OR (una por cada función a generar). Cada puerta OR del plano de salida tiene conectada a sus entradas la totalidad de salidas del plano AND (todos los términos producto generados en el plano AND). La sintetización de funciones se realiza sencillamente *rompiendo* conexiones (es decir, *programando*), o bien en el plano AND, o en el plano OR, o en ambos.

**EJEMPLO 3.14**

Suponga que se dispone de un SPLD de 3 entradas y 4 salidas, que genera en el plano de entrada todos los términos mínimos de las entradas, y en el que se puede programar el plano de salida. Indicar las conexiones que habría que destruir en el plano de salida para sintetizar las cuatro funciones cuya tabla verdad se da en la Tabla 3.10.

**SOLUCIÓN**

Cada salida del circuito programable contiene una puerta OR con 8 entradas, correspondientes a las salidas de las 8 puertas AND de 3 entradas del plano de entrada. Por lo tanto, tendremos que destruir las conexiones de las entradas de las puertas OR correspondientes a los productos que no intervengan en la función; es decir, las combinaciones para las que la función correspondiente es 0. Si en cada puerta OR del plano de salida denominamos 0 a la entrada correspondiente al término  $e_2e_1'e_0'$ , 1 a la entrada del término  $e_2'e_1'e_0$ , ..., y 7 a la entrada del término  $e_2e_1e_0$ , las entradas a destruir serán:

- Puerta OR cuya salida es la función  $s_3$ ; desconectar sus entradas 2, 4 y 6.
- Puerta OR cuya salida es la función  $s_2$ ; desconectar sus entradas 0, 3, 5 y 7.
- Puerta OR cuya salida es la función  $s_1$ ; desconectar sus entradas 2, 5 y 6.
- Puerta OR cuya salida es la función  $s_0$ ; desconectar sus entradas 1, 4 y 6.

En el caso del ejemplo anterior se observa que no es necesario minimizar las funciones, ya que la implementación se realiza a partir de la forma canónica o de la tabla verdad. También se observa que un mismo circuito puede implementar simultáneamente muchas funciones (tantas como salidas).

Como vimos en la Sección 1.2.1, una memoria ROM (*Read Only Memory*) es un dispositivo que contiene información grabada durante su proceso de fabricación, y que no puede borrarse posteriormente. No obstante, existen versiones que pueden ser programadas por el usuario una sola vez (PROM, *Programmable Read Only Memory*) o múltiples veces (EPROM, *Erasable and Programmable Read Only Memory*). Las memorias ROM se utilizan en múltiples aplicaciones, como parte de la memoria principal de un computador, donde se graban programas y datos que deben permanecer constantemente (rutinas de arranque, por ejemplo), para implementar memorias de control, y también pueden utilizarse para implementar funciones de conmutación.

Una memoria ROM de, por ejemplo, 1.024 palabras de 16 bits, puede utilizarse para sintetizar simultáneamente 16 funciones de 10 variables. Cada combinación de entradas (dirección) selecciona (por medio de un decodificador) una palabra de memoria apareciendo el contenido de ésta en las salidas. En otras palabras, el plano de entrada genera todos los términos mínimos de la función. Cada bit de salida proporciona una función distinta. En consecuencia, se trata de *grabar* físicamente en la memoria ROM los valores que toma la función para cada combinación de entrada. Observamos que en el caso de una ROM no es necesario minimizar la función o funciones a implementar, ya que el plano de salida hace la unión de términos mínimos.

**EJEMPLO 3.15**

Identificar, por medio de su tabla verdad y su expresión canónica, la función  $s_7$  sintetizada por la memoria ROM de la Figura 3.17.

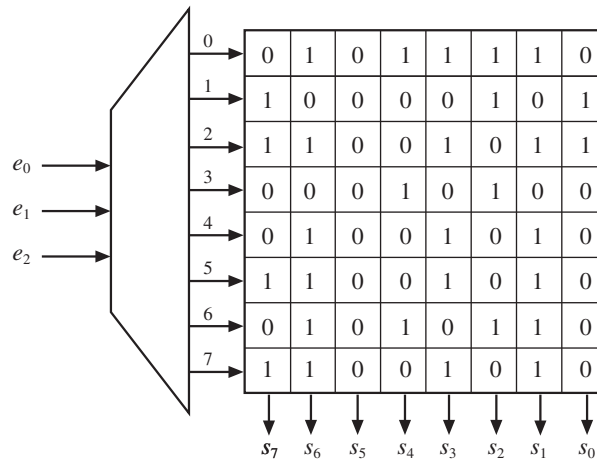
**SOLUCIÓN**

Del contenido de la memoria se deduce lo que se indica en la Tabla 3.11.

Entradas			Funciones de salida			
$e_2$	$e_1$	$e_0$	$s_3$	$s_2$	$s_1$	$s_0$
0	0	0	1	0	1	1
0	0	1	1	1	1	0
0	1	0	0	1	0	1
0	1	1	1	0	1	1
1	0	0	0	1	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	1	1	0	1	1

**Tabla 3.10.** Funciones del Ejemplo 3.14.





**Figura 3.17.** Ejemplo de memoria ROM de 8 funciones de 3 variables.

	$e_2 e_1 e_0$		Palabra seleccionada		$s_7$
Para la entrada:	0 0 0	Se selecciona la palabra:	0	Cuyo primer bit es:	0
Para la entrada:	0 0 1	Se selecciona la palabra:	1	Cuyo primer bit es:	1
Para la entrada:	0 1 0	Se selecciona la palabra:	2	Cuyo primer bit es:	1
Para la entrada:	0 1 0	Se selecciona la palabra:	3	Cuyo primer bit es:	0
Para la entrada:	1 0 0	Se selecciona la palabra:	4	Cuyo primer bit es:	0
Para la entrada:	1 0 1	Se selecciona la palabra:	5	Cuyo primer bit es:	1
Para la entrada:	1 1 0	Se selecciona la palabra:	6	Cuyo primer bit es:	0
Para la entrada:	1 1 1	Se selecciona la palabra:	7	Cuyo primer bit es:	1

**Tabla 3.11.** Ejemplo 3.15.

Con lo que la tabla verdad es la que se indica en la Tabla 3.12.

$e_2 e_1 e_0$	$s_7$
0 0 0	0
0 0 1	1
0 1 0	1
0 1 0	0
1 0 0	0
1 0 1	1
1 1 0	0
1 1 1	1

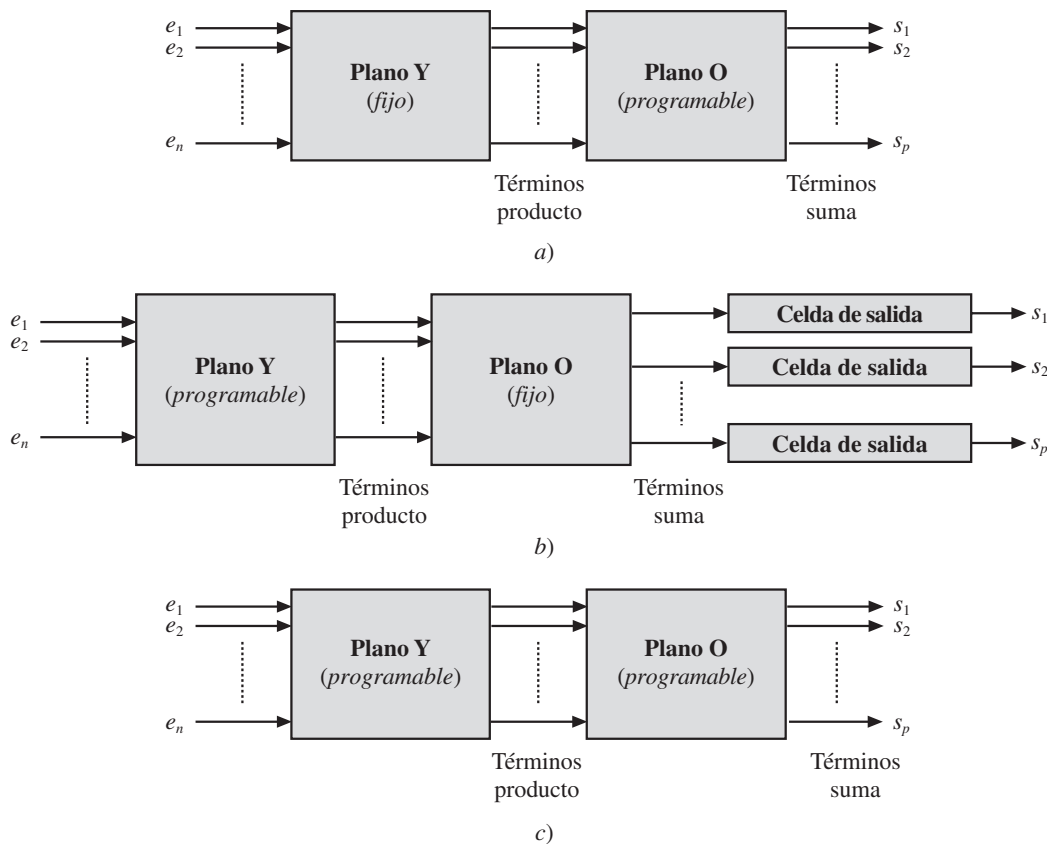
**Tabla 3.12.** Tabla verdad almacenada como función  $s_7$  en la ROM de la Figura 3.13.

En consecuencia, aplicando el Teorema de Shannon, la forma canónica de  $s_7$  es:

$$s_7 = e_2' e_1' e_0 + e_2' e_1 e_0' + e_2 e_1' e_0 + e_2 e_1 e_0$$

Los primeros SPLD de uso extendido fueron los PAL<sup>1</sup>, a comienzo de la década de los setenta. En los **PAL** (*Programmable Array Logic*) se programa el plano de entrada (AND), mientras que el

<sup>1</sup> PAL es una marca registrada de AMD.

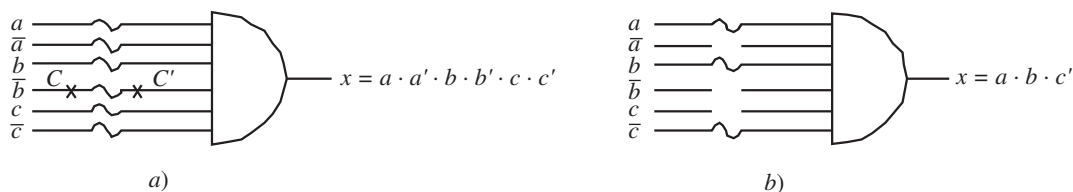


**Figura 3.18.** Esquemas simplificados de dispositivos lógicos configurables sencillos (SPLD): a) PROM; b) PAL, y c) PLA.

de salida (OR) permanece fijo (Figura 3.18b). Existen diversas técnicas de programación de conexiones, la más primitiva es rompiendo fusibles. En la Figura 3.19 se muestra como ejemplo la obtención del término  $a \cdot b \cdot c'$  a la salida de una puerta AND, mediante la destrucción de los fusibles de las entradas  $a'$ ,  $b'$  y  $c$ . La destrucción de fusibles se hace desde el exterior, aplicando intensidades de corriente tales que la conexión adquiriera una temperatura suficientemente elevada como para fundir el fusible.

En los PLA (*Programmable Logic Array*) se puede programar tanto el plano de entrada como el de salida (Figura 3.18c).

Existen otros circuitos lógicos programables por el usuario más sofisticados, como los **dispositivos lógicos configurables complejos** o **CPLD** (*Complex PLD*) y los **conjuntos de puertas configurables in situ** o **FPGA** (*Field Programmable Gate Arrays*), que incluyen tanto circuitos



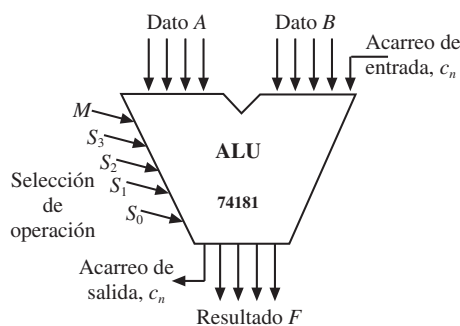
**Figura 3.19.** Programación de un SPLD de fusibles: a) situación inicial; b) situación después de haber roto los fusibles de las entradas  $a'$ ,  $b'$  y  $c$ .

combinacionales como secuenciales, y con ellos se pueden implementar circuitos de gran complejidad, como puede ser la totalidad del procesador de un computador.

### 3.1.5.8. Unidades aritmético-lógicas (ALU)

Una unidad aritmético-lógica, en el contexto de circuitos de conmutación, es un circuito que realiza múltiples operaciones aritméticas y lógicas. Este tipo de circuitos se suelen representar como se indica en la Figura 3.20.

Como ejemplo se puede considerar el circuito integrado 74181 (Figura 3.20), que dispone de entradas para dos datos  $A$  y  $B$  de cuatro bits y una salida,  $F$ , de cuatro bits. Adicionalmente dispone de una entrada de acarreo ( $c_n$ ) y una salida del acarreo ( $c_{n+1}$ ); estas líneas son para poder conectar en cascada varios ALU con objeto de operar con datos de más de cuatro bits. Por ejemplo, se puede obtener una ALU de 32 bits acoplando 8 ALU 74181. La operación concreta que realiza la ALU se determina con cinco señales de control:  $M$ ,  $S_0$ ,  $S_1$ ,  $S_2$  y  $S_3$ . La señal  $M$  indica si la operación va a ser de tipo aritmético ( $M = 0$ ) o lógico ( $M = 1$ ). En la Tabla 3.13 se muestran las distintas operaciones que implementa esta ALU.



**Figura 3.20.** Representación de una ALU de cuatro bits (74181).

Control					Funciones de salida	
$s_3$	$s_2$	$s_1$	$s_0$		$M = 0$ (operación aritmética)	$M = 1$ (operación lógica)
0	0	0	0		$F = A$ más $C_n$	$F_i = A_i'$
0	0	0	1		$F = (A + B)$ más $C_n$	$F_i = A_i' + B_i'$
0	0	1	0		$F = (A + B')$ más $C_n$	$F_i = A_i' + B_i$
0	0	1	1		$F = -1$ más $C_n$	$F_i = 0$
0	1	0	0		$F = A$ más $(A \cdot B')$ más $C_n$	$F_i = A_i' \cdot B_i'$
0	1	0	1		$F = (A + B)$ más $(A \cdot B')$ más $C_n$	$F_i = B_i'$
0	1	1	0		$F = A$ menos $B$ menos 1 más $C_n$	$F_i = A_i \cdot B_i' + A_i' \cdot B_i$
0	1	1	1		$F = (A \cdot B')$ menos 1 más $C_n$	$F_i = A_i \cdot B_i'$
1	0	0	0		$F = A$ más $(A \cdot B)$ más $C_n$	$F_i = A_i' \cdot B_i$
1	0	0	1		$F = A$ más $B$ más $C_n$	$F_i = A_i \cdot B_i + A_i' \cdot B_i'$
1	0	1	0		$F = (A + B')$ más $(A \cdot B)$ más $C_n$	$F_i = A \cdot B_i$
1	0	1	1		$F = (A \cdot B)$ menos 1 más $C_n$	$F_i = A_i \cdot B_i$
1	1	0	0		$F = A$ más $A$ más $C_n$	$F_i = 1$
1	1	0	1		$F = (A + B)$ más $A$ más $C_n$	$F_i = A_i + B_i'$
1	1	1	0		$F = (A + B')$ más $A$ más $C_n$	$F_i = A_i + B_i$
1	1	1	1		$F = A$ menos 1 más $C_n$	$F_i = A_i$

**Tabla 3.13.** Operaciones realizadas por la ALU 74181.

## 3.2. Sistemas secuenciales

Un **sistema secuencial**, a diferencia de otro combinacional, tiene memoria, de forma que sus salidas en un instante dado no sólo dependen de las entradas en ese momento sino también de las entradas anteriores (*historia*) del mismo. Los sistemas secuenciales más extendidos (denominados **síncronos**) se construyen, además de con puertas lógicas y circuitos combinacionales, con elementos de memoria, que estudiaremos en la Sección 3.2.1. La Sección 3.2.2 se dedica a analizar bloques secuenciales básicos, como son registros, contadores y memorias.

La *historia* del sistema queda establecida por la situación inicial en la que se encontrase el sistema cuando empezó a funcionar y por las entradas que sucesivamente haya ido recibiendo. La *historia* se representa por medio de unas variables, que se denominan **estados**. La situación inicial del sis-

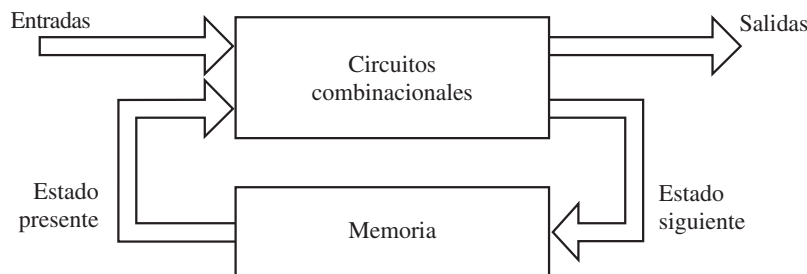
tema se encarna mediante un **estado inicial**, y las entradas en un momento dado, además de generar las salidas correspondientes, producen un cambio de estado.

Un ejemplo de sistema secuencial es el sistema de control de un ascensor. Se dispone de entradas, que son los pulsadores de los distintos pisos, y de dos salidas,  $s_0$ ,  $s_1$ , que pueden actuar de la siguiente forma:  $s_0 = 0$ ,  $s_1 = 0$ , permanecer parado;  $s_0 = 1$ ,  $s_1 = 0$ , bajar, y  $s_0 = 0$ ,  $s_1 = 1$ , subir. Supongamos que el ascensor se encuentra en un edificio de 10 plantas y se hace una entrada de llamada desde la quinta planta; entonces la salida del sistema no está determinada ya que depende de donde se encuentre actualmente el ascensor: si está en una planta superior a la quinta, la salida debe ser  $s_0 = 1$ ,  $s_1 = 0$  (bajar), si ya está en la quinta planta debe ser  $s_0 = 0$ ,  $s_1 = 0$  (permanecer parado), y si está por debajo debe ser  $s_0 = 0$ ,  $s_1 = 1$  (subir). Para diseñar el sistema debemos incluir unas variables de estado que codifiquen la planta donde se encuentra actualmente, y proporcionar las salidas en función de dichas variables de estado y de la entrada.

Así como en los sistemas combinacionales un sistema queda caracterizado por las funciones de conmutación de las salidas en función de las entradas en ese momento, un sistema secuencial se suele describir por dos tipos de expresiones que proporciona:

- las salidas en función de las entradas y el estado actual, y
- el estado siguiente, en función de las entradas y el estado actual.

De la descripción anterior se deduce que el sistema debe disponer de una memoria para almacenar el estado. Un modelo estructural de un sistema secuencial es el que se muestra en la Figura 3.21.



**Figura 3.21.** Modelo estructural para un sistema secuencial.

Puede apreciarse que tanto las salidas como los estados siguientes son generados por circuitos combinacionales, siéndoles de aplicación todo lo visto en la Sección 3.1. Nos encontramos como componentes nuevos con los elementos de memoria, que son objeto de la sección siguiente.

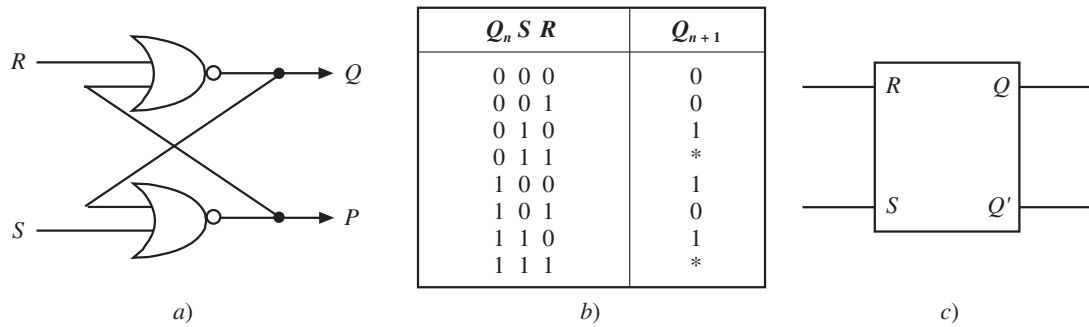
Existen procedimientos sistemáticos para diseñar sistemas secuenciales que caen fuera de las pretensiones de este libro; remitimos al lector interesado en el tema a las siguientes referencias bibliográficas: [5, 12].

### 3.2.1. ELEMENTOS DE MEMORIA. BIESTABLES

Un **elemento de memoria** es un circuito capaz de almacenar un bit, existiendo distintos circuitos que ofrecen distintos comportamientos. Los elementos de memoria también se conocen con el nombre de **biestables** o **flip-flops**. Un elemento de memoria se puede obtener realimentando dos puertas lógicas (NAND o NOR), tal y como se muestra en la Figura 3.22.

En el circuito de la Figura 3.22, denominado **biestable RS**, se verifica lo siguiente:

$$\begin{aligned} Q &= (R + P)' \\ P &= (S + Q)' \end{aligned}$$



**Figura 3.22.** Biestable RS: a) circuito; b) tabla verdad, y c) símbolo.

Si las dos entradas del circuito ( $R$  y  $S$ ) son 0; es decir, están inactivas,  $Q = P'$  y  $P = Q'$ ; es decir, ambas salidas son complementarias.

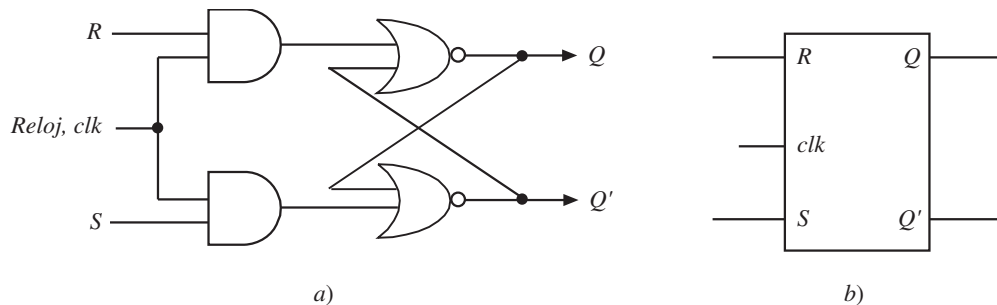
Si  $R = 1$  y  $S = 0$ , se hará  $Q = 0$  y, por tanto,  $P = 1$ ; diciéndose entonces que el circuito ha almacenado un 0, ya que si a continuación se hace  $S = R = 0$ , la salida  $Q$  permanecerá a 0 y la  $P$  a 1. Por el contrario, si  $R = 0$  y  $S = 1$ , se hará  $Q = 1$  y, por tanto,  $P = 0$ ; diciéndose entonces que el circuito ha almacenado un 1, ya que si a continuación se hace  $S = R = 0$ , la salida  $Q$  permanecerá a 1 y la  $P$  a 0. Obsérvese que siempre la salida  $P$  es complementaria a la  $Q$ , por lo que se suele denominar  $Q'$ .

En el caso de que tanto  $S$  como  $R$  sean 1, el comportamiento del circuito no queda claramente determinado, y esta situación la denotamos con un asterisco en la Figura 3.22b.

La representación esquemática de un biestable RS se muestra en la Figura 3.22c.

Conviene hacer notar que el circuito estudiado, al igual que en todos los elementos de memoria que veremos en esta sección, el estado coincide con la salida, y ésta toma el valor 0 o 1; siendo este el motivo de porque a estos circuitos se les denomina **biestables** o **básculas**; cuando se produce un cambio del estado (salida) se dice que el circuito *bascula*.

Los elementos de memoria son más útiles si se les añade una entrada adicional de control, denominada **entrada de reloj** ( $clk$ , abreviadamente, de *clock*) que habilite o no la aplicación de las entradas al circuito realimentado. En otras palabras, si  $clk = 0$  no se pueden producir cambios de estado, y al llegar un pulso  $c$ , sí se producen. Este efecto se logra añadiendo dos puertas AND en las entradas de un biestable SR, tal como muestra la Figura 3.23a. Estos circuitos, con entrada de reloj, se denominan **cerrojos** (*latch*) o **biestables sincronizados**.



**Figura 3.23.** Biestables RS cerrojo: a) circuito y b) símbolo.

Otras variantes de elementos de memoria lo constituyen los biestables  $D$ ,  $JK$  y  $T$ , cuyos circuitos y tablas verdad se muestran en la Figura 3.24.

En los **biestables cerrojo** considerados, las salidas cambian cuando lo hacen las entradas y el nivel de la señal de reloj es alto ( $clk = 1$ ). Este tipo de funcionamiento puede presentar problemas siempre que las entradas cambien indeseablemente antes de que  $clk$  pase a 0. Para evitarlo se han diseñado **biestables sincronizados** o **disparados por flanco**, en los que las entradas sólo actúan

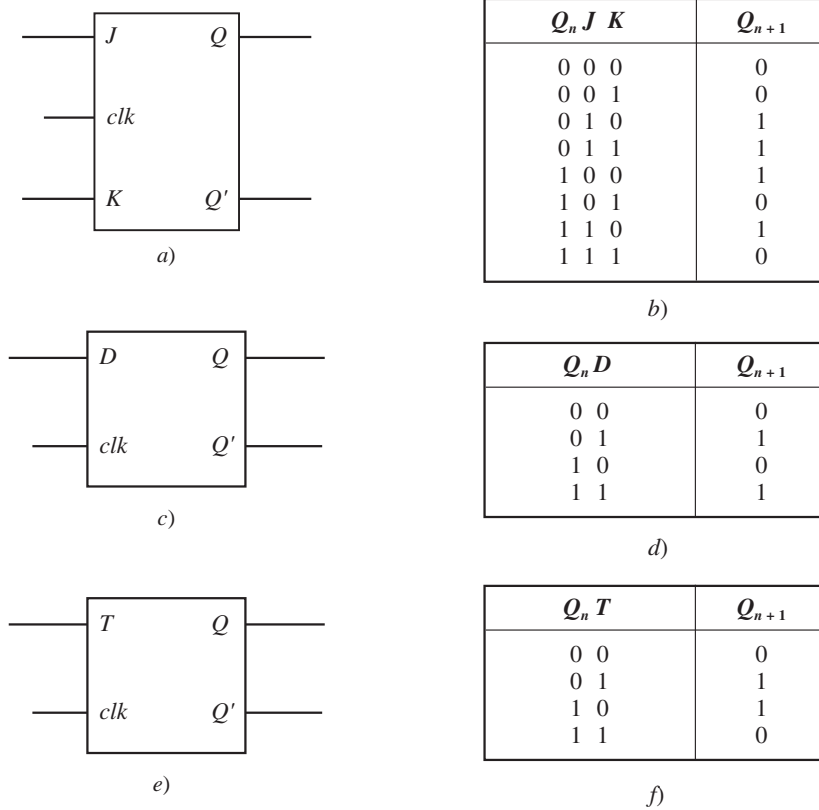


Figura 3.24. Biestables  $JK$ ,  $D$  y  $T$ ; símbolos y tablas de comportamiento.

en el instante en que la señal de reloj pasa de 0 a 1 (flanco de subida) o pasa de 1 a 0 (flanco de bajada).

### 3.2.2. BLOQUES SECUENCIALES BÁSICOS

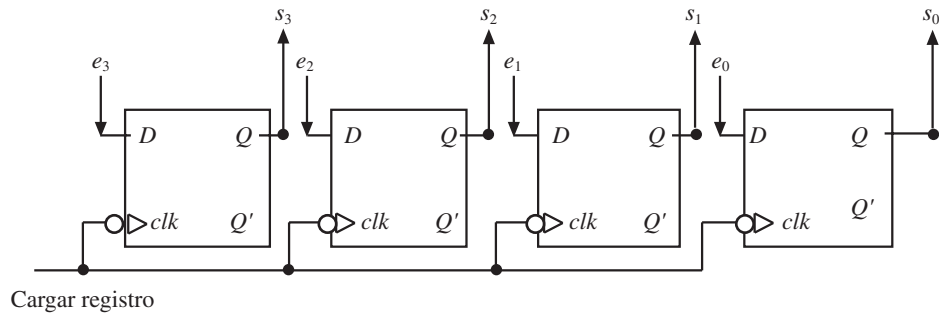
De la misma forma que ocurre con los circuitos combinacionales, podemos establecer una biblioteca de componentes o bloques secuenciales básicos, que son de uso común. Entre ellos se encuentran los registros, contadores, bancos de registros y memorias RAM.

#### 3.2.2.1. Registros

Un registro es una memoria que almacena un dato o una instrucción; es decir, una memoria de un número muy limitado de bits. Usualmente el tamaño de los registros es de 4, 8, 32, 64 o 128 bits.

Un registro de  $n$  bits se puede formar combinando adecuadamente  $n$  biestables, interconectando entre sí las señales de control. En la Figura 3.25 se muestra un registro de 4 bits, formado por 4 biestables  $D$ . Si el símbolo del circuito tiene en la entrada de reloj un triángulo indica que el biestable actuará con el flanco de subida del pulso de reloj, si además, como es el caso de la Figura 3.25, contiene un círculo, actuará con el flanco de bajada.

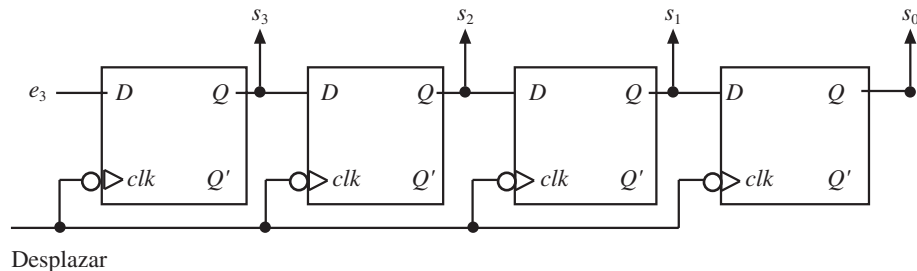
En el circuito del registro se observa que todas las entradas de reloj están unidas, y la señal común a todas se denomina **cargar**, ya que cuando llega un pulso los bits que hubiese en las entradas ( $e_3, e_2, e_1, e_0$ ) se almacenan en los biestables, apareciendo por tanto en las salidas ( $s_3, s_2, s_1, s_0$ ). El es-



**Figura 3.25.** Ejemplo de registro de 4 bits, con carga en paralelo.

tado (y las salidas) no cambian, a no ser que se dé una nueva orden de cargar y hayan cambiado las entradas, o se dé una orden de puesta a cero. La orden de puesta a cero, como su nombre indica, pone a 0 todos los biestables.

Con frecuencia se utilizan otros registros más sofisticados. Entre ellos se encuentran los **registros de desplazamiento**. Un ejemplo sencillo es el de la Figura 3.26. La entrada  $e_s$  se denomina entrada serie. Cuando llega un pulso de reloj, con su flanco de bajada, la salida del biestable de orden 1 se memoriza en el biestable 0, la del orden 2 se almacena en el 1, la del 3 en el 2, y el valor que hubiese en la entrada serie se almacena en el biestable de orden 3.



**Figura 3.26.** Ejemplo de registro de desplazamiento.

Un ejemplo de **registro de desplazamiento con carga en paralelo** está constituido por  $n$  etapas, y por ejemplo puede tener las siguientes señales de control (Figura 3.27):

- **Reloj ( $clk$ ):** el contenido del registro sólo puede cambiar cuando llega un pulso de reloj.
- **Puesta a cero ( $R$ , de *Reset*):** los estados y salidas de las  $n$  etapas se ponen a 0.
- **Carga/Desplazamiento ( $S$ ):** si esta señal es cero, cuando llegue un pulso de reloj se almacena en el registro la información que hubiese en la entrada. Si  $S = 1$ , al llegar el pulso de reloj se efectúa un desplazamiento de la información almacenada en el registro, a derecha o izquierda, según sea la señal  $L/R'$ .
- **Tipo de desplazamiento ( $L/R'$ ):** si esta señal es 0 (y siempre que  $S = 1$ ) se produce un desplazamiento a la derecha, y si es 1 a la izquierda. Un desplazamiento a la derecha hace que el bit en posición 0 se pierda, el que hay en posición 1 pase a la posición 0, el de la 2 pase a la 1, y así sucesivamente, el de la  $n - 1$  a la  $n - 2$ . El valor que haya en la entrada serie ( $e_{MSB}$ ) pasa a la posición  $n - 1$ . En un desplazamiento a la izquierda se hace el proceso a la inversa: el bit en posición  $n - 1$  se pierde, el de posición  $n - 2$  pasa a la  $n - 1$ , ..., el de posición 0 pasa a la 1, y el valor de la entrada serie de la derecha ( $e_{LSB}$ ) se almacena en la posición 0.

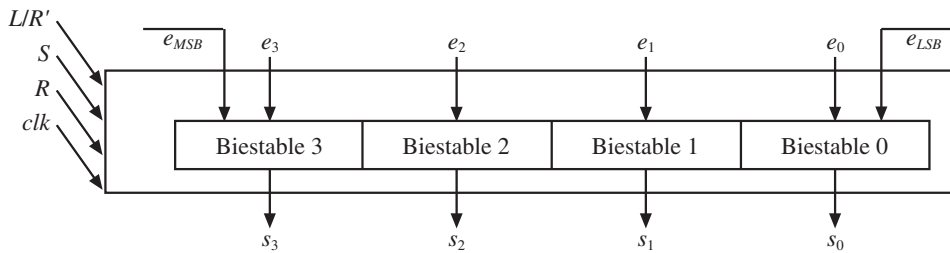


Figura 3.27. Representación simplificada de un registro de desplazamiento con carga en paralelo.

### 3.2.2.2. Contadores

Un contador binario es un circuito que a su salida proporciona una secuencia repetitiva de valores. Así, por ejemplo, un contador binario sincronizado de  $n = 4$  bits puede generar la siguiente secuencia:

0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111  
 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111  
 (a partir de aquí se repite la secuencia)  
 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111  
 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111, etc.

En decimal, los valores obtenidos serán: 0, 1, 2, ..., 14, 15, 0, 1, ..., 14, 15, 0, 1... Al ser sincronizado, el contador avanzará siempre que llegue un pulso de reloj. Con  $n$  biestables podemos construir un contador módulo  $2^n$ .

Se pueden diseñar contadores:

- Que sigan una secuencia de valores predeterminada, no necesariamente valores sucesivos.
- Que cuenten ascendente y descendente.
- Con carga en paralelo; es decir, que se pueda dar un valor inicial a los biestables del contador.
- Con puesta a cero, de forma que con una señal de control la salida del contador se ponga a cero.

#### EJEMPLO 3.16

El **puntero de instrucciones** (IP) o contador de programa (PC) de un procesador se suele implementar como un contador. Este contador debe de disponer de las siguientes señales de control:

- Puesta a cero: cuando se arranca el computador el puntero.
- Carga en paralelo.
- Avance.

### 3.2.2.3. Bancos de registros

Un **banco de registros** (*register file*) es un conjunto de registros direccionables. El acceso a la información en un banco de registros es mucho más rápido que en la memoria principal del sistema, por eso es habitual que en los procesadores se incluyan uno o varios bancos de registros, donde almacenar temporalmente los datos o resultados intermedios más utilizados por un programa.

En la Figura 3.28 se muestra una representación de un ejemplo de banco de registros. Un banco de registros puede considerarse

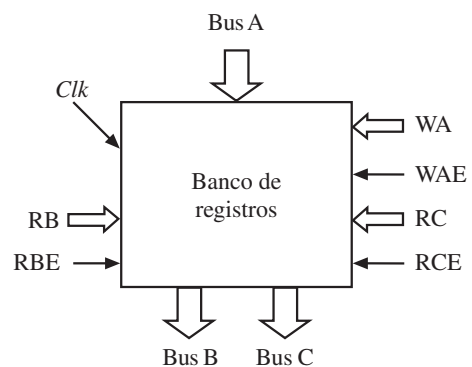


Figura 3.28. Representación esquemática de un banco de registros.



como una tabla o estructura bidimensional, donde cada fila es un registro compuesto de  $m$  celdas constituidas por biestables  $D$ . Las lecturas y escrituras se hacen por grupos de  $m$  bits (*palabra*) que se almacenan o leen en cada uno de sus  $2^n$  registros. El acceso a un registro concreto, para leer o escribir, se realiza dando su dirección ( $n$  bits). Hay bancos de registros que permiten hacer lecturas y escrituras simultáneas; así, en el ejemplo de la Figura 3.28 se permiten realizar simultáneamente dos lecturas y una escritura.

Los terminales que contiene el banco de registros de la Figura 3.27 son:

- Entrada/salida de datos:
  - Entrada de datos: bus A, de  $m$  líneas.
  - Salidas de datos: buses B y C, de  $m$  líneas cada uno de ellos.
- Direccionamiento:
  - Dirección de registro donde escribir la información que se encuentre en el bus A: WA, de  $n$  líneas.
  - Dirección de registro a leer a través del bus B: RB, de  $n$  líneas.
  - Dirección de registro a leer a través del bus C: RC, de  $n$  líneas.
- Señales de control:
  - WE: habilitación de escritura.
  - RBE: habilitación de lectura a través del bus B.
  - RCE: habilitación de lectura a través del bus C.
  - *Clk*: señal de reloj, con el flanco de subida se almacena, en el registro de dirección indicada en WA, los bits que hubiese en el bus A.

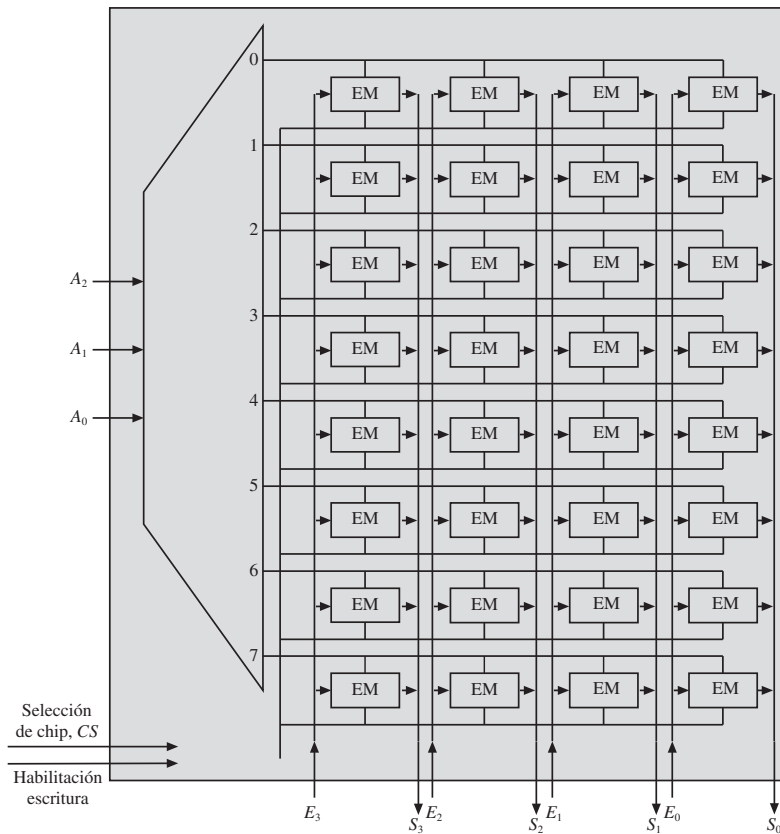
Tamaños usuales de bancos de registros son de 8, 16, 32, 62 o 128 registros de 8, 16, 32, 64 o 128 bits cada uno.

### 3.2.2.4. Memoria RAM

Una memoria RAM puede considerarse como un banco de registros de una gran capacidad; en efecto, el número de bits para seleccionar la dirección de palabra suele ser 16 o 32 bits, con lo que la capacidad máxima direccionable es de  $2^{16} = 64$  K palabras o  $2^{32} = 4$  G palabras.

Con objeto de comprender mejor cómo funciona una memoria RAM, en la Figura 3.29 se muestra una de tan sólo 8 palabras de 4 bits. Cada fila representa una palabra de memoria. Los terminales de que dispone son:

- *Entrada/salida de datos*. Aunque en la figura se muestran separadas las entradas y salidas, habitualmente se utilizan las mismas líneas para entradas y salidas, separándose internamente. En la figura se tiene:
  - Entrada de datos: líneas  $E_3, E_2, E_1, E_0$ .
  - Salidas de datos: líneas  $S_3, S_2, S_1, S_0$ . Las salidas son tri-estado; cuando la señal de control  $CS = 1$ , las líneas de salida se ponen en estado de baja impedancia.
- *Direccionamiento*:
  - En la figura el direccionamiento se hace con las líneas  $A_2, A_1, A_0$ . Éstas actúan como entradas de un decodificador que selecciona la palabra direccionada (palabra 0, 1, 2, 3, ..., 7; en la figura).
- *Señales de control*:
  - Habilitación de escritura, cuando es 1 se escribe la información que haya en  $E_3, E_2, E_1, E_0$  en la palabra direccionada por  $A_2, A_1, A_0$ .



**Figura 3.29.** Representación esquemática y simplificada de una memoria RAM de 8 palabras de 4 bits.

— *CS (Chip Select)*. Esta señal debe ser uno para que el circuito actúe; es decir, para que sus líneas de salida pasen al estado de baja impedancia y se realicen las operaciones de escritura. La señal *CS* hace posible conectar distintos chips para configurar distintos tamaños de memoria.

Cada celda o elemento de memoria (EM, en la figura) es un biestable *D*, pero éste ocupa mucha superficie con lo que la capacidad de memoria de un chip no sería muy alta. Como se indicó en el Capítulo 1 (Sección 1.2.2), las prestaciones de una memoria se miden por su capacidad y su tiempo de acceso, cualidades que por lo general son contrapuestas. Las memorias RAM se suelen clasificar en dos tipos:

- **Memorias RAM estáticas (o SRAM):** cada celda contiene un circuito con tan sólo de 5 a 6 transistores, manteniendo su información mientras esté conectada a la energía eléctrica.
- **Memorias RAM dinámicas (o DRAM):** cada celda contiene un solo transistor para almacenar el bit de información (además de otro de selección de celda). Son las memorias con las que se obtiene una mayor densidad de integración, pero son más lentas que las SRAM. Cada transistor de memorización se comporta como un pequeño condensador, almacenándose los ceros y unos en forma de presencia o ausencia de carga eléctrica. Una vez que se carga un condensador, almacenando un 1 (por ejemplo), se descarga lentamente por corrientes de pérdidas, por lo que es necesario recargarlo (*refrescarlo*) periódicamente para que se mantenga su carga y pueda seguir siendo reconocida como un 1. Las operaciones periódicas de refresco hacen que funcionen más lentamente que las SRAM.

Usualmente la memoria RAM (de capacidad relativamente alta) se implementa con circuitos DRAM; sin embargo, la caché, que requiere mayor velocidad se implementa con circuitos SRAM (Sección 5.1.2).

### 3.3. Conclusiones

En este capítulo hemos estudiado las bases de la lógica digital como herramienta para diseñar los elementos constructivos de las distintas unidades funcionales del computador.

En primer lugar hemos presentado (Secciones 3.1.1 y 3.1.2) los principios del álgebra de conmutación, que proporciona la herramienta matemática adecuada para el diseño de sistemas digitales.

Hemos clasificado los sistemas digitales en combinacionales y secuenciales, en función de que no tengan o tengan memoria, y hemos analizado las técnicas para que a partir de una descripción del funcionamiento de un sistema combinacional se pueda diseñar el mismo (Secciones 3.1.3 y 3.1.4). También se han descrito (Sección 3.1.5) los módulos combinacionales básicos utilizados en las distintas unidades de un computador (circuitos aritméticos, codificadores, decodificadores, multiplexores, etc.).

Por lo que respecta a los sistemas secuenciales, hemos analizado (Sección 3.2.1) el funcionamiento de distintos elementos de memoria o biestables, y, por último (Sección 3.2.2), se han descrito, desde un punto de vista funcional, los registros, los contadores, los bancos de registros y las memorias RAM, que pueden considerarse, junto con los biestables, los módulos básicos de los sistemas secuenciales.



#### Test

**T3.1.** Un sistema digital utiliza:

- a) Valores discretos (2, 3 o más) para representar las variables.
- b) Ceros y unos para codificar físicamente las variables y valores.
- c) Dos valores para representar las variables.
- d) Los dígitos decimales (0, 1, ..., 9) para representar las variables.

**T3.2.** Un sistema binario:

- a) Utiliza valores discretos (2, 3 o más) para representar las variables.
- b) Utiliza ceros y unos para codificar físicamente las variables y valores.
- c) Utiliza dos valores para representar las variables.
- d) Utiliza los dígitos decimales (0, 1, ..., 9) para representar las variables.

**T3.3.** El valor de la expresión  $x \cdot x'$  es:

- a) 1.
- b) 0.
- c)  $x$ .
- d) Ninguna de las otras alternativas, ya que depende del valor que se dé a la variable  $x$ .

**T3.4.** El valor de la expresión  $x + x'$  es:

- a) 1.
- b) 0.

c)  $x$ .

d) Ninguna de las otras alternativas, ya que depende del valor que se dé a la variable  $x$ .

**T3.5.** El valor de la expresión  $x + x$  es:

- a) 1.
- b) 0.
- c)  $x$ .
- d) Ninguna de las otras alternativas, ya que depende del valor que se dé a la variable  $x$ .

**T3.6.** El valor de la expresión  $x \cdot x$  es:

- a) 1.
- b) 0.
- c)  $x$ .
- d) Ninguna de las otras alternativas, ya que depende del valor que se dé a la variable  $x$ .

**T3.7.** El valor de la expresión  $x + xy$  es:

- a)  $y$ .
- b) 0.
- c)  $x$ .
- d) Ninguna de las otras alternativas, ya que depende del valor que se dé a las variables  $x$  e  $y$ .

**T3.8.** El valor de la expresión  $x \cdot (x + y)$  es:

- a) 1.
- b) 0.

- c)  $x$ .
- d)  $x + y$ .

**T3.9.** El valor de la expresión  $(x + y)'$  es:

- a) 1.
- b)  $x' \cdot y'$ .
- c)  $x' + y'$ .
- d)  $x \cdot y$ .

**T3.10.** El valor de la expresión  $(x \cdot y)'$  es:

- a) 1.
- b)  $x' \cdot y'$ .
- c)  $x' + y'$ .
- d) 0.

**T3.11.** El valor de la expresión  $x + x' \cdot y$  es:

- a) 1.
- b)  $x \cdot y$ .
- c)  $x + y$ .
- d) 0.

**T3.12.** El valor de la expresión  $x \cdot (x' + y)$  es:

- a) 1.
- b)  $x \cdot y$ .
- c)  $x + y$ .
- d) 0.

**T3.13.** La función cuya salida es 1 si y sólo si todas las entradas son 1 es la:

- a) AND.
- b) OR.
- c) NAND.
- d) NOR.

**T3.14.** La función cuya salida es 0 si y sólo si todas las entradas son 0 es la:

- a) AND.
- b) OR.
- c) NAND.
- d) NOR.

**T3.15.** La función cuya salida es 0 si y sólo si todas las entradas son 1 es la:

- a) AND.
- b) OR.
- c) NAND.
- d) NOR.

**T3.16.** La función cuya salida es 1 si y sólo si todas las entradas son 0 es la:

- a) AND.
- b) OR.
- c) NAND.
- d) NOR.

**T3.17.** La función cuya salida es 1 si y sólo si es impar el número de entradas que están a 1 es la:

- a) NAND.
- b) NOR.
- c) EXOR.
- d) EXNOR.

**T3.18.** La función cuya salida es 1 si y sólo si es par el número de entradas que están a 1 es la:

- a) NAND.
- b) NOR.
- c) EXOR.
- d) EXNOR.

**T3.19.** El resultado de una determinada operación lógica entre dos variables con valores:  $a = 0$  y  $b = 1$ , es  $z = 0$ ; dicha operación puede ser:

- a)  $a \cdot b$ .
- b)  $a + b$ .
- c)  $a \oplus b$ .
- d)  $(a \cdot b)'$ .

**T3.20.** El resultado de una determinada operación lógica entre dos variables con valores:  $a = 0$  y  $b = 1$ , es  $z = 1$ ; dicha operación puede ser:

- a)  $a \cdot b$ .
- b)  $a + b$ .
- c)  $(a + b)'$ .
- d)  $(a \oplus b)'$ .

**T3.21.** El resultado de una determinada operación lógica entre dos variables con valores:  $a = 1$  y  $b = 1$ , es  $z = 0$ ; dicha operación puede ser:

- a)  $a \cdot b$ .
- b)  $a + b$ .
- c)  $(a \oplus b)'$ .
- d)  $(a \cdot b)'$ .

**T3.22.** El resultado de una determinada operación lógica entre dos variables con valores:  $a = 1$  y  $b = 1$ , es  $z = 1$ ; dicha operación puede ser:

- a)  $a \oplus b$ .
- b)  $(a \cdot b)'$ .
- c)  $(a + b)'$ .
- d)  $(a \oplus b)'$ .

**T3.23.** El resultado de una determinada operación lógica entre dos variables con valores:  $a = 0$  y  $b = 0$ , es  $z = 0$ ; dicha operación puede ser:

- a)  $a \oplus b$ .
- b)  $(a \cdot b)'$ .
- c)  $(a + b)'$ .
- d)  $(a \oplus b)'$ .

**T3.24.** El resultado de una determinada operación lógica entre dos variables con valores:  $a = 0$  y  $b = 0$ , es  $z = 1$ ; dicha operación puede ser:

- a)  $a \cdot b$ .
- b)  $a + b$ .
- c)  $a \oplus b$ .
- d)  $(a \cdot b)'$ .

**T3.25.** Un sistema secuencial de  $n$  salidas es un sistema que queda definido por:

- a) Sus salidas en función de las entradas en ese instante y las  $2^n$  entradas anteriores.
- b) Sus salidas en función de entradas y estado presente, y su estado siguiente en función de entradas y estado presente.
- c) Las salidas siguientes en función de las entradas en ese instante y salidas en el instante anterior.
- d) Las salidas en el estado presente (no tienen memoria).

**T3.26.** Un circuito de  $n$  salidas y  $2^n$  entradas y que a la salida proporciona el valor binario correspondiente al orden (0 a  $n - 1$ ) de la única entrada que esté a uno, se denomina:

- a) Demultiplexor.
- b) Codificador binario.
- c) Decodificador binario.
- d) Multiplexor.

**T3.27.** Un circuito con  $n$  entradas y  $2^n$  salidas que activa únicamente la salida correspondiente al valor binario en la entrada (0, ...,  $n$ ), se denomina:

- a) Demultiplexor.
- b) Codificador binario.
- c) Decodificador binario.
- d) Multiplexor.

**T3.28.** Un circuito con una entrada de datos,  $n$  señales de control y  $2^n$  salidas, y que proporciona la entrada en la salida seleccionada por las señales de control, se denomina:

- a) Demultiplexor.
- b) Codificador binario.
- c) Decodificador binario.
- d) Multiplexor.

**T3.29.** Un circuito con  $2^n$  entradas de datos,  $n$  señales de control y una salida, y que proporciona a la salida la entrada seleccionada por las señales de control, se denomina:

- a) Demultiplexor.
- b) Codificador binario.
- c) Decodificador binario.
- d) Multiplexor.

**T3.30.** Para conectar varios elementos a un mismo bus es necesario utilizar:

- a) Un demultiplexor.
- b) Un multiplexor.
- c) Un decodificador.
- d) Un multiplexor o los circuitos que se conecten deben tener salida tri-estado.

**T3.31.** Considérese un bloque combinacional formada por dos planos (matrices), uno AND de entrada, que sintetiza términos mínimos, y otro OR de salida que suma los términos mínimos previamente sintetizados para obtener a la función. Si el plano AND es fijo y el plano OR programable, el bloque se denomina:

- a) ROM.
- b) PLA.
- c) PAL.
- d) Multiplexor.

**T3.32.** Considérese un bloque combinacional formada por dos planos (matrices), uno AND de entrada, que sintetiza términos productos, y otro OR de salida que suma los productos previamente sintetizados para obtener la función. Si el plano AND es programable y el plano OR fijo, el bloque se denomina:

- a) ROM.
- b) PLA.
- c) PAL.
- d) Multiplexor.

**T3.33.** Considérese un bloque combinacional formada por dos planos (matrices), uno AND de entrada, que sintetiza términos productos, y otro OR de salida que suma los productos previamente sintetizados para obtener la función. Si el plano AND es programable y el plano OR fijo, el bloque se denomina:

- a) ROM.
- b) PLA.
- c) PAL.
- d) Multiplexor.

**T3.34.** Los biestables tal que cuando llega un pulso de reloj su salida,  $z$ , se hace igual a la entrada,  $x$ ; en ese momento ( $z = x$ ), independientemente del valor de la salida anterior, se denominan:

- a) RS.
- b) JK.
- c) D.
- d) T.

**T3.35.** Los biestables cuyas dos entradas síncronas,  $a$  y  $b$ , son tales que si  $a = b = 0$  la salida ( $z$ ) no cambia; si  $a = 1, b = 0$  la salida pasa a  $z = 0$ , si  $a = 0, b = 1$  la salida pasa a  $z = 1$ , y si  $a = b = 1$  no se sabe el valor que tomará la salida ( $z = *$ ), se denominan:

- a) JK.
- b) D.
- c) T.
- d) RS.

**T3.36.** Los biestables cuyas dos entradas síncronas,  $a$  y  $b$ , son tales que si  $a = b = 0$  la salida ( $z$ ) no cambia; si  $a = 1, b = 0$  la salida pasa a  $z = 0$ , si  $a = 0, b = 1$  la salida pasa a  $z = 1$ , y

si  $a = b = 1$  la salida cambia al valor complementario del que tuviese anteriormente, se denominan:

- a) RS.
- b) JK.
- c) D.
- d) T.

**T3.37.** Un biestable tal que cuando llega un pulso de reloj tal que su entrada, sea cual sea, aparece en la salida,  $z$ , complementada se denominan:

- a) RS.
- b) JK.
- c) D.
- d) T.

**T3.38.** Pueden almacenarse simultáneamente los  $p$  bits de una palabra en:

- a) Sólo en un registro con carga en paralelo.
- b) Sólo en un contador con carga en paralelo.
- c) Sólo en un banco de registros.
- d) En todos los anteriores.

**T3.39.** Un circuito de  $p$  bits que permite, por medio de una señal de control, transferir sus bits a posiciones consecutivas de mayor o menor orden se denomina:

- a) Banco de registros.
- b) Contador.
- c) Tabla de transición.
- d) Registro de desplazamiento.

## Problemas resueltos



### PRINCIPIOS DEL ÁLGEBRA DE CONMUTACIÓN

**P3.1.** Efectuar bit a bit las siguientes operaciones lógicas:

10101 <b>OR</b> 11110	10000 <b>OR</b> 11110
11101 <b>AND</b> 00110	111011 <b>AND</b> 011011
<b>NOT</b> 101100	<b>NOT</b> 001011
10110 <b>NOR</b> 11101	01110 <b>NAND</b> 01101

#### SOLUCIÓN

Teniendo en cuenta el contenido de la Tabla 3.3 se obtiene:

10101 <b>OR</b> 11110 = 11111	10000 <b>OR</b> 11110 = 11110
11101 <b>AND</b> 00110 = 00100	111011 <b>AND</b> 011011 = 011011
<b>NOT</b> 101100 = 010011	<b>NOT</b> 001011 = 110100
10110 <b>NOR</b> 11101 = 00000	01110 <b>NAND</b> 01101 = 10011

**P3.2.** Se dispone de un registro  $R$  de 8 bits. En otro registro,  $MAS$ , se puede ubicar un patrón de bits determinado por el usuario. Indicar qué patrón de bits (**máscara**) hay que ubicar en  $MAS$  y la operación lógica a realizar entre  $R$  y  $MAS$  para seleccionar los 4 bits más significativos de  $R$  (haciendo ceros los cuatro menos significativos).

#### SOLUCIÓN

Con la operación lógica AND entre 2 bits,  $a$  y  $b$ , obtenemos el valor de  $a$ , si  $b = 1$ , y 0 si  $b = 0$ ; con lo que se obtiene el comportamiento deseado haciendo la operación AND con la máscara 1111 0000. Por ejemplo:

Registro R →	0	1	0	1	1	0	1	0
Máscara →	1	1	1	1	0	0	0	0
Resultado con AND →	0	1	1	1	0	0	0	0

### MINIMIZACIÓN DE FUNCIONES

**P3.3.** Minimizar algebraicamente la siguiente función:

$$x = abc + a'bc + b'c + ab' + ab + bc$$

## SOLUCIÓN

$$\begin{aligned}
 x &= abc + a'bc + b'c + ab' + ab + bc = \\
 &= (a' + a)bc + b'c + a(b' + b) + bc = \\
 &= bc + b'c + a + bc = \\
 &= bc + b'c + a = \\
 &= c(b + b') + a = \\
 &= c + b
 \end{aligned}$$

## FORMA NORMALIZADA

**P3.4.** Obtener la forma canónica de la expresión:

$$y = a' + bc + b'c'$$

## SOLUCIÓN

Primero obtenemos su tabla verdad:

<i>a</i> <i>b</i> <i>c</i>	<i>a'</i>	<i>bc</i>	<i>b'c'</i>	<i>y</i>
0 0 0	1	0	1	1
0 0 1	1	0	0	1
0 1 0	1	0	0	1
0 1 1	1	1	0	1
1 0 0	0	0	1	1
1 0 1	0	0	0	0
1 1 0	0	0	0	0
1 1 1	0	1	0	1

Luego, aplicando el Teorema de Shannon, su forma canónica es:

$$y = a' \cdot b' \cdot c' + a' \cdot b' \cdot c + a' \cdot b \cdot c' + a' \cdot b \cdot c + a \cdot b' \cdot c' + a \cdot b \cdot c$$

**P3.5.** Sean las dos siguientes funciones de conmutación:

$$z = acd + abd' + acd' + ab + b'c$$

$$y = a(c + b) + b'c$$

*a)* Determina si son iguales.

*b)* Obtener sus formas canónicas.

## SOLUCIÓN

Podemos ver si son iguales con manipulaciones algebraicas, o comprobando si las tablas verdad de ambas funciones son iguales. Como nos piden la forma canónica, obtendremos las tablas verdad de las funciones *z* (Tabla 3.14) e *y* (Tabla 3.15).

Observamos que para las mismas combinaciones de entrada las salidas son las mismas, luego las funciones son iguales.

La forma canónica de la función se puede obtener de la tabla verdad, sumando los términos mínimos para los que la función es 1; es decir:

$$z = y = a'b'c'd' + a'b'cd + a'b'cd' + a'b'cd + a'b'cd' + a'b'cd' + a'b'cd' + a'b'cd$$

Analíticamente también podríamos haber demostrado que las dos expresiones corresponden a la misma función ( $z = y$ ). En efecto:

$$\begin{aligned}
 z &= acd + abd' + acd' + ab + b'c = \\
 &= ac(d + d') + ab(d' + 1) + b'c = \\
 &= ac + ab + b'c = \\
 &= a(c + b) + b'c = \\
 &= y
 \end{aligned}$$

$a$	$b$	$c$	$d$	$d'$	$acd$	$abd'$	$acd'$	$ab$	$b'c$	$z$
0	0	0	0	1	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	1	1
0	0	1	1	0	0	0	0	0	1	1
0	1	0	0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	0	0	0	0
0	1	1	0	1	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0
1	0	1	0	1	0	0	1	0	1	1
1	0	1	1	0	1	0	0	0	1	1
1	1	0	0	1	0	1	0	1	0	1
1	1	0	1	0	0	0	0	1	0	1
1	1	1	0	1	0	1	1	1	0	1
1	1	1	1	0	1	0	0	1	0	1

Tabla 3.14. Problema 3.5; función  $z$ .

$a$	$b$	$c$	$d$	$b'$	$b'c$	$c + b$	$a(c + b)$	$y$
0	0	0	0	1	0	0	0	0
0	0	0	1	1	0	0	0	0
0	0	1	0	1	1	1	0	1
0	0	1	1	1	1	1	0	1
0	1	0	0	0	0	1	0	0
0	1	0	1	0	0	1	0	0
0	1	1	0	0	0	1	0	0
0	1	1	1	0	0	1	0	0
1	0	0	0	1	0	0	0	0
1	0	0	1	1	0	0	0	0
1	0	1	0	1	1	1	1	1
1	0	1	1	1	1	1	1	1
1	1	0	0	0	0	1	1	1
1	1	0	1	0	0	1	1	1
1	1	1	0	0	0	1	1	1
1	1	1	1	0	0	1	1	1

Tabla 3.15. Problema 3.5; función  $y$ .

## IMPLEMENTACIÓN DE FUNCIONES

**P3.6.** Diseñar un sistema digital que tenga cuatro entradas y cuya salida sea 1 cuando en la combinación de entrada sean iguales los números de ceros y de unos.

## SOLUCIÓN

Denominamos  $a$ ,  $b$ ,  $c$  y  $d$  a las variables de entrada, y  $z$  a la función, en la tabla verdad ponemos  $z = 1$  cuando en la combinación de entrada el número de ceros sea igual al de unos; es decir, haya dos ceros y dos unos. La tabla verdad que se obtiene es la que se muestra en la Tabla 3.16.

$a$	$b$	$c$	$d$	$z$
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Tabla 3.16. Problema 3.6.

Aplicando el Teorema de Shannon se obtiene que:

$$z = a'b'c'd + a'b'c'd' + a'b'cd' + a'b'cd + a'b'cd' + a'b'cd'$$



Ahora hay que simplificar esta expresión, pero no se encuentra ninguna forma de hacerlo, con lo que pasamos a implementarla. Realizamos una implementación de dos niveles, el primero con 6 puertas AND de 4 entradas y el segundo con una puerta OR de 6 entradas. En la Figura 3.30 se muestra el esquema de circuito; por comodidad hemos omitido los inversores de entrada.

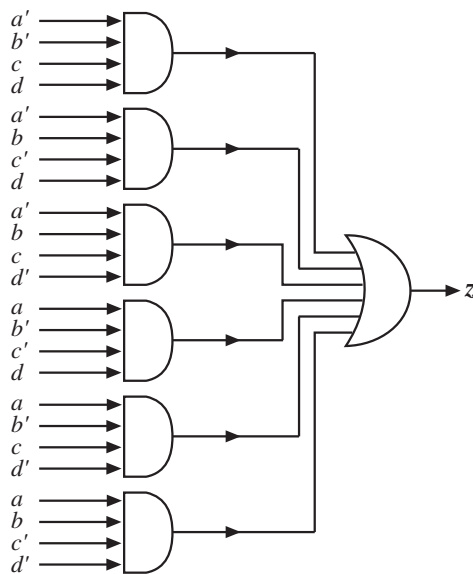
**P3.7.** Implementar el circuito del problema anterior (Problema 3.6) con tan sólo puertas NAND.

#### SOLUCIÓN

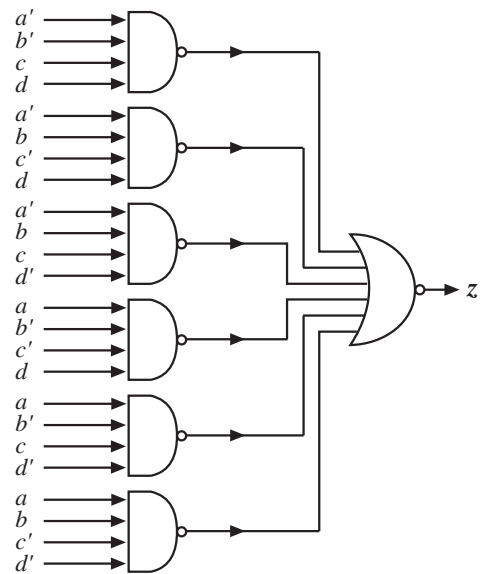
Según se ha indicada en la Sección 3.1.2 y el Ejemplo 3.10, por aplicación de la Ley de De Morgan las sumas (OR) se pueden convertir en operaciones NAND con las entradas invertidas; en efecto, como

$$x + y = (x' \cdot y')'$$

podemos cambiar el circuito OR de salida por una puerta NAND, siempre que apliquemos el complemento de las salidas de las puertas AND de la primera etapa, cosa que se consigue sin más que cambiar dichas puertas por puertas NAND, el circuito que resulta es el que se muestra en la Figura 3.31.



**Figura 3.30.** Implementación del sistema descrito en el Problema 3.6.



**Figura 3.31.** Implementación del sistema descrito en el Problema 3.6 con puertas NAND.

**P3.8.** Se dispone de dos números binarios cada uno representado por 2 bits, diseñar un sistema digital cuya salida sea 1 cuando ambos números sean iguales.

#### SOLUCIÓN

Vamos a denominar a las dos cifras del primer número ( $N$ )  $a$  y  $b$  y a las del segundo ( $M$ )  $c$  y  $d$ . La tabla verdad, que muestra todos los casos posibles, es la que se da en la Tabla 3.17.

La función de conmutación la podemos obtener aplicando el Teorema de Shannon:

$$z = a'b'c'd' + a'b'cd + ab'cd' + abcd$$

Esta función no se puede minimizar. Se puede implementar como se ha indicado en los Ejemplos 3.9 y 3.10; para lo que se necesitaría cuatro puertas AND de cuatro entradas y una OR de 4 entradas, o, alternativamente, cinco puertas NAND. No obstante, la implementación sería más sencilla con puertas EXNOR. Recuérdese (Sección 3.1.1 y Tabla 3.3) que la salida de una puerta EXNOR es 1 si sus dos en-

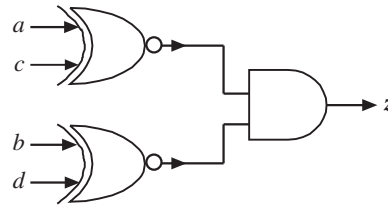
tradas son iguales. Entonces podríamos comparar  $a$  con  $c$  y  $b$  con  $d$ , y si ambas igualdades se cumplen, la función  $z$  debe ser 1. En otras palabras, la función  $z$  se puede expresar como:

$$z = (a \oplus c)' \cdot (b \oplus d)'$$

cuya implementación se da en la Figura 3.32.

$N$		$M$		$z$
$a$	$b$	$c$	$d$	
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

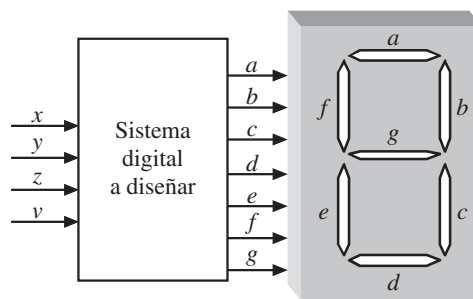
**Tabla 3.17.** Problema 3.8.



**Figura 3.32.** Implementación del sistema descrito en el Problema 3.8 con puertas EXNOR.

## BLOQUES COMBINACIONALES

**P3.9.** Implementar un circuito combinacional para visualizar en un indicador de 7 segmentos el valor hexadecimal de los cuatro bits de entrada al circuito. El indicador de 7 segmentos es un dispositivo optoelectrónico que tiene 7 fotodiodos lineales, y cada uno emite luz cuando su entrada es 1 (Figura 3.33).



**Figura 3.33.** Indicador de 7 segmentos.

## SOLUCIÓN

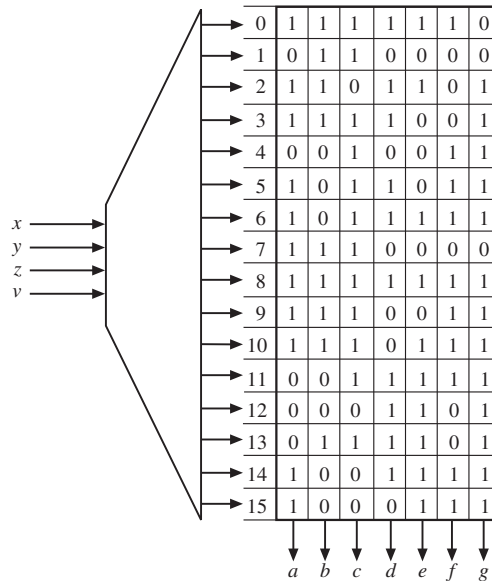
En la Tabla 3.18 se muestran los caracteres que se deben visualizar y los distintos diodos emisores de luz que deben activarse para cada combinación de entrada.

Como hay que generar simultáneamente siete funciones, lo más cómodo es utilizar una **tabla de consulta**, que es sencillamente una pequeña ROM. La información que hay que grabar en la tabla de consulta coincide con los valores de las funciones. Así, en la palabra de dirección 0 deben almacenarse los siete

$x \ y \ z \ v$	Carácter a visualizar	Segmentos a activar						
		$a$	$b$	$c$	$d$	$e$	$f$	$g$
0 0 0 0	0	1	1	1	1	1	1	0
0 0 0 1	1	0	1	1	0	0	0	0
0 0 1 0	2	1	1	0	1	1	0	1
0 0 1 1	3	1	1	1	1	0	0	1
0 1 0 0	4	0	0	1	0	0	1	1
0 1 0 1	5	1	0	1	1	0	1	1
0 1 1 0	6	1	0	1	1	1	1	1
0 1 1 1	7	1	1	1	0	0	0	0
1 0 0 0	8	1	1	1	1	1	1	1
1 0 0 1	9	1	1	1	0	0	1	1
1 0 1 0	A	1	1	1	0	1	1	1
1 0 1 1	B	0	0	1	1	1	1	1
1 1 0 0	c	0	0	0	1	1	0	1
1 1 0 1	d	0	1	1	1	1	0	1
1 1 1 0	E	1	0	0	1	1	1	1
1 1 1 1	F	1	0	0	0	1	1	1

**Tabla 3.18.** Activación de un indicador de 7 segmentos.

valores de las siete funciones para la entrada 0000; en la palabra de dirección 1, los siete valores de las siete funciones para la entrada 0001, y así sucesivamente. Los contenidos de la tabla de consulta serán los indicados en la Figura 3.34.



**Figura 3.34.** Tabla de consulta para implementar el visualizador de caracteres hexadecimales.

- P3.10.** Se dispone en un registro,  $R$ , de 8 bits un carácter ASCII. Suponiendo que sólo se graban en él caracteres numéricos, vocales o caracteres de control, diseñar un sistema combinacional que tenga de entradas las salidas que interese de  $R$  y la salida sean tres funciones,  $n$ ,  $v$  y  $c$ , que se hagan 1, si y sólo si el registro almacena:  $a$ ) un carácter numérico;  $b$ ) una vocal;  $c$ ) un carácter de control, respectivamente.

## SOLUCIÓN

De la tabla de caracteres ASCII (Apéndice 2), se tiene lo siguiente:

- Los caracteres numéricos están comprendidos entre  $30 = 0011\ 0000$  y  $39 = 0011\ 1001$ .
- Las vocales están comprendidas entre  $61 = 0110\ 0001$  y  $7A = 0111\ 1010$ .
- Los caracteres de control están comprendidos entre  $00 = 0000\ 0000$  y  $1F = 0001\ 1111$ .

Debemos identificar las posiciones de los bits que diferencian unos tipos de caracteres de otros. Utilizando la notación que indica la Figura 3.35, se observa que considerando los bits  $R_6$  y  $R_5$  se pueden diferenciar entre grupos. Concretamente se verifica:

- Caracteres numéricos:  $R_6 = 0$  y  $R_5 = 1$ .
- Vocales:  $R_6 = 1$  y  $R_5 = 1$ .
- Caracteres de control:  $R_6 = 0$  y  $R_5 = 0$ .

Registro →	$R_7$	$R_6$	$R_5$	$R_4$	$R_3$	$R_2$	$R_1$	$R_0$
Cifras →	0	0	1	1	x	x	x	x
Vocales →	0	1	1	0	x	x	x	x
	0	1	1	1	x	x	x	x
Control →	0	0	0	0	x	x	x	x

**Figura 3.35.** Identificación de caracteres ASCII.

En consecuencia, con dos inversores y tres circuitos AND podemos implementar el sistema para identificar los tipos de caracteres, tal y como se muestra en la Figura 3.36.

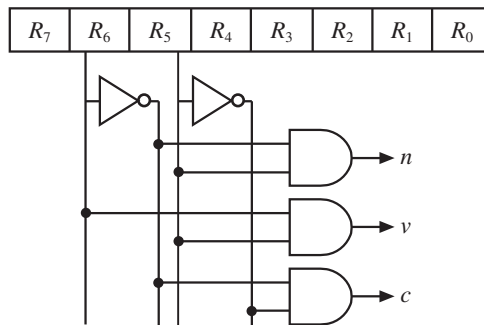
- P3.11.** Diseñar un sistema de detección de paridad par; es decir, cuya salida sea 1 si y sólo si la paridad de los 3 bits de sus entradas es par.

## SOLUCIÓN

Primero construimos la tabla verdad. La función,  $p$ , debe ser uno si hay cero o un número par de unos (como son tres bits, dos unos). La tabla verdad se muestra en la Tabla 3.19.

En consecuencia, la forma canónica de la función es  $p = a'b'c' + a'b'c + a'b'c + abc'$ .

Esta función no se puede simplificar. No obstante, recordemos (Figura 3.3) que una puerta EXNOR proporciona a la salida un 1 si y sólo si el número de unos a su entrada es par. Es decir, el sistema se implementa con una sencilla puerta EXNOR de tres entradas.



**Figura 3.36.** Sistema digital para identificación de distintos tipos de caracteres ASCII.

$a$	$b$	$c$	$p$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

**Tabla 3.19.** Tabla de verdad para un detector de paridad par.

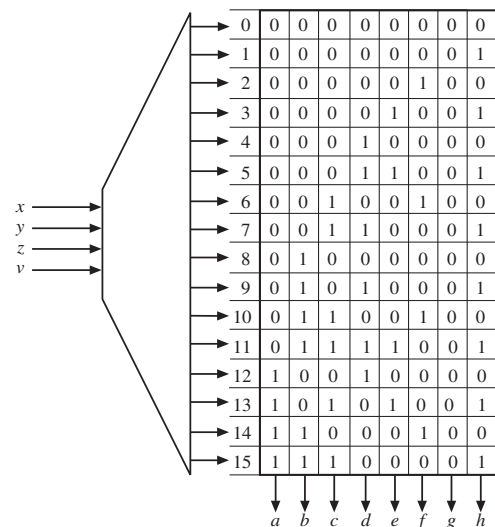
- P3.12.** Diseñar con una ROM una **tabla de consulta**, de forma que al proporcionar en la entrada un número entero sin signo de 4 bits, nos proporcione a la salida su cuadrado. Indicar los contenidos que debe tener la memoria ROM.

### SOLUCIÓN

En la Tabla 3.20 incluimos los números binarios de 4 bits su equivalente en decimal, y sus respectivos cuadrados. El número del que se desea obtener su cuadrado debe presentarse a la entrada de la memoria (como dirección), y la ROM proporcionará a su salida el contenido de esa dirección. Entonces basta con que almacenemos el cuadrado de cada número en la posición asociada a este número, tal y como se muestra en la Figura 3.37.

$x$ $y$ $z$ $v$	Valor decimal	Cuadrado	Cuadrado en binario							
			$a$	$b$	$c$	$d$	$e$	$f$	$g$	$h$
0 0 0 0	0	0	0	0	0	0	0	0	0	0
0 0 0 1	1	1	0	0	0	0	0	0	0	1
0 0 1 0	2	4	0	0	0	0	0	1	0	0
0 0 1 1	3	9	0	0	0	0	1	0	0	1
0 1 0 0	4	16	0	0	0	1	0	0	0	0
0 1 0 1	5	25	0	0	0	1	1	0	0	1
0 1 1 0	6	36	0	0	1	0	0	1	0	0
0 1 1 1	7	49	0	0	1	1	0	0	0	1
1 0 0 0	8	64	0	1	0	0	0	0	0	0
1 0 0 1	9	81	0	1	0	1	0	0	0	1
1 0 1 0	10	100	0	1	1	0	0	1	0	0
1 0 1 1	11	121	0	1	1	1	1	0	0	1
1 1 0 0	12	144	1	0	0	1	0	0	0	0
1 1 0 1	13	169	1	0	1	0	1	0	0	1
1 1 1 0	14	196	1	1	0	0	0	1	0	0
1 1 1 1	15	225	1	1	1	0	0	0	0	1

**Tabla 3.20.** Cuadrados de los números de 0 a 15.



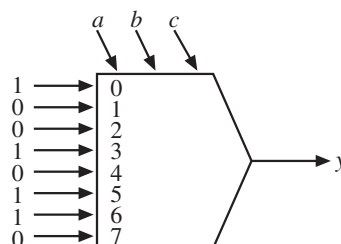
**Figura 3.37.** Tabla de consulta para obtener los cuadrados de enteros de 4 bits.

- P3.13.** Sintetizar la función  $y = a'b'c' + ab'c + a'bc + abc'$  con un multiplexor.

*Pista:* Aplicar las señales de entrada,  $a$ ,  $b$  y  $c$ , como señales de control, y las entradas de datos ponerlas a 0 o a 1, según interese.

### SOLUCIÓN

Recuérdese (Sección 3.1.5.4) que con las señales de selección de un multiplexor se lleva a la salida el valor que haya en la entrada seleccionada. Así, si en la selección hay  $abc = 000$ , en la salida,  $z$ , aparece la información que haya en la entrada 0, que corresponde al término mínimo  $a'b'c'$ , luego esa entrada la debemos poner a 1 (físicamente a la tensión eléctrica que corresponda al 1 lógico). Los otros términos mínimos de la función son el  $ab'c$  que corresponde a la entrada 5, el  $a'bc$  que corresponde a la entrada 3 y el  $abc'$  que corresponde a la entrada 6. En consecuencia, todas las entradas las ponemos a 0, salvo la 0, 3, 5 y 6. El circuito es el que se muestra en la Figura 3.38.



**Figura 3.38.** Síntesis de una función de conmutación con un multiplexor.

En resumen, con el circuito de la figura, cuando  $abc = 000$ ,  $y = 1$ ; cuando  $abc = 001$ ,  $y = 0$ ; cuando  $abc = 010$ ,  $y = 0$ ; cuando  $abc = 011$ ,  $y = 1$ , etc. Es decir, el circuito implementa la función deseada.

**P3.14.** Suponiendo que el circuito de la Figura 3.39 es una PROM, indicar las conexiones que habría que destruir para implementar las funciones:

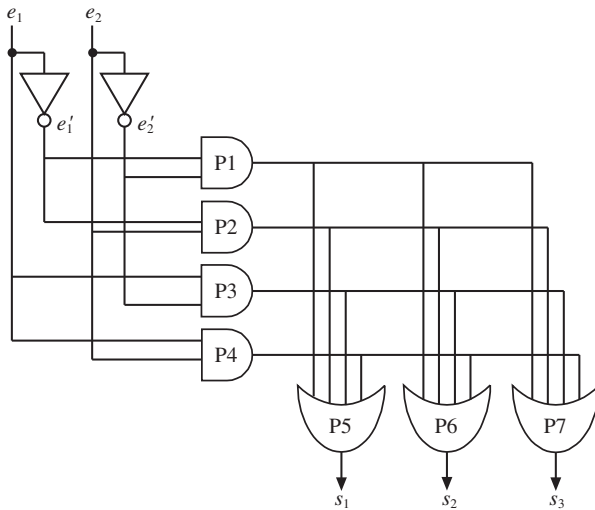
$$s_1 = e_1'e_2' + e_1'e_2 + e_1e_2$$

$$s_2 = e_1'e_2 + e_1e_2'$$

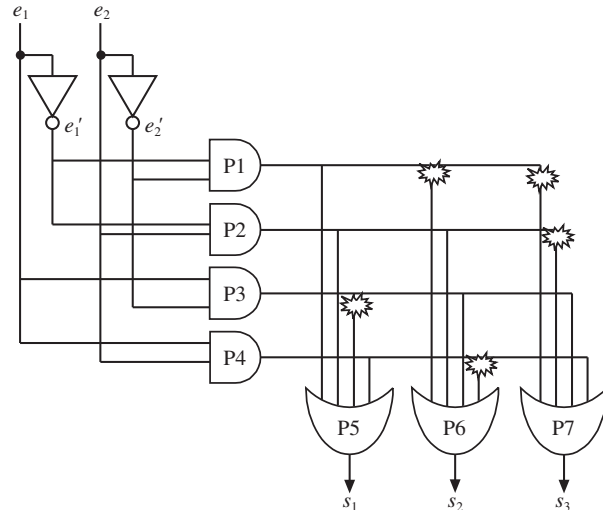
$$s_3 = e_1e_2' + e_1e_2$$

SOLUCIÓN

Recuérdese que en una ROM se programa las conexiones del plano de salida. Sencillamente anulamos en cada función los términos mínimos que no intervienen. El resultado se muestra en la Figura 3.40, donde hemos marcado las conexiones destruidas.



**Figura 3.39.** Circuito programable (ROM) sintetizador de tres funciones de dos variables.



**Figura 3.40.** ROM programada para sintetizar las tres funciones del problema P3.14.

**P3.15.** Suponiendo que el circuito de la Figura 3.41 es una PLA, indicar las conexiones que habría que realizar para implementar las funciones:

$$s_1 = e_1'e_2' + e_1'e_2 + e_1e_2$$

$$s_2 = e_1'e_2 + e_1e_2'$$

$$s_3 = e_1e_2' + e_1e_2$$

SOLUCIÓN

Recuérdese (Sección 3.1.5.6) que en una PLA se programan las conexiones del plano de entrada. Sencillamente anulamos en cada función los términos mínimos que no intervienen. El resultado se muestra en la Figura 3.42, donde hemos marcado las conexiones creadas.

Podemos, en primer lugar, simplificar las funciones:

$$s_1 = e_1'e_2' + e_1'e_2 + e_1e_2 = e_1'e_2' + e_1'e_2 + e_1'e_2 + e_1e_2 = (e_1'e_2' + e_1'e_2) + (e_1'e_2 + e_1e_2) = e_1' + e_2$$

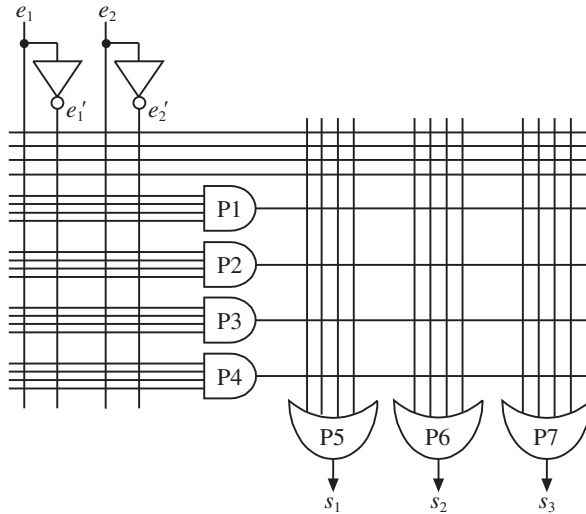
$$s_2 = e_1'e_2 + e_1e_2'$$

$$s_3 = e_1e_2' + e_1e_2 = e_1$$

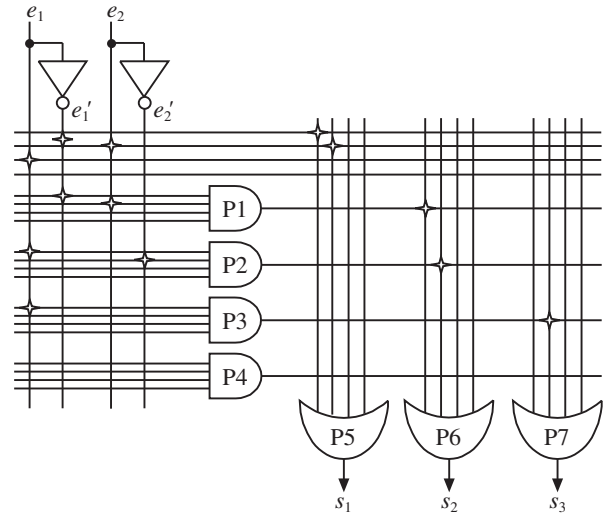
En el plano de entrada debemos obtener los siguientes términos:

$$e_1', e_2, e_1'e_2, e_1e_2', e_1e_2, e_1e_2$$

y en plano de salida, conectamos a las entradas de las puertas OR los términos correspondientes a cada función. El resultado se muestra en la Figura 3.42, donde hemos marcado con estrellas las conexiones a realizar.



**Figura 3.41.** PLA programable sintetizador de tres funciones de dos variables.



**Figura 3.42.** PAL programado para sintetizar las tres funciones del Problema P3.15.

### SISTEMAS SECUENCIALES

**P3.16.** Se denomina **tabla de excitación** de un biestable a una tabla en que se muestran los valores que hay que proporcionar en sus entradas para obtener todas las transiciones de estados posibles ( $0 \rightarrow 0$ ;  $0 \rightarrow 1$ ;  $1 \rightarrow 0$ , y  $1 \rightarrow 1$ ). Escribir la tabla de excitación de un biestable *RS*.

#### SOLUCIÓN

Recordemos que la tabla verdad de un biestable *RS* es la que se indica en Tabla 3.21. La tabla de excitación determina qué valores hay que dar a *S* y *R* para obtener un cambio de estado determinado. Entonces, como primeras columnas de la tabla de excitación debemos poner todas las combinaciones de cambios de estado posibles. Para cada cambio de estado posible, por inspección de la tabla verdad original podemos determinar los valores de *S* y *R* que provocan el cambio. En concreto, los casos posibles son:

- Cambio de estado de  $0 \rightarrow 0$ . La señal *S* debe ser 0 y *R* es indiferente (las indiferencias las denotamos con asterisco \*).
- Cambio de estado de  $0 \rightarrow 1$ . La señal *S* debe ser 1 y *R* debe ser 0.
- Cambio de estado de  $1 \rightarrow 0$ . La señal *S* debe ser 0 y *R* debe ser 1.
- Cambio de estado de  $1 \rightarrow 1$ . La señal *S* puede ser 0 o 1 (indiferente) y *R* debe ser 0.

Esta descripción es la que se resume en la tabla de excitación (Tabla 3.22).

$Q_n$	$S$	$R$	$Q_{n+1}$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	*
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	*

**Tabla 3.21.** Tabla verdad de un biestable *RS*.

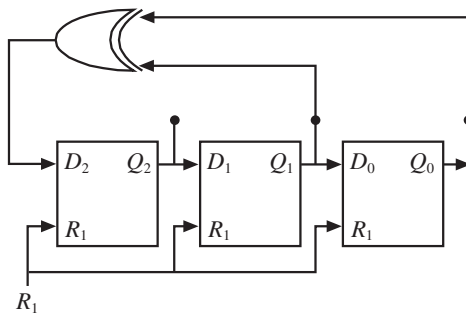
$Q_n$	$Q_{n+1}$	$S$	$R$
0	0	0	–
0	1	1	0
1	0	0	1
1	1	–	0

**Tabla 3.22.** Tabla de excitación de un *RS*.

**P3.17.** Dado el registro de desplazamiento de la Figura 3.43, y suponiendo que inicialmente todos los biestables están al valor 1, obtener los diferentes estados por los que pasa este sistema secuencial (es decir, obtener las diferentes configuraciones por las que va pasando el registro de desplazamiento).

#### SOLUCIÓN

Observamos que, al llegar cada pulso de reloj, la información del biestable uno se almacena en el cero, la del dos en la del uno y la información de la salida de la puerta EXOR en el biestable de la etapa dos. Recuérdese que la función EXOR genera un 1 solo y sólo si las entradas son distintas. En definitiva, el comportamiento es el que se muestra en la tabla siguiente:



	$Q_2$	$Q_1$	$Q_0$	$D2 = Q_1 \oplus Q_0$
Estado inicial →	1	1	1	0
Primer pulso →	0	1	1	0
Segundo pulso →	0	0	1	1
Tercer pulso →	1	0	0	0
Cuarto pulso →	0	1	0	1
Quinto pulso →	1	0	0	1

*Y se repite la secuencia 100, 010, 100, 010, ...*

**Figura 3.43.** Registro de desplazamiento con realimentación.

## Problemas propuestos



### PRINCIPIOS DEL ÁLGEBRA DE CONMUTACIÓN

**P3.18.** Se dispone de un registro  $R$  de 8 bits. En otro registro,  $MAS$ , se puede ubicar un patrón de bits determinado por el usuario. Indicar qué patrón de bits (**máscara**) hay que ubicar en  $MAS$  y la operación lógica a realizar entre  $R$  y  $MAS$  para poner a uno los bits de orden 0, 2 y 4 de  $R$ .

**P3.19.** Se dispone de un registro  $R$  de 8 bits. En otro registro,  $MAS$ , se puede ubicar un patrón de bits determinado por el usuario. Indicar qué patrón de bits (**máscara**) hay que ubicar en  $MAS$  y la operación lógica a realizar entre  $R$  y  $MAS$  para complementar los 4 bits menos significativos de  $R$ .

### MINIMIZACIÓN DE FUNCIONES

**P3.20.** Minimizar algebraicamente la siguiente función:

$$x = a'b'c + a'bc' + a'bc + abc' + ab'c'$$

### FORMAS NORMALIZADAS

**P3.21.** Obtener la forma canónica de la expresión:

$$u = (x + y') \cdot (x + z') + xy$$

**P3.22.** Sean las dos siguientes funciones de conmutación:

$$z = (b + c' + d') \cdot (a + b + d) \cdot (a' + b + d)$$

$$y = a'bd' + c'd + bd + cad'$$

- Determinar si son iguales.
- Obtener sus formas canónicas.

### IMPLEMENTACIÓN DE FUNCIONES

**P3.23.** Diseñar un sistema digital que tenga cuatro entradas y cuya salida sea 1 cuando en la combinación de entrada el número de ceros sea menor que el de unos.

**P3.24.** Implementar el circuito del problema anterior (Problema P3.23) con tan sólo puertas NAND.

**P3.25.** Diseñar un sistema digital cuya entrada es un código del mes del año, y disponga de tres salidas:  $x$ ,  $y$ ,  $f$ . La salida  $x$  debe ser 1 si y sólo si el mes de la entrada tiene 31 días, la salida  $y$  debe ser uno si y sólo si el mes de entrada tiene 30 días, y la salida  $f$  debe ser 1, si y sólo si el mes es febrero. *Nota:* Establecer un código con el mínimo número de bits posibles para los meses del año.

**P3.26.** Diseñar un sistema digital en el que su entrada representa un número natural de 4 bits, y su salida debe ser 1 si el número que representa la entrada es primo.



**P3.27.** Diseñar un circuito de 4 entradas ( $a, b, c, d$ ) tal que su salida ( $z$ ) sea 1 si el número decimal correspondiente a la entrada es impar.

**P3.28.** Diseñar un sistema combinacional que genere un bit de paridad impar asociado a los bits correspondientes a una cifra hexadecimal que se encuentre en la entrada.

**P3.29.** Diseñar un circuito que reste dos números enteros sin signo de dos bits. El circuito debe disponer de tres salidas, las dos de la resta y una para el adeudo, caso de que el sustraendo sea mayor que el minuendo.

**P3.30.** Diseñar un circuito cuya entrada son dos números,  $A$  y  $B$ , enteros sin signo representados con  $n = 2$  bits, y una salida. La salida debe ser 1 si  $B > A$ .

## BLOQUES COMBINACIONALES

**P3.31.** Obtener la tabla verdad de un decodificador de  $n = 4$  entradas.

**P3.32.** Diseñar un codificador binario con 2 entradas.

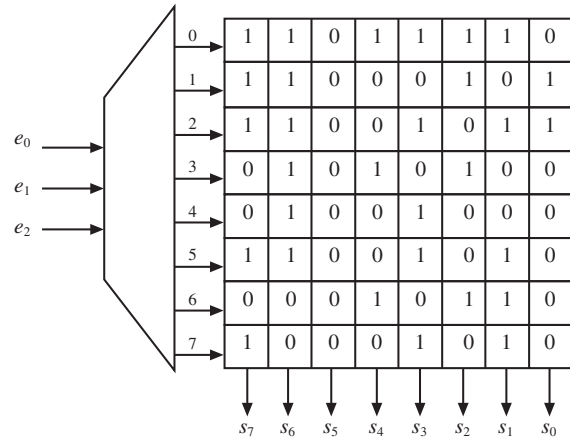
**P3.33.** En la ROM de la Figura 3.39 indicar las conexiones que habría que destruir para implementar las funciones:

$$\begin{aligned} s_1 &= e_1'e_2 + e_1e_2' \\ s_2 &= e_1'e_2' + e_1e_2 \\ s_3 &= e_1'e_2' + e_1e_2' + e_1e_2 \end{aligned}$$

**P3.34.** En la PLA de la Figura 3.41, qué conexiones habría que establecer para implementar las funciones:

$$\begin{aligned} s_1 &= e_1'e_2 + e_1e_2' \\ s_2 &= e_1'e_2' + e_1e_2 \\ s_3 &= e_1'e_2' + e_1e_2' \end{aligned}$$

**P3.35.** Dada la memoria ROM de la Figura 3.44, indicar la función que se obtiene en la salida  $s_7$ .



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

Figura 3.44. ROM del Problema P3.35.

## SISTEMAS SECUENCIALES

**P3.36.** Se denomina **tabla de excitación** de un biestable a una tabla en que se muestran los valores que hay que proporcionar en sus entradas para obtener todas las transiciones de estados posibles ( $0 \rightarrow 0$ ;  $0 \rightarrow 1$ ;  $1 \rightarrow 0$ , y  $1 \rightarrow 1$ ). Escribir la tabla de excitación de un biestable  $JK$ .

**P3.37.** Se denomina **tabla de excitación** de un biestable a una tabla en que se muestran los valores que hay que proporcionar en sus entradas para obtener todas las transiciones de estados posibles ( $0 \rightarrow 0$ ;  $0 \rightarrow 1$ ;  $1 \rightarrow 0$ , y  $1 \rightarrow 1$ ). Escribir la tabla de excitación de un biestable  $D$ .

**P3.38.** Se denomina **tabla de excitación** de un biestable a una tabla en que se muestran los valores que hay que proporcionar en sus entradas para obtener todas las transiciones de estados posibles ( $0 \rightarrow 0$ ;  $0 \rightarrow 1$ ;  $1 \rightarrow 0$ , y  $1 \rightarrow 1$ ). Escribir la tabla de excitación de un biestable  $T$ .

**P3.39.** Describir el comportamiento del circuito que se muestra en la Figura 3.45, cada vez que llega un pulso de reloj y suponiendo que inicialmente las salidas son  $u = v = x = y = 0$ . *Nota:* Las entradas en cada biestable  $JK$  actúan con el flanco de bajada del reloj.

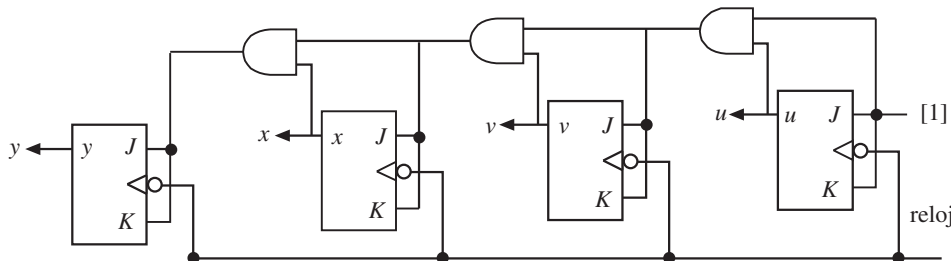


Figura 3.45. Contador binario.

# El procesador

Según indicamos en el Capítulo 1 el procesador es la unidad que determina el funcionamiento de un computador. El presente capítulo se dedica expresamente a dicha unidad. En primer lugar (Sección 4.1) se describen los elementos internos de un procesador. Posteriormente (Sección 4.2) se analiza la forma en que la unidad de control descompone una instrucción máquina en microoperaciones, que se ejecutan en sucesivos ciclos de reloj. Para entender mejor el funcionamiento de un computador presentamos en la Sección 4.3 el lenguaje máquina de un procesador. Las tres secciones citadas anteriormente se circunscriben dentro de un computador didáctico elemental (que denominamos CODE-2) que dispone de los elementos básicos de cualquier computador tipo von Neumann, pero su sencillez permite entender más fácilmente la estructura y funcionamiento de los computadores en general. Hoy día la mayor parte de los procesadores se construyen bajo la forma de microprocesadores o microcontroladores, por lo que a ellos les dedicamos la Sección 4.4. Existen dos tendencias importantes en cuanto al diseño de procesadores, dependiendo de que el repertorio de instrucciones sea complejo o no. Estas tendencias, denominadas CISC y RISC, respectivamente, han marcado notablemente las arquitecturas actuales, y a ellas se referirá la Sección 4.5.

## 4.1. Elementos internos de un procesador

La Figura 4.1 muestra algunos detalles internos del procesador (CODE-2) al que nos vamos a referir en las tres primeras secciones de este capítulo. La totalidad de los conceptos que se describirán son de aplicación a la mayoría de los computadores actuales. La interconexión entre los elementos no se explicita completamente ya que depende del computador concreto, y se analizará más adelante (Capítulo 7).

El procesador dispone de un conjunto de registros, denominados **registros de uso general** (*r0* a *rD*, en el ejemplo de la Figura 4.1), que suelen estar integrados en un **banco de registros** (*RF*, *Register File*) (Sección 3.2.2.3). Estos registros se utilizan como almacén temporal de los datos con los que va a operar la ALU o de resultados intermedios. También pueden dedicarse a almacenar direcciones de memoria. La longitud de estos registros suele ser de una palabra, aunque es corriente que existan registros de distinta longitud; así, en un computador de 32 bits, es habitual que se puedan utilizar registros de media palabra (16 bits), de palabra (32 bits) y de doble palabra (64 bits). A veces los lenguajes máquina incluyen instrucciones para operar entre un registro significado, denominado **acumulador**, y otro registro o una posición de memoria, depositándose el resultado en el acumulador. En

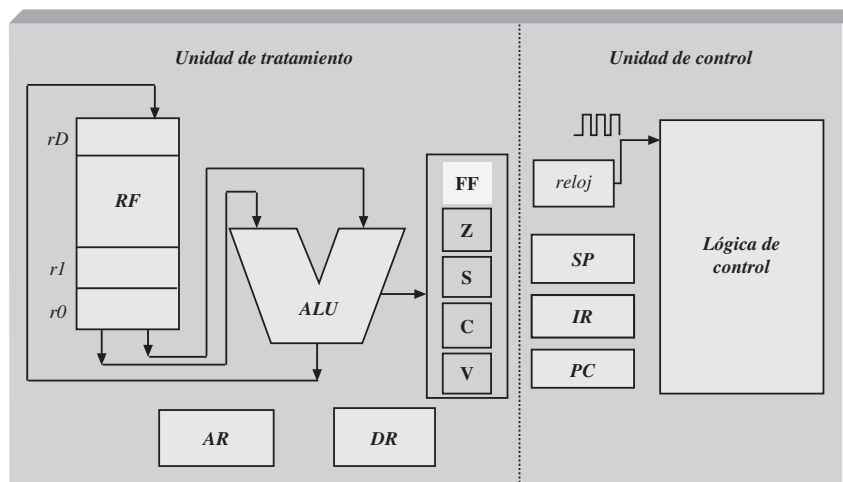


Figura 4.1. Elementos básicos de un procesador.

el ejemplo de la Figura 4.1 no se ha incluido un acumulador, realizándose todas las operaciones con datos del banco de registros. En definitiva, del esquema de la figura se deduce que la *ALU* opera con el contenido de dos de los registros de *RF* guardando el resultado también en uno de sus registros.

Asociado a la *ALU* hay unos **biestables indicadores** o **biestables de condición** (en inglés, *flag flip-flops*; *FF*, en la figura), que se ponen a 1 o 0 dependiendo de la última operación realizada en la *ALU*. Así, es normal que existan biestables tales como:

- **C:** Acarreo.
- **S:** Indicador de signo (si el último resultado de la *ALU* es negativo, por ejemplo  $S \leftarrow 1$ ).
- **Z:** Indicador de cero (si el último resultado de la *ALU* es cero,  $Z \leftarrow 1$ ).
- **P:** Indicador de paridad, se pone a 1 si la paridad del resultado es par.
- **V:** Indicador de desbordamiento, etc.

La memoria principal y los periféricos se conectan con el procesador, Figura 4.2, por medio de dos buses: uno de direcciones y otro de datos; además de con las dos señales de control *R/W'*, para seleccionar entre lectura o escritura, e *IO/M'*, para elegir entre Entrada-Salida o Memoria. Para un buen fun-

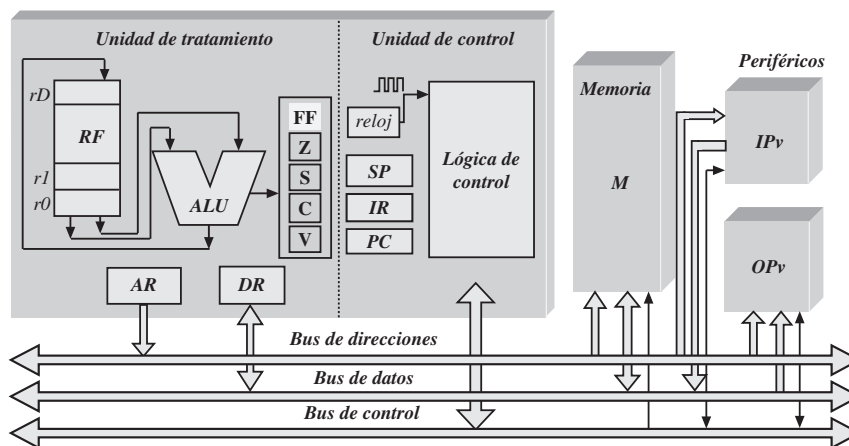


Figura 4.2. Interconexión unibus del procesador con la memoria y los puertos de E/S.

cionamiento del conjunto, el procesador dispone de un **registro dirección** (*AR*, *Address Register*), donde deberá ubicarse la dirección del dato/instrucción a leer o escribir, y un **registro de datos** (*DR*, *Data Register*), donde se almacenará el dato a escribir en la memoria o puertos de salida, o la información leída de la memoria o de un puerto de entrada, dependiendo del caso. En definitiva, el procesador para:

- *Escribir en memoria o en un puerto de salida*, debe ubicar en *AR* la dirección donde se debe escribir el dato, en *DR* el dato a escribir en memoria o en el puerto, y generar las siguientes señales de control:  $R/W' = 0$ , e  $IO/W' = 0$  si la escritura es en memoria, o  $IO/W' = 1$  si es en un puerto.
- *Leer en memoria o de un puerto de entrada*, debe ubicar en *AR* la dirección donde se encuentra el dato o instrucción a leer y generar las siguientes señales de control:  $R/W' = 1$  e  $IO/W' = 0$ , si la lectura es de memoria, o  $IO/W' = 1$  si es de un puerto de entrada. El dato leído se cargará en *DR*.

Como se indicó en el Capítulo 1 el procesador está formado por la unidad de control y la unidad de tratamiento (Figura 4.1). La unidad de control contiene la **lógica de control**, que está constituida por los circuitos que generan las distintas señales de control, y el **reloj**, que es un generador de pulsos, con los que se sincronizan todas las microoperaciones que implican la ejecución de las distintas instrucciones máquina. También en la unidad de control se encuentra el **registro de instrucción** (*IR*, *Instruction Register*), que está dedicado a memorizar temporalmente la instrucción del programa que la unidad de control está interpretando o ejecutando. Recuérdese (Sección 1.2.1) que el programa a ejecutar reside en la memoria principal, y la unidad de control va *captando o buscando las instrucciones* secuencialmente de la memoria principal, para poder interpretarlas y generar las órdenes de ejecución. La captación de instrucción implica leer la instrucción de memoria y almacenarla en el registro de instrucción. Pero para captar la instrucción, el procesador debe, de alguna forma, saber el punto del programa que está en ejecución, o, en otras palabras, contabilizar la dirección de memoria donde se encuentra la instrucción en ejecución. Este es el objetivo del **contador de programa**<sup>1</sup> (*PC*, *Program Counter*), que es un registro-contador que actúa como contador ascendente binario, con posibilidad de carga paralela (Sección 3.2.2), y que *contiene en todo momento la dirección de memoria donde se encuentra la instrucción siguiente a ejecutar*.

El último elemento que queda por describir es el **puntero de pila** (*SP*, *Stack Pointer*). El puntero de pila físicamente es, como el contador de programa, un contador binario ascendente con la opción de carga en paralelo. Su cometido está relacionado con una estructura pila (LIFO) que se mantiene en la memoria principal relacionada con las llamadas a (y retornos de) subrutinas, y que se analizarán en detalle en la Sección 4.2.2.

## 4.2. Esquema de funcionamiento de un procesador

Para iniciar la ejecución de un programa se ubica en el *PC* la dirección de memoria donde comienza dicho programa. El contenido de *PC* se transvasa a *AR* (esto, abreviadamente, se simboliza así:  $AR \leftarrow PC$ ), y la unidad de control da a la memoria la orden de leer ( $R/W' = 1$ ;  $IO/W' = 0$ ). Después de un tiempo determinado (**tiempo de acceso a memoria**), en el bus de entrada/salida de memoria aparecerá la información contenida en la posición de dirección *PC*; es decir, la instrucción del programa (en este caso la primera), cargándose en el registro *DR* ( $DR \leftarrow M(AR)$ ). Posteriormente la información contenida en *DR*, esto es, la instrucción, se carga en el registro *IR* ( $IR \leftarrow DR$ ). También el contenido del contador de programa es incrementado ( $PC \leftarrow PC + 1$ ), apuntando así a la siguiente instrucción (por simplificar, en esta descripción se supone que cada instrucción ocupa una sola posición de memoria).

<sup>1</sup> También se suele conocer con el nombre de **puntero de instrucciones** (*IP*, *Instruction Pointer*).

Una vez cargada la instrucción en *IR* es decodificada y ejecutada bajo la monitorización de la unidad de control.

#### 4.2.1. GESTIÓN DE LAS INSTRUCCIONES DE TRANSFERENCIAS DE DATOS Y ARITMÉTICO-LÓGICAS

Si, por ejemplo, la instrucción máquina en ejecución es *sumar el contenido de dos registros y dejar el resultado en otro*, la unidad de control genera las señales de control necesarias para llevar los contenidos de los registros donde se encuentran los sumandos a los dos buses de entrada de la ALU, realizar la suma (Figura 3.20) y llevar el resultado de la ALU al registro indicado. Es conveniente señalar que en la propia instrucción de suma se especifican los registros donde se encuentran los sumandos y al que hay que llevar el resultado.

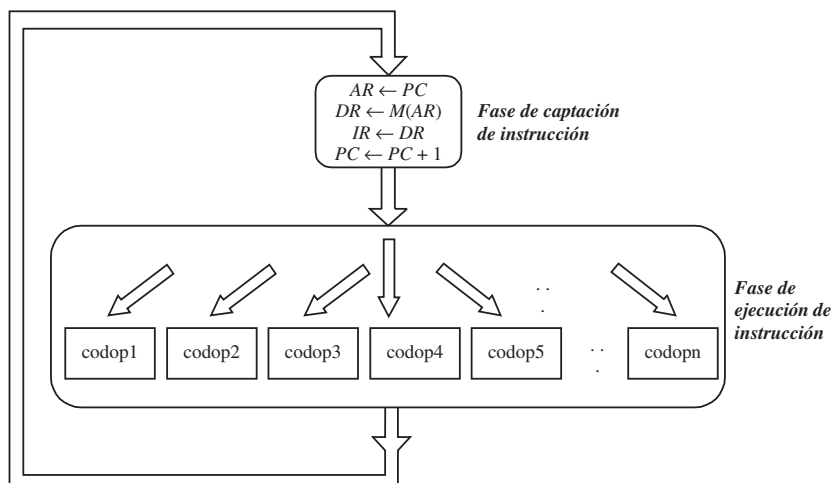
Una vez ejecutada la instrucción en curso, la unidad de control vuelve a repetir el ciclo anterior; es decir, capta una nueva instrucción (la lleva a *IR*) y después la decodifica y ejecuta. Este ciclo se repite iterativamente hasta que concluye la ejecución del programa.

Resumiendo lo dicho en los párrafos anteriores, el contador de programa siempre contiene la dirección de memoria de la siguiente instrucción que se va a ejecutar. Por tanto, una vez leída una instrucción el valor de *PC* debe incrementarse en 1, para *apuntar* a la siguiente instrucción. Todas las instrucciones comienzan siempre con una **fase de captación de instrucción**. Después tiene lugar una **fase de ejecución**, que es específica del código de operación (**codop**) de cada instrucción, y en ella se activan los distintos elementos del computador necesarios para la ejecución de la instrucción propiamente dicha. En definitiva la unidad de control genera las señales de control necesarias para realizar las siguientes operaciones (Figura 4.3):

- **Fase de captación de instrucción:**

$$\begin{aligned} AR &\leftarrow PC \\ DR &\leftarrow M(AR) \\ IR &\leftarrow DR \\ PC &\leftarrow PC + 1 \end{aligned}$$

(Suponemos que cada instrucción ocupa una sola posición de memoria.)



**Figura 4.3.** Diagrama que muestra el ciclo y microoperaciones generadas por la unidad de control para ejecutar una instrucción máquina.

- **Fase de ejecución.** Se realizan las operaciones específicas correspondientes al **codop** de la instrucción almacenada en *IR*. Dependiendo de la instrucción se efectúan operaciones tales como captación o búsqueda de operandos en memoria, cálculos en la ALU, almacenamiento de resultados en registros o memoria, etc.

De la descripción anterior se deduce que la ejecución de una instrucción lleva consigo por lo menos un acceso a memoria (para captar la instrucción), pudiendo efectuar accesos adicionales (captación de datos, memorización de resultados, etc.).

El tipo de operaciones indicadas anteriormente (carga de un registro:  $AR \leftarrow PC$ ,  $IR \leftarrow DR$ ; lectura de memoria:  $DR \leftarrow M(AR)$ , incremento del contador de programa:  $PC \leftarrow PC + 1$ , etc.) son las acciones más elementales que puede hacer el computador, y reciben el nombre de **microoperaciones**. Se puede concluir que una *instrucción máquina* implica la realización de un conjunto determinado de *microoperaciones* en un orden preestablecido.

#### EJEMPLO 4.1

Supóngase que se dispone de un computador organizado en palabras de 16 bits. En la posición *H'0039* de memoria (Figura 4.4) se encuentra la instrucción *H'0700*, que simbólicamente se representa como *LD r7,(rD)*, y carga el contenido del registro *r7* del banco de memoria con el dato que se encuentra en la posición de memoria que especifica *rD*; es decir:

$$r7 \leftarrow M(rD)$$

Suponiendo que en *rD* se encuentra el valor *H'54C2*, y los contenidos de memoria de la Figura 4.4, indicar las microoperaciones que se realizarán durante la ejecución de la instrucción, y los cambios que van teniendo lugar en *PC*, *IR*, *AR*, *DR* y *r7*.

Direcciones	Contenidos	
0000	7AC4	
0007	65C9	
0039	0700	
003A	607D	← instrucciones
003B	2D07	
003C	C000	
54C2	D7A2	← dato
FFFF	3FC4	

**Figura 4.4.** Contenidos de memoria para los Ejemplos 4.1, 4.2 y 4.3.

#### SOLUCIÓN

Teniendo en cuenta el objetivo de la instrucción y las Figuras 4.3 y 4.4, las microoperaciones que efectuará el computador son las que se muestran en la Tabla 4.1. Se observa que para ejecutar la instrucción máquina del ejemplo se necesitan 7 microoperaciones que genera la unidad de control. La instrucción realiza dos accesos a memoria: uno para captar la instrucción y otro para captar el dato a almacenar en *R7*.

Fase	Microoperación	Contenidos de registros					← Valores iniciales
		<i>PC</i>	<i>IR</i>	<i>AR</i>	<i>DR</i>	<i>r7</i>	
		0039	—	—	—	—	
Captación de instrucción	$AR \leftarrow PC$	0039	—	<u>0039</u>	—	—	
	$DR \leftarrow M(AR)$	0039	—	0039	<u>0700</u>	—	
	$IR \leftarrow DR$	0039	<u>0700</u>	0039	0700	—	
	$PC \leftarrow PC + 1$	<u>003A</u>	0700	0039	0700	—	
Ejecución de instrucción	$AR \leftarrow rD$	003A	0700	<u>54C2</u>	0700	—	
	$DR \leftarrow M(AR)$	003A	0700	54C2	<u>D7A2</u>	—	
	$r7 \leftarrow DR$	003A	0700	54C2	D7A2	<u>D7A2</u>	

**Tabla 4.1.** Microoperaciones y contenidos de los registros durante la ejecución de la instrucción  $LD\ r7,(rD)$ .

### EJEMPLO 4.2

Supóngase ahora que en la posición  $H'003A$  de memoria (Figura 4.4) se encuentra la instrucción  $H'607D$ , que simbólicamente se representa como  $ADDS\ r0,r7,rD$ , que suma (en notación complemento a 2) los contenidos de  $r7$  y  $rD$  y almacena el resultado en  $r0$ ; es decir:

$$r0 \leftarrow r7 + rD$$

Indicar las microoperaciones que se realizarán durante la ejecución de la instrucción, y los cambios que van teniendo lugar en  $PC$ ,  $IR$ ,  $AR$ ,  $DR$  y  $r0$ .

### SOLUCIÓN

Teniendo en cuenta el objetivo de la instrucción, las Figuras 4.3 y 4.4, las microoperaciones que efectuará el computador son las que se muestran en la Tabla 4.2. Se observa que para ejecutar la instrucción máquina del ejemplo se necesitan 6 microoperaciones que genera la unidad de control, y sólo se hace un acceso a memoria para captar la instrucción.

Fase	Microoperación	Contenidos de registros					← Valores iniciales
		<i>PC</i>	<i>IR</i>	<i>AR</i>	<i>DR</i>	<i>r0</i>	
		003A	0700	54C2	D7A2	—	
Captación de instrucción	$AR \leftarrow PC$	003A	0700	<u>003A</u>	D7A2	—	
	$DR \leftarrow M(AR)$	003A	0700	003A	<u>607D</u>	—	
	$IR \leftarrow DR$	003A	<u>607D</u>	003A	607D	—	
	$PC \leftarrow PC + 1$	<u>003B</u>	607D	003A	607D	—	
Ejecución de instrucción	$r0 \leftarrow r7 + rD$	003B	607D	003A	607D	<u>2C6F*</u>	

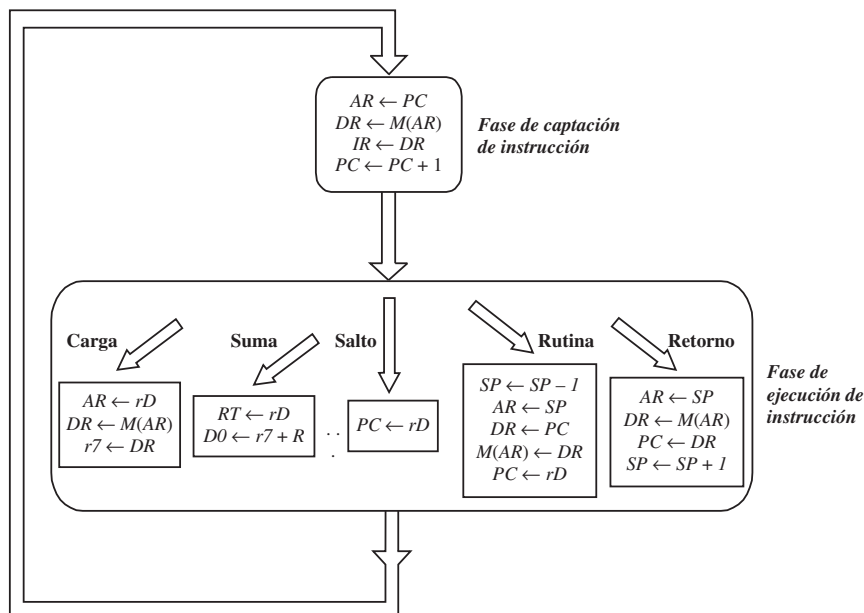
\* Ya que  $D7A2 + 54CD = 2C6F$ .

**Tabla 4.2.** Microoperaciones y contenidos de los registros durante la ejecución de la instrucción  $ADDS\ r0,r7,rD$ .

Al final de la ejecución de la instrucción los biestables indicadores pasan a tener los siguientes valores:

$$C = 1; S = 0; Z = 0; V = 0$$

En la Figura 4.5 se indican esquemáticamente, dentro del ciclo de ejecución de la unidad de control, las microoperaciones a realizar con las instrucciones de carga de registro y de suma. Las micro-



**Figura 4.5.** Diagrama mostrando las microoperaciones que tienen lugar cuando se ejecutan las instrucciones de carga de registro, suma, salto, llamada a subrutina y retorno de subrutina.

operaciones correspondientes a otras instrucciones máquina de tipo aritmético o lógico son muy similares.

### 4.2.2. GESTIÓN DE LAS INSTRUCCIONES DE CONTROL

Según lo visto anteriormente, el PC, durante la fase de captación de instrucción, se incrementa en 1, apuntando a la siguiente instrucción a ejecutar. Se plantea el problema de la ejecución de las instrucciones que alteran el orden secuencial de ejecución (**instrucciones de control de flujo**), ya que con el esquema anterior el programa se va ejecutando secuencialmente, según el orden en que están las instrucciones almacenadas en la memoria principal, y las instrucciones de bifurcación precisamente pueden alterar ese orden de ejecución.

Conviene distinguir entre dos tipos de instrucciones de control de flujo:

1. Bifurcaciones (o saltos condicionales) y saltos (incondicionales).
2. Llamadas a subrutinas y retornos de subrutinas, condicionales o no.

A continuación se describen estos dos tipos de instrucciones.

#### Instrucciones de control de flujo que no hacen referencia a subrutinas

Con este tipo de instrucciones se puede alterar el orden de ejecución de un programa, saltando a una instrucción ubicada en una dirección de memoria arbitraria,  $ds$ , establecida por el programador. Una vez ejecutada la instrucción de salto el programa ejecuta la instrucción contenida en la posición  $ds$  y las que se encuentren sucesivamente a partir de ella (hasta el fin del programa o hasta otra alteración del flujo de control provocada por otra instrucción de este tipo). La dirección de salto, dependiendo del juego de instrucciones del procesador, puede darse de distintas formas; por ejemplo puede estar en la propia instrucción (direccionamiento directo), o puede estar contenida en un registro del procesador (direccionamiento indirecto). En el caso de CODE-2, la dirección de salto debe estar en el registro  $rD$ .



Supóngase por ejemplo que en la posición de memoria *1A2E* se tuviese la instrucción de salto *BR*, y que en *rD* estuviese almacenado el valor *32B7*. Esta instrucción significa bifurcar (*branch*) a la dirección *32B7*. Cuando finaliza la ejecución de la instrucción citada, en vez de ejecutarse la siguiente instrucción que esté en memoria (en la posición *1A2F*), se ejecutaría la que estuviese en la posición *32B7*, y siguientes.

La implementación de este tipo de instrucciones por el procesador es muy sencilla (Figura 4.5): en su fase de ejecución el procesador cambia el contenido del contador de programa por el contenido de *rD* —en el ejemplo *32B7*— (recuérdese que *PC* físicamente es un contador con posibilidad de carga en paralelo):

$$PC \leftarrow rD$$

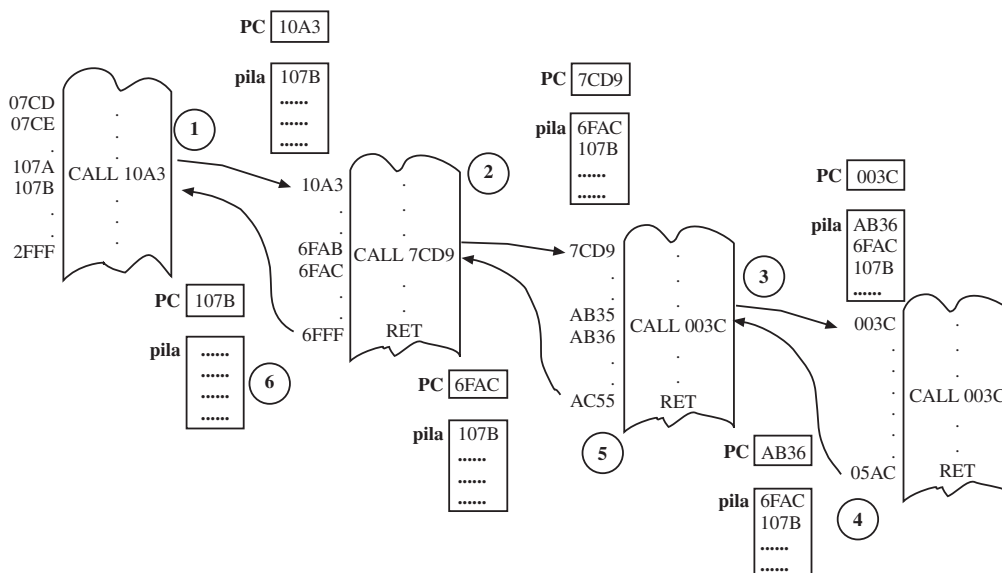
De esta forma, en la siguiente captación de instrucción, al leer de memoria la instrucción cuya dirección está en *PC* (como siempre, Figura 4.5), lo que se capta es la instrucción cuya dirección estaba en el registro *rD*, en otras palabras, se produce automáticamente el salto.

En las **instrucciones de salto condicional**, el salto se produce sólo si se cumple alguna condición, establecida por el valor de alguno de los biestables indicadores. Por ejemplo, la instrucción *BZ* hace que se salte a ejecutar la instrucción que está a partir de *32B7* (y sucesivas) si el biestable *Z* está a 1.

### Instrucciones de control de flujo que hacen referencia a procedimientos

Una llamada a una subrutina es una ruptura de la secuencia normal de ejecución de las instrucciones de un programa, de forma que tras la ejecución de la instrucción de llamada se ejecuta otro programa, denominado **subrutina**, **rutina** o **procedimiento**. Una vez ejecutado el procedimiento, se retorna (por medio de una **instrucción de retorno**) al programa desde el que se hizo la llamada, que continúa ejecutándose a partir de la posición desde la que se saltó a la subrutina. A su vez un procedimiento puede incluir llamadas a otros procedimientos, y así sucesivamente (Figura 4.6).

Desde el punto de vista de la arquitectura del procesador, la diferencia entre una bifurcación o salto y una llamada a un procedimiento es que cuando acaba de ejecutarse el procedimiento llamado hay que *retornar* al programa (o procedimiento) que lo llamó, concretamente se debe retornar a la ins-



**Figura 4.6.** Ejemplo de cómo se gestionan las llamadas y retornos de los procedimientos. **CALL ds** es la instrucción de llamada y **RET** la de retorno.

trucción inmediatamente después de la de llamada, que es precisamente la que se encuentra en la dirección contenida en el PC al ejecutarse la instrucción de llamada. Por tanto, habrá que memorizar temporalmente *los contenidos del PC de las instrucciones de llamada*. Cuando se retorna a un procedimiento de llamada, la dirección de vuelta puede eliminarse de la memoria. El almacenamiento de las direcciones de las instrucciones de llamada se realiza en una memoria o estructura de datos de tipo **pila (memoria LIFO)** (Sección 10.2.5). En la Figura 4.6 puede verse un ejemplo de tres llamadas sucesivas a procedimientos. Puede observarse que cuando se llama a una subrutina el valor del contador de programa en ese momento se almacena en la cabecera de la pila, y la dirección de llamada se almacena en el PC. En las instrucciones de retorno, sencillamente el valor de la cabecera de la pila en ese momento se almacena en el contador de programa, desapareciendo aquél de la pila. Obsérvese que la recuperación de las direcciones de retorno ha de hacerse en orden inverso al de su almacenamiento, y este es precisamente el motivo para utilizar una estructura de tipo LIFO para almacenarlas.

De acuerdo con lo anterior, las microoperaciones a realizar durante la *fase de ejecución* de las instrucciones relacionadas con las llamadas a procedimientos serán:

*Llamada a procedimiento*

$Pila \leftarrow PC$

$PC \leftarrow rD$

Donde, en este caso, suponemos que en *rD* está almacenada la dirección de comienzo de la subrutina.

*Retorno de procedimiento*

$PC \leftarrow Pila$

Nótese que para producir el retorno lo único que debe hacer la unidad de control es transferir la dirección almacenada en la cabecera de la pila (que es la dirección de retorno) al PC. La siguiente instrucción que se capte, por lo tanto, será ya del programa que llamó al procedimiento.

En los computadores de muy alta velocidad la memoria pila suele diseñarse con circuitos específicos (**pila hardware**), y forma parte, por tanto, del procesador; pero, en la mayor parte de los computadores, la pila se gestiona o *simula* en la memoria principal (**pila software**). En efecto, *las direcciones de retorno* (los contenidos de PC en las instrucciones de llamada) se almacenan en una región de la memoria principal que se denomina **zona LIFO**. El programador de lenguaje máquina (o el sistema operativo o el traductor) determinan la zona de memoria que se reservará como memoria LIFO para almacenar las direcciones de retorno.

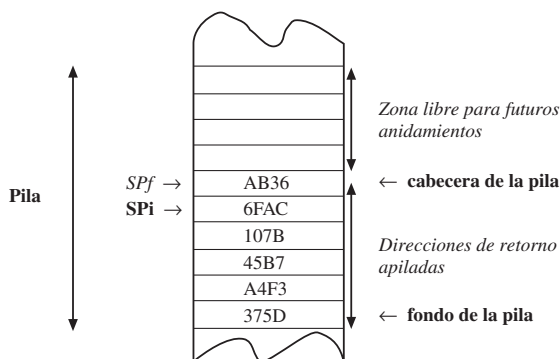
El procesador contiene en su interior un registro específico, donde la unidad de control almacena la *dirección* de la zona LIFO donde se guardó la última dirección de retorno. Este registro se denomina **puntero pila** o **SP (Stack Pointer)** (Figura 4.1). Cuando se ejecuta una instrucción de retorno, la unidad de control ordena leer la última dirección almacenada en la pila; es decir, *el contenido de la dirección de memoria cuya posición está en SP*. Este contenido se cargará en el contador de programa (PC), concluyendo con ello la ejecución de la instrucción de retorno.

Antes de efectuar alguna llamada a procedimiento debe cargarse en SP un valor inicial (la dirección límite de la zona LIFO). Existen instrucciones máquina específicas para efectuar esta carga. El procesador automáticamente se encarga de almacenar y recuperar las direcciones de retorno, según se vayan ejecutando instrucciones máquina de llamada y retorno de procedimientos.

En el ejemplo de procesador que estamos analizando en este capítulo las direcciones de salto se encuentran en el registro *rD*, y el registro *rE* (no incluido en la Figura 4.1) actúa como puntero de pila. Para implementar la llamada a una subrutina, primero hay que decrementar SP para que apunte a la siguiente posición libre de la memoria, que pasará a ser la nueva cabecera de la pila (Figura 4.7), después hay que salvar en la posición SP de memoria el valor del contador de programa (para lo cual hay que llevar SP a AR y PC a DR), y después debe llevarse la dirección de salto (que está en *rD*) a PC. En definitiva, las microoperaciones a realizar en una instrucción de llamada a subrutina son (Figura 4.5):

$$\begin{aligned}
 SP &\leftarrow SP - 1 \\
 AR &\leftarrow SP \\
 DR &\leftarrow PC \\
 M(AR) &\leftarrow DR \\
 PC &\leftarrow rD
 \end{aligned}$$

Siempre que se llama a una subrutina se realizan las microoperaciones que se acaban de describir, de forma que la zona LIFO se va rellenando, en la Figura 4.7, hacia arriba (desde direcciones altas hacia direcciones bajas), y en  $SP$  se conserva la posición donde se encuentra la dirección de retorno de la rutina en ejecución.



**Figura 4.7.** Esquema que muestra cómo se gestiona la pila en la memoria principal.  $SP_i$  y  $SP_f$  representan, respectivamente, las direcciones contenidas en  $SP$  antes y después de hacer la llamada a la rutina.

En las ejecuciones de retornos de procedimientos el puntero se va incrementando (la zona LIFO se va vaciando; en la Figura 4.7 hacia abajo, hacia las direcciones altas). En el procesador que estamos considerando, la unidad de control genera las señales de control que efectúan las siguientes microoperaciones (Figura 4.5):

$$\begin{aligned}
 AR &\leftarrow SP \\
 DR &\leftarrow M(AR) \\
 PC &\leftarrow DR \\
 SP &\leftarrow SP + 1
 \end{aligned}$$

recuperándose así en el  $PC$  la dirección de retorno del procedimiento que estaba ejecutándose. Una nueva llamada a un procedimiento destruiría (sin causar ningún problema, por no necesitarse ya) la última dirección de retorno recuperada.

Si se previese que como máximo se produjesen 64 llamadas sucesivas entre subrutinas (es decir, nivel de anidamiento 64), se tendrían que reservar 64 posiciones de memoria como zona LIFO, y antes de llamar a cualquier procedimiento habría que cargar  $SP$  con el valor de la frontera de direcciones más alta de dicha zona.

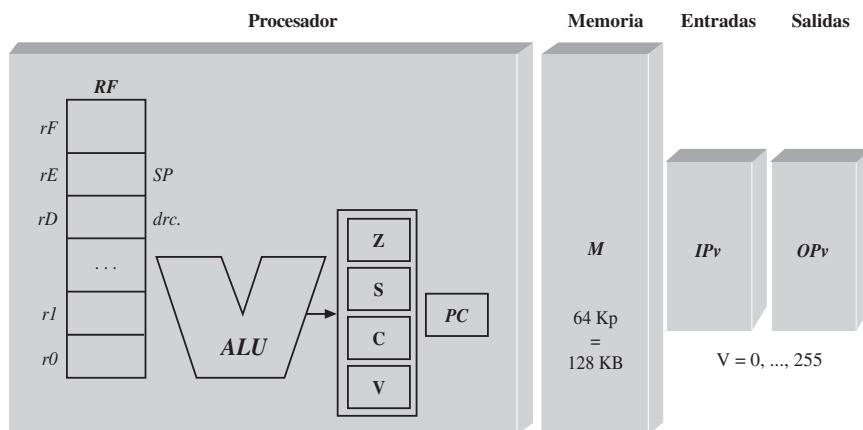
### 4.3. Programación en código máquina (CODE-2)

En esta sección se describe la programación en lenguaje máquina de la versión 2 de un **Computador Didáctico Elemental (CODE-2)**. CODE-2 es de 16 bits de longitud de palabra y tiene tan sólo 16 instrucciones máquina. En esta sección sólo se va a describir su funcionamiento al *nivel de máquina convencional* (lenguaje máquina, Sección 1.5). En primer lugar (Sección 4.3.1) se describen los elementos de CODE-2 a los que se tiene acceso desde el lenguaje máquina, posteriormente se estudian los formatos (Sección 4.3.2) y el repertorio de instrucciones (Sección 4.3.3). Por último (Sección 4.3.4) se muestran algunos trucos de programación y un ejemplo de programa (Sección 4.3.5).

### 4.3.1. ELEMENTOS A LOS QUE SE TIENE ACCESO DESDE EL LENGUAJE MÁQUINA

En la Figura 4.8 se muestra un esquema simplificado de CODE-2, que incluye los elementos accesibles directamente desde el lenguaje máquina; éstos son:

- *Banco de registros*, compuesto de 16 registros  $r0$ , ...,  $rF$ . El registro  $rE$  se utiliza como **puntero de pila** ( $SP$ ) y el  $rD$  como **registro de dirección**, para almacenar posiciones de memoria con objeto de realizar direccionamientos indirectos e indexados. A pesar de la dedicación específica de los registros  $rE$  y  $rD$ , el programador puede utilizarlos también para otros cometidos.
- *Unidad aritmético lógica (ALU)*, con la que se pueden realizar las operaciones aritméticas de suma y resta (entero en complemento a 2), la operación lógica NAND, desplazamientos lógicos del acumulador a derecha e izquierda y desplazamiento aritmético a la derecha.
- *Biestables indicadores (FF)*. Se activan después de realizar en la ALU las instrucciones de suma, resta, NAND y de desplazamientos; todo ello de acuerdo con el resultado de la operación correspondiente. Se incluyen biestables de cero ( $Z$ ), signo ( $S$ ), acarreo ( $C$ ) y desbordamiento ( $V$ ).
- *Memoria principal (M)*, de  $2^{16} = 64$  K palabras de 16 bits (128 KBytes).
- *Puertos de entrada (IP, input port)*. El lenguaje máquina admite la inclusión de hasta 256 puertos de entrada ( $IP00$  a  $IPFF$ ) de periféricos.
- *Puertos de salida (OP, output port)*. El lenguaje máquina admite la inclusión de hasta 256 puertos de salida ( $OP00$  a  $OPFF$ ) de periféricos.

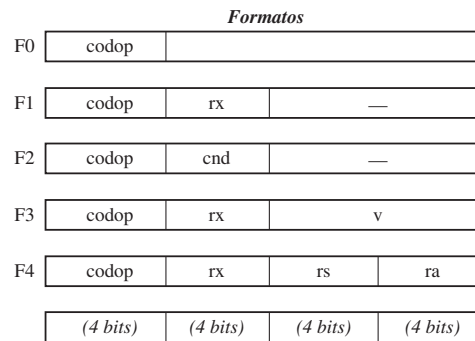


**Figura 4.8.** Esquema simplificado del Computador Didáctico Elemental (CODE-2).

### 4.3.2. FORMATOS DE INSTRUCCIONES Y DATOS

El repertorio de instrucciones de CODE-2 considera cinco tipos de formato ( $F0$  a  $F4$ ), utilizando cada instrucción uno de ellos (Figura 4.9). Todas las instrucciones (y formatos, por tanto) comienzan con un campo de 4 bits (*codop*), que corresponde al código de operación. Por tanto CODE-2 puede tener como máximo 16 instrucciones. Los otros campos de la instrucción dependen del formato que se utilice:

- El *formato F0* corresponde a instrucciones que sólo tienen código de operación.
- El *formato F1* dispone del campo de código de operación y un campo de 4 bits para especificar un registro,  $rx$  ( $x = 0, 1, 2, \dots, E, F$ ), pudiéndose así referenciar los registros del  $r0$  a  $rF$ .



**Figura 4.9.** Formatos de las instrucciones de CODE-2.

- El *formato F2*, además del campo *codop*, contiene un campo denominado *cnd* que sirve para especificar una condición en las instrucciones de salto o de llamadas a subrutinas, de acuerdo con el significado de la Tabla 4.3.

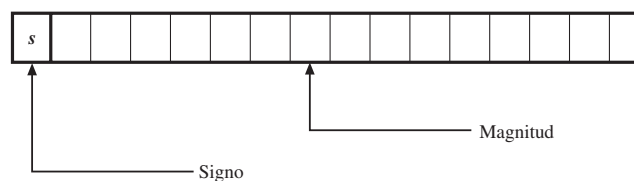
Tipo de salto o llamada	—	cnd	
Incondicional	<i>R</i>	0	0000
Condional, si biestable de cero es 1	<i>Z</i>	1	0001
Condional, si biestable de signo negativo es 1	<i>S</i>	2	0010
Condional, si biestable de acarreo es 1	<i>C</i>	3	0011
Condional, si biestable de desbordamiento es 1	<i>V</i>	4	0100

**Tabla 4.3.** Tipos de saltos y llamadas a subrutinas, y códigos utilizados para especificar cada uno de ellos.

- El *formato F3*, además del campo *codop*, contiene otros dos campos: el *rx* (de 4 bits) que sirve para especificar uno de los registros y el *v* utilizado para dar un valor inmediato (dato o dirección) de 8 bits.
- El *formato F4* contiene 4 campos, cada uno de ellos de 4 bits: *codop*, *rd* (registro del destino del resultado), *rs* (registro de la fuente del dato) y *ra* (registro auxiliar). Los tres últimos campos sirven para especificar cualquier registro y pueden por tanto tomar un valor de 0 a *F*.

En la Figura 4.10 se incluye un esquema del formato de los datos. Sencillamente corresponde a números enteros en notación de complemento a 2. Como son datos de 16 bits, los números mayor y menor que se puede representar (al nivel de lenguaje máquina) son, respectivamente (Sección 2.4.1):

$$N_{mayor} = 2^{16-1} - 1 = 32.767; N_{menor} = -2^{16-1} = -32.768$$



**Figura 4.10.** Formatos de los datos de CODE-2.

### 4.3.3. REPERTORIO DE INSTRUCCIONES MÁQUINA

En la Tabla 4.4 se dan las 16 instrucciones de las que consta el repertorio de instrucciones máquina de CODE-2. Al escribir  $d \leftarrow s$  se indica que el contenido del elemento *s* (fuente) se copia el elemen-

to  $d$  (destino). El contenido de la fuente en ningún caso cambia; es decir, se *copia* el valor de la fuente en el destino.

Los elementos destino, dependiendo de la instrucción concreta, pueden ser:

- Registros,  $r0$  a  $rF$ .
- Una posición de memoria; así,  $M(x)$  especifica el contenido de la posición de memoria  $x$ .
- El contador de programa,  $PC$ .
- Un puerto de salida; así,  $OPv$  especifica el contenido del puerto  $v$  de salida.

Por otra parte los elementos fuente pueden ser:

- Registros,  $r0$  a  $rF$ .
- Una posición de memoria, como en el caso de fuente.
- Un puerto de entrada; así,  $IPv$  especifica el contenido del puerto  $v$  de entrada.

Los bits particulares de un registro se especifican dándolos entre paréntesis, y se numeran de derecha a izquierda, empezando por 0. Así,  $rx(0)$  significa el bit menos significativo del registro  $rx$ ;  $rx(15)$  representa el bit más significativo del registro  $rx$ , y  $rx(7:0)$  representan los 8 bits (byte) menos significativo de  $rx$ .

A cada instrucción se le ha asignado un nemónico (columna 4 de la Tabla 4.4) que resulta muy cómodo de utilizar cuando se está redactando el programa, como paso previo a la utilización de los códigos binarios que son los que forman el programa en lenguaje máquina (los nemónicos se utilizan también en el lenguaje ensamblador, que no se incluye en el presente libro).

Codop		Nombre	Nemónico	Parámetros	Formato	Explicación	N.º ciclos
Binario	Hex						
0000	0	Cargar	LD	$rx, [v]$	F3	$rx \leftarrow M(rD + v)$	9
0001	1	Almacenar	ST	$[v], rx$	F3	$M(rD + v) \leftarrow rx$	9
0010	2	Carga inmediata baja	LLI	$rx, v$	F3	$rx(15:8) \leftarrow H'00; rx(7:0) \leftarrow v$	6
0011	3	Carga inmediata alta	LHI	$rx, v$	F3	$rx(15:8) \leftarrow v$	8
0100	4	Entrada	IN	$rx, IPv$	F3	$rx \leftarrow IPv$	8
0101	5	Salida	OUT	$OPv, rx$	F3	$OPv \leftarrow rx$	8
0110	6	Suma	ADDS	$rx, rs, ra$	F4	$rx \leftarrow rs + ra$	7
0111	7	Resta	SUBS	$rx, rs, ra$	F4	$rx \leftarrow rs - ra$	7
1000	8	NAND	NAND	$rx, rs, ra$	F4	$rx \leftarrow (rs \cdot ra)'$	7
1001	9	Desplaza izquierda	SHL	$rx$	F1	$C \leftarrow rx(15), rx(i) \leftarrow rx(i-1), i = 15, \dots, 1; rx(0) \leftarrow 0$	6
1010	A	Desplaza derecha	SHR	$rx$	F1	$C \leftarrow rx(0), rx(i) \leftarrow rx(i+1), i = 0, \dots, 14; rx(15) \leftarrow 0$	6
1011	B	Desplaza arit. dcha.	SHRA	$rx$	F1	$C \leftarrow rx(0), rx(i) \leftarrow rx(i+1), i = 0, \dots, 14$	6
1100	C	Salto	B-	cnd	F2	Si cnd se cumple, $PC \leftarrow rD$	6
1101	D	Subrutina	CALL-	cnd	F2	Si cnd se cumple, $rE \leftarrow rE - 1, M(rE) \leftarrow PC, PC \leftarrow rD$	6/10
1110	E	Retorno	RET	—	F0	$PC \leftarrow M(rE); rE \leftarrow rE + 1$	8
1111	F	Parar	HALT	—	F0	Parar	6

**Tabla 4.4.** Repertorio de instrucciones de CODE-2.

Por otra parte cada instrucción, además del nemónico del código de operación, puede tener ningún, uno, dos o tres parámetros, dependiendo de que su formato sea de tipo  $F0$ ,  $F1$ ,  $F2$ ,  $F3$  o  $F4$ , respectivamente, tal como se indica en la sexta columna de la Tabla 4.4. Por otra parte, en la última columna de la Tabla 4.4 se incluye el número de ciclos de reloj que consume cada instrucción. Consultando esta columna, y conociendo la frecuencia de reloj, se puede obtener el tiempo que tarda

en ejecutarse cada instrucción, o la duración de un programa concreto sumando lo que tarda cada una de las instrucciones que se ejecutan. Llamando  $ni$  al número de ciclos de la instrucción  $i$ , y  $F$  a la frecuencia de reloj de CODE-2, dicha instrucción tarda en ejecutarse:

$$ti = ni \cdot T = \frac{ni}{F}$$

donde  $T$  es el período del reloj (*tiempo de ciclo*).

A continuación vamos a describir cada una de las instrucciones, siguiendo el orden de la Tabla 4.4.

#### **LD (cargar un registro con un dato de la memoria)**

*Descripción de la función a realizar:*  $rx \leftarrow M(rD + v)$

Esta instrucción carga en un registro el contenido de la posición de memoria cuya dirección se obtiene sumando el contenido del registro  $rD$  con el valor del campo  $v$  de la propia instrucción.

#### **EJEMPLO 4.3**

Supóngase que en el registro  $rD$  se encuentra el valor  $H'003B$  y se ejecuta la instrucción  $LD\ rD, [A7]$ .

El código máquina de esta instrucción es:  $H'01A7 = B'0000\ 0001\ 1010\ 0111$ .

Como  $v = A7$ , se verifica que  $rD + v = 003B + A7 = 00E2$ ; con lo que la instrucción llevaría el contenido de la posición  $00E2$  de memoria al registro  $rD$ ; es decir:

$$rD \leftarrow M(00E2)$$

#### **ST (almacenar el contenido de un registro en la memoria)**

*Descripción de la función a realizar:*  $M(rD + v) \leftarrow rx$

Esta instrucción carga el contenido del registro  $rx$ , especificado en la instrucción, en la posición de memoria cuya dirección se obtiene sumando el contenido del registro  $rD$  con el valor del campo  $v$  de la propia instrucción.

#### **EJEMPLO 4.4**

Supóngase la instrucción  $H'1000$  y que en el registro  $r0$  se encuentra el valor  $H'BC79$  y en  $rD$  el valor  $003B$ . ¿Qué realiza esta instrucción?

#### **SOLUCIÓN**

Como el código de operación es  $0001$  se trata de una instrucción de almacenar el contenido de un registro en memoria. El valor del campo  $v$  en la instrucción es  $00$ ; con lo que la dirección de memoria será:  $rD + v = 003B$ . En consecuencia, lo que realiza la instrucción es:

$$M(003B) \leftarrow r0$$

Es decir, se carga en la dirección  $003B$  el valor  $BC79$ .

#### **LLI (carga inmediata baja)**

*Descripción de la función a realizar:*  $rx(15:8) \leftarrow H'00$ ;  $rx(7:0) \leftarrow v$

Esta instrucción carga en el byte menos significativo del registro  $rx$ , especificado en la instrucción, el valor del campo  $v$  de la propia instrucción. El byte menos significativo se pone a  $H'00$ .

**EJEMPLO 4.5**

La instrucción *LLI r5, 01* carga en el registro *r5* el valor *0001*; es decir:

$$r5 \leftarrow H'0001$$

**LHI (carga inmediata alta)**

*Descripción de la función a realizar:*  $rx(15:8) \leftarrow v$

Esta instrucción carga en el byte más significativo del registro *rx*, especificado en la instrucción, el valor del campo *v* de la propia instrucción. El byte menos significativo conserva su valor previo.

**EJEMPLO 4.6**

Se puede introducir el valor *H'FABC* en el registro *rD* con las siguientes instrucciones:

*LLI rD,BC*, o en código máquina 2DBC  
*LHI rD,FA*, o en código máquina 3DFA

**IN (entrada)**

*Descripción de la función a realizar:*  $rx \leftarrow IPv$

Esta instrucción capta el contenido del puerto *IPv* (donde *v* se especifica en la propia instrucción) y lo lleva al registro *rx*, dado también en la propia instrucción.

**EJEMPLO 4.7**

La instrucción *IN r5, 01* carga en el registro *r5* el valor que hubiese en el puerto de entrada *IP01*; es decir:

$$r5 \leftarrow IP01$$

**OUT (salida)**

*Descripción de la función a realizar:*  $OPv \leftarrow rx$

Esta instrucción lleva el contenido del registro *rx* al puerto de salida *OPv* (donde *v* se especifica en la propia instrucción).

**EJEMPLO 4.8**

Para llevar el contenido del registro *rF* al puerto de salida *OPA7*, se debe utilizar la siguiente instrucción:

*OUT OPA7, rF*, o en código máquina 5FA7

**ADDS (suma)**

*Descripción de la función a realizar:*  $rx \leftarrow rs + ra$

Esta instrucción suma el contenido de los registros *rs* y *ra*, depositando el resultado en el registro *rx*. La suma se hace para números enteros representados en complemento a dos. Los tres registros son especificados en tres campos de la propia instrucción. Los cuatro biestables indicadores (*Z*, *S*, *C* y *V*) se activan de acuerdo con el resultado de la operación.

**EJEMPLO 4.9**

Para sumar los contenidos de *r5* y *r6* y ubicar el resultado en *r7* se puede utilizar la siguiente instrucción:

*ADDS r7,r5,r6*, o en código máquina 6756



**SUBS (resta)**

*Descripción de la función a realizar:*  $rx \leftarrow rs - ra$

Esta instrucción resta el contenido de los registros  $rs$  y  $ra$ , depositando el resultado en el registro  $rx$ . La resta se hace para números enteros representados en complemento a dos. Los tres registros son especificados en tres campos de la propia instrucción. Los 4 biestables indicadores ( $Z$ ,  $S$ ,  $C$  y  $V$ ) se activan de acuerdo con el resultado de la operación.

**EJEMPLO 4.10**

Para restar  $r5$  consigo mismo y ubicar el resultado en  $r7$  se puede utilizar la siguiente instrucción:

*SUBS r7,r5,r5*, o en código máquina 7755

El resultado de esta operación es cero, con lo que los biestables indicadores tomarán los siguientes valores:  $Z = 1$ ,  $S = 0$ ,  $C = 0$ ,  $V = 0$ .

**NAND (operación lógica NAND)**

*Descripción de la función a realizar:*  $rx \leftarrow (rs \cdot ra)'$

Esta instrucción efectúa la operación lógica NAND bit a bit (es decir, entre bits del mismo orden) de los datos contenido en los registros  $rs$  y  $ra$ , depositando el resultado en el registro  $rx$ . Los tres registros son especificados en tres campos de la propia instrucción. Se activan los biestables indicadores  $Z$  y  $S$  de acuerdo con el resultado de la operación; mientras que  $C$  y  $V$  permanecen inalterados. Recuerdese que la operación lógica NAND se define como se indicó en la Tabla 3.3

**EJEMPLO 4.11**

Suponga los siguientes contenidos en los registros  $r9$  y  $rA$ :  $r9 \leftarrow H'5B7A$ ;  $rA \leftarrow H'70B4$ . Determinar el resultado del registro  $rD$  después de ejecutarse el código máquina 8D9A.

**SOLUCIÓN**

Recuérdese (Tabla 3.3) que con la operación NAND el resultado es cero si y sólo si los dos bits de operandos son uno; con lo que el resultado de hacer la operación NAND entre  $r9$  y  $rA$  será:

$r9 \rightarrow$	5B7A	NAND	0101 1011 0111 1010	
$rA \rightarrow$	70B4		0111 0000 1011 0100	
$rD \rightarrow$			1010 1111 1100 1111	$\rightarrow AF CF$

Es decir, el nuevo valor de  $rD$  será: *AF CF*, y los biestables indicadores:  $Z = 0$  y  $S = 1$ ;  $C$  y  $V$  conservarán sus valores previos.

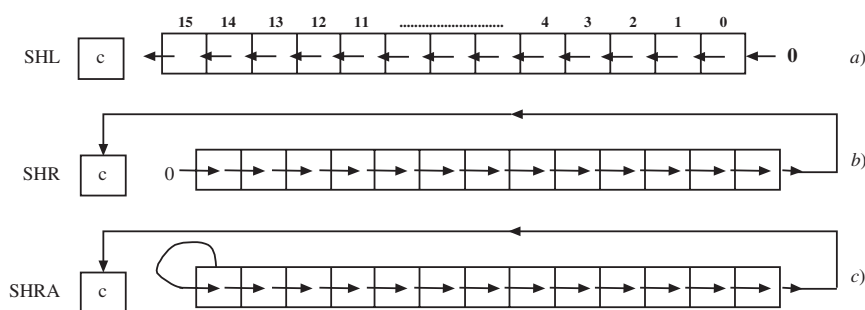
**SHL (desplazamiento a izquierda)**

*Descripción de la función a realizar:*  $C \leftarrow rx(15)$ ,  $rx(i) \leftarrow rx(i - 1)$ ,  $i = 15, \dots, 1$ ;  $rx(0) \leftarrow 0$

Esta instrucción, Figura 4.11a, desplaza los bits del registro  $rx$ , especificado en la propia instrucción, una posición a la izquierda. El bit más significativo pasa al biestable de acarreo ( $C$ ) y el bit menos significativo se hace 0. El biestable de signo ( $S$ ) pasa a tomar el valor del nuevo bit más significativo, y el de cero ( $Z$ ) pasa a valer 1, si el nuevo valor del registro es 0000<sub>H</sub>.

**EJEMPLO 4.12**

Suponga que en  $rF$  se encuentra el valor *E03B*, desplazar sus bits un lugar a la izquierda.



**Figura 4.11.** Desplazamiento de los bits de un registro provocado por las instrucciones: a) SHL, b) SHR y c) SHRA los datos de CODE-2.

### SOLUCIÓN

Para desplazar a la izquierda los bits de  $rF$  utilizamos la siguiente instrucción:

*SHL rF*, o en código máquina: *9F00*

El funcionamiento de esta instrucción es como sigue:

		C	
		↓	
$rF$ inicial →	E03B	–	1110 0000 0011 1011
$rF$ final →		1	1100 0000 0111 0110 → C076

Es decir, el nuevo valor de  $rF$  será: *C076*, y los biestables indicadores:  $Z = 0$ ,  $S = 1$  y  $C = 1$ ;  $V$  conservará su valor previo.

### SHR (desplazamiento a derecha)

Descripción de la función a realizar:  $C \leftarrow rx(0)$ ,  $rx(i) \leftarrow rx(i + 1)$ ,  $i = 0, \dots, 14$ ;  $rx(15) \leftarrow 0$

Esta instrucción, Figura 4.11b, desplaza los bits del registro  $rx$ , especificado en la propia instrucción, una posición a la derecha. El bit menos significativo pasa al biestable de acarreo ( $C$ ) y el bit más significativo se hace 0. El biestable de signo ( $S$ ) pasa a tomar el valor del nuevo bit más significativo, y el de cero ( $Z$ ) pasa a valer 1, si el nuevo valor del registro es  $H'0000_H$ .

### EJEMPLO 4.13

Suponga que en  $rF$  se encuentra el valor *C076*, desplazar sus bits un lugar a la derecha.

### SOLUCIÓN

Para desplazar a la izquierda los bits de  $rF$  utilizamos la siguiente instrucción:

*SHR rF*, o en código máquina: *AF00*

El funcionamiento de esta instrucción es como sigue:

		C	
		↓	
$rF$ inicial →	C076	–	1100 0000 0111 0110
$rF$ final →		0	0110 0000 0011 1011 → 603B

Es decir, el nuevo valor de  $rF$  será: *603B*, y los biestables indicadores tomarán los siguientes valores:  $Z = 0$ ,  $S = 0$  y  $C = 0$ ; y  $V$  conservará su valor previo.

**SHRA (desplazamiento aritmético a derecha)**

*Descripción de la función a realizar:*  $C \leftarrow rx(0)$ ,  $rx(i) \leftarrow rx(I + 1)$ ,  $i = 0, \dots, 14$

Esta instrucción, Figura 4.11c, desplaza los bits del registro  $rx$ , especificado en la propia instrucción, una posición a la derecha. El bit menos significativo pasa al biestable de acarreo ( $C$ ) y el bit más significativo conserva su valor original. El biestable de signo ( $S$ ) pasa a tomar el valor del bit más significativo, y el de cero ( $Z$ ) pasa a valer 1, si el nuevo valor del registro es H'0000.

**EJEMPLO 4.14**

Suponga que en  $rF$  se encuentra el valor  $C076$ , desplazar aritméticamente sus bits un lugar a la derecha.

**SOLUCIÓN**

Para desplazar a la izquierda los bits de  $rF$  utilizamos la siguiente instrucción:

*SHRA rF*, o en código máquina: *BF00*

El funcionamiento de esta instrucción es como sigue:

		C	
		↓	
$rF$ inicial $\rightarrow$	$C076$	—	1100 0000 0111 0110
$rF$ final $\rightarrow$		0	1110 0000 0011 1011 $\rightarrow E03B$

Es decir, el nuevo valor de  $rF$  será:  $E03B$ , y los biestables indicadores tomarán los siguientes valores:  $Z = 0$ ,  $S = 1$  y  $C = 0$ ; y  $V$  conservará su valor previo.

**B- (salto)**

*Descripción de la función a realizar:* Si  $cnd$  es incondicional o si se cumple la condición, entonces  $PC \leftarrow rD$

<i>Instrucción en nemónico:</i>	<i>BR (salto incondicional)</i>
	<i>BZ (salto si Z = 1)</i>
	<i>BS (salto si S = 1)</i>
	<i>BC (salto si C = 1)</i>
	<i>BV (salto si V = 1)</i>

Esta instrucción provoca un salto a la instrucción cuya dirección se encuentra en el registro  $rD$ . El campo  $cnd$  que incluye el formato de esta instrucción (formato  $F2$ , Figura 4.3) especifica la condición de salto, según los códigos de la Tabla 4.3. Es decir, dependiendo de si el valor del biestable indicador especificado por  $cnd$  es 1 después de la última operación realizada en la ALU (con una instrucción ADDS, SUBS, NAND, SHL, SHR o SHRA), se realiza o no el salto. Si el salto es incondicional (instrucción BR,  $cnd = 0000$ ), se efectuará siempre el salto, independientemente del valor de los biestables indicadores.

**EJEMPLO 4.15**

Indicar las instrucciones para provocar un salto incondicional a la dirección  $A3BC$ .

**SOLUCIÓN**

Primero tendremos que cargar en  $rD$  la dirección de salto, y después utilizar una instrucción de salto incondicional:

Nemónico	Hex
LLI $rD, BC$	2DBC
LHI $rD, A3$	3DA3
BR	C000

**CALL- (llamada a subrutina)**

*Descripción de la función a realizar:* Si *cnd* es incondicional o se cumple la condición, entonces  $rE \leftarrow rE - 1$ ,  $M(rE) \leftarrow PC$ ,  $PC \leftarrow rD$

*Instrucción en nemónico:*      *CALLR (llamada incondicional)*  
    *CALLZ (llamada si Z = 1)*  
    *CALLS (llamada si S = 1)*  
    *CALLC (llamada si C = 1)*  
    *CALLV (llamada si V = 1)*

Esta instrucción provoca una llamada a una subrutina cuya dirección se encuentra en el registro *rD*. El campo *cnd* que incluye el formato de esta instrucción (formato *F2*, Figura 4.3) especifica la condición de llamada, según los códigos de la Tabla 4.3. Es decir, dependiendo de si el valor del biestable indicador especificado por *cnd* es 1 después de la última operación realizada en la ALU (con una instrucción ADDS, SUBS, NAND, SHL, SHR o SHRA), se realiza o no la llamada a la subrutina. Si la llamada es incondicional (instrucción CALLR, *cnd* = 0000), se efectuará siempre la llamada, independientemente del valor de los biestables indicadores.

**EJEMPLO 4.16**

Supóngase que se desea realizar una llamada a la subrutina que comienza en la dirección 7AB5 caso de que el resultado de la última operación realizada en la ALU hubiese sido negativo. ¿Qué instrucciones habría que utilizar?

**SOLUCIÓN**

Primero tendremos que cargar en *rD* la dirección de salto, y después utilizar una instrucción de llamada a subrutina si *S* = 1:

Nemónico	Hex
LLI <i>rD,BC</i>	2DB5
LHI <i>rD,A3</i>	3D7A
CALLS	D200

**RET (retorno de subrutina)**

*Descripción de la función a realizar:*  $PC \leftarrow M(rE)$ ;  $rE \leftarrow rE + 1$

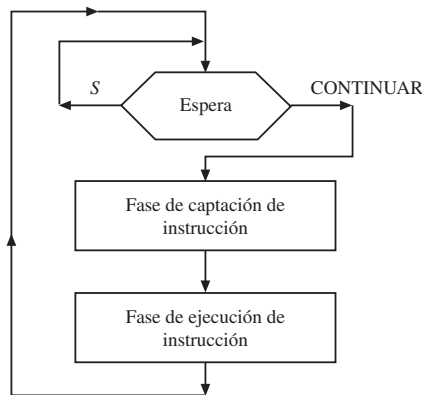
Esta instrucción se incluye al final de una subrutina, y provoca un retorno al último programa que la llamó; concretamente a la instrucción siguiente al CALL-. Recuérdese (Sección 4.2.2) que en la ejecución de una instrucción de retorno, sencillamente debe cambiarse el contenido del contador de programa por el valor de la cabecera de la pila. En el caso de CODE-2 la dirección de la cabecera de la pila está en *rE*, con lo que la operación que realizará la unidad de control es:  $PC \leftarrow M(rE)$ . Además hay que actualizar el valor del puntero de pila; es decir, el procesador hace  $rE \leftarrow rE + 1$ .

**HALT (parada)**

*Descripción de la función a realizar:* entra CODE-2 en estado de espera.

Esta instrucción provoca que CODE-2 entre en un **estado de espera**, deteniéndose su funcionamiento justo antes de la fase de captación de la siguiente instrucción, tal y como se indica en la Figura 4.12. Puede continuarse la ejecución del programa deteniendo pulsando la tecla CONTINUAR del panel de control.

El uso de esta instrucción es de interés, por ejemplo, como final de un programa, o cuando se desea detenerse CODE-2 después de haber dado un resultado por un puerto de salida para que el programador u operador de CODE-2 confirme la visualización de dicho resultado.



**Figura 4.12.** El estado de espera se produce antes del inicio de la captación de la siguiente instrucción.

#### 4.3.4. ALGUNOS TRUCOS DE PROGRAMACIÓN

La realización de un programa que implemente un algoritmo consiste en la descripción de éste utilizando únicamente las instrucciones que ofrece el repertorio de instrucciones del lenguaje de programación a utilizar. Con otras palabras, y refiriéndonos al caso de CODE-2, para realizar cualquier programa *únicamente* podemos utilizar las 16 instrucciones de la Tabla 4.4. Utilizando estas instrucciones con ingenio debemos poder realizar cualquier programa, por complicado que sea.

Para dar una idea de las posibilidades del repertorio de instrucciones de CODE-2, en esta sección incluimos algunas técnicas frecuentemente usadas.

##### Puesta a cero o a uno de un registro

Con mucha frecuencia los programas utilizan los valores 0 y 1 y es aconsejable introducir estos valores en dos registros, por ejemplo el  $r0$  y  $r1$ , respectivamente. Esto puede hacerse con las siguientes instrucciones:

Instrucción máquina			Explicación
Nemónico	Hexadecimal	Binario	
LLI $r0,00$	2000	0010 0000 0000 0000	Cargar $r0$ con H'0000
LLI $r1,01$	2101	0010 0001 0000 0001	Cargar $r1$ con H'0001

##### Copiar el contenido de un registro en otro

Se puede llevar el contenido de un registro a otro utilizando una instrucción de suma. Suponiendo que en  $r0$  hay el valor 0000, el valor de  $r4$  se puede pasar a  $rE$  así:

Instrucción máquina			Explicación
Nemónico	Hexadecimal	Binario	
ADDS $rE,r4,r0$	6E40	0110 1110 0100 0000	$rE \leftarrow r4 + r0$

Los biestables indicadores quedan modificados de acuerdo con el valor de  $r0$ .

##### Detectar si un número es cero o negativo

Se puede detectar si un número, en un registro determinado, es cero activando los biestables indicadores con su valor. Esto puede realizarse sin más que sumarle (o restarle) el valor 0. Así, supóngase

que se desea saltar a la instrucción cuya dirección está en  $rD$  si el valor del registro  $r4$  es cero, y que en  $r0$  está almacenado el valor 0000; el código máquina para lograr ese objetivo sería:

Instrucción máquina			Explicación
Nemónico	Hexadecimal	Binario	
ADDS $r4, r4, r0$	6440	0110 0100 0100 0000	Sumar 0 al número para activar los biestables.
BZ	C100	1100 0001 0000 0000	Saltar si el biestable de cero (Z) se activa.

De igual forma se puede detectar si  $r4$  es negativo; sin más que sustituir la última instrucción (BZ) por la BS.

### Comparar dos números

La comparación de un número consiste en activar los biestables indicadores de acuerdo con la diferencia de sus valores. Así, por ejemplo, si se deseara saltar a la instrucción ubicada en F300 si  $r5 = r4$ , a la ubicada en 547C si  $r5 < r4$  y a la ubicada en 737C si  $r5 > r4$ , se podrían utilizar las siguientes instrucciones:

Instrucción máquina			Explicación
Nemónico	Hex	Binario	
SUBS $rF, r5, r4$	7F54	0111 1111 0101 0100	Comparar $r5$ con $r4$ .
LLI $rD, 00$	2D00	0010 1101 0000 0000	Dirección de salto si son iguales.
LLH $rD, F3$	3DF3	0011 1101 1111 0011	
BZ	C100	1100 0001 0000 0000	Saltar si son iguales ( $Z = 1$ ).
LLI $rD, 7C$	2D7C	0010 1101 0111 1100	Dirección de salto si $r4 > r5$ .
LLH $rD, 54$	3D54	0011 1101 0101 0100	
BS	C200	1100 0010 0000 0000	Saltar si $r4 > r5$ ( $S = 1$ ).
LLH $rD, 73$	3D73	0011 1101 0111 0011	Dirección de salto si $r4 < r5$ .
BR	C000	1100 0000 0000 0000	Salto incondicional.

Obsérvese que:

- Con la instrucción de resta activamos los biestables indicadores, de acuerdo con la diferencia de los dos registros a comparar.
- Las instrucciones LLI y HLH no modifican los valores de los biestables indicadores.
- En estas instrucciones se utilizan los registro  $rF$  y  $rD$ , modificándose sus valores previos.

### No hacer nada (instrucción no operativa)

A veces interesa incluir en un programa instrucciones que no realicen ninguna operación; esto se puede hacer sumando a un registro cualquiera el valor cero:

Instrucción máquina			Explicación
Nemónico	Hexadecimal	Binario	
ADDS $r4, r4, r0$	6440	0110 0100 0100 0000	$r4 \leftarrow r4 + 0$

Ahora bien, esta instrucción no conserva el estado de los biestables indicadores.

Las instrucciones no operativas se utilizan frecuentemente para hacer rutinas de retardo; éstas tienen por objeto consumir un determinado intervalo de tiempo. La instrucción ADDS consume 7 ciclos de reloj (Tabla 4.4); es decir, si la frecuencia de reloj de CODE-2 fuese de 10 MHz, su ejecución tardaría  $7/(10 \cdot 10^6) = 0,7 \mu s$ . Con un bucle de instrucciones puede realizarse una rutina de un retardo concreto (ver Problema 4.18).

### Contadores ascendentes y descendentes

Suponiendo que en  $r1$  hemos memorizado el 0001, es fácil incrementar o decrementar en 1 el valor de cualquier registro sin más que utilizar instrucciones de suma y de resta:

Instrucción máquina			Explicación
Nemónico	Hexadecimal	Binario	
ADDS $r4, r4, r1$	6441	0110 0100 0100 0001	Incremento de $r4$ en 1.
SUBS $r5, r5, r1$	7551	0111 0101 0101 0001	Decremento de $r5$ en 1.

### Detectar si un número es par

Por ejemplo, se quiere saltar a la instrucción en la posición A73B si el número que está almacenado en el registro  $r3$  es par, y a la B73C si es impar. Vamos a utilizar dos métodos que se basan en comprobar si el último bit del número binario es 0, en cuyo caso el número sería par, o es 1, en cuyo caso el número sería impar.

Para comprobar si el último número es 0, con el primer método utilizamos la operación lógica NAND. Es fácil comprobar, con ayuda de la Tabla 3.3, que si hacemos la operación lógica NAND de cualquier número con H'0001 el resultado será FFFF, si el número acaba en 0 (es decir, si es par), y FFFE, si el número acaba en 1 (es decir, si es impar). Con las siguientes instrucciones se implementa el procedimiento anterior, donde en  $r4$  se almacena la máscara FFFF y se ha supuesto que en  $r1$  está almacenado 0001.

Dirección memoria	Instrucción máquina		Explicación
	Nemónico	Hexadecimal	
00A0	LLI $r4, FF$	24FF	Parte baja de la máscara.
00A1	LHI $r4, FF$	34FF	Parte alta de la máscara.
00A2	NAND $r5, r3, r1$	8531	Operación NAND entre dato y H'0001.
00A4	LLI $rD, 3B$	2D3B	Dirección baja de salto si par.
00A5	LHI $rD, A7$	3DA7	Dirección alta de salto si par.
00A6	SUBS $rF, r5, r4$	7F54	Comparar con la máscara.
00A7	BZ	C100	Saltar a A73B si es par.
00A4	LLI $rD, 3C$	2D3C	Dirección baja de salto si impar.
00A5	LHI $rD, B7$	3DB7	Dirección alta de salto si impar.
00AB	BR	C000	Salto incondicional a B73C por ser impar.

El tercer segundo método comprueba si la último bit de  $r3$  es cero, haciendo primero un desplazamiento del número a la derecha (SHR) y efectuando un salto si  $C = 1$ , ya que en este caso el número sería impar. Las instrucciones serían las siguientes:

Dirección memoria	Instrucción máquina		Explicación
	Nemónico	Hexadecimal	
00A0	LLI $rD, 3C$	2D3C	Dirección baja de salto si impar.
00A1	LHI $rD, B7$	3DB7	Dirección alta de salto impar.
00A2	SHR $r3$	A300	Desplazar $r3$ a la derecha.
00A3	BC	C300	Saltar si $C = 1$ (número impar).
00A4	LLI $rD, 3B$	2D3B	Dirección baja de salto si par.
00A5	LLH $rD, A7$	3DA7	Dirección alta de salto si par.
00A6	BR	C000	Salto incondicional de A733 por ser par.

Puede observarse que el segundo método es más eficiente (utiliza menos instrucciones).

### 4.3.5. EJEMPLO DE PROGRAMA

#### Programa de suma de dos tablas

Vamos a realizar un programa para CODE-2 que suma uno a uno los datos de dos tablas,  $T1$  y  $T2$ , y los resultados los incluya en una tercera tabla,  $T3$ . Suponer que  $T1$  comienza en  $d1 = 0040$  y  $T2$  en  $d2 = 0080$ , y  $T3$  debe generarse a partir de  $d3 = 00C0$ . La longitud de las tablas es de  $nT = H'20$  elementos.

#### a) Descripción del algoritmo

El algoritmo, sencillamente, lleva iterativamente el contenido de un elemento  $T1(i)$  de la tabla  $T1$  a un registro y el correspondiente elemento  $T2(i)$  de la tabla  $T2$  a otro registro. Se suman los dos registros mencionados en un tercer registro, y se memoriza este valor en  $T3(i)$ .

En la Figura 4.13 puede verse un organigrama del algoritmo a implementar. Al comienzo del programa se incluye un conjunto de instrucciones que sirven para especificar los parámetros iniciales del programa. Las variables que se utilizan son:

- Número de elementos de la tabla ( $nT$ ).
- Valor  $T1(i)$ , valor  $T2(i)$ , valor  $T3(i) = T1(i) + T2(i)$ .
- Índice  $i$ .

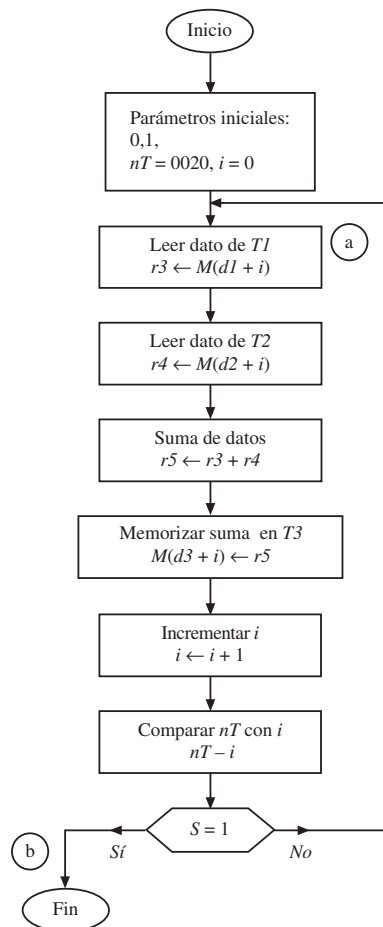


Figura 4.13. Organigrama del programa de suma de dos tablas.



Como las direcciones base (de inicio,  $d1$ ,  $d2$  y  $d3$ ) de las tablas son menores de  $FF$  podemos utilizar direccionamiento indexado para acceder a sus elementos: la dirección de base se incluye en el campo  $v$  de la instrucción y utilizamos  $rC$  como registro índice.

#### b) Asignación de registros y de memoria

En la Tabla 4.5 se incluye la asignación de registros y de posiciones de memoria para los parámetros, variables y el código máquina del programa.

Parámetro o variable	Registro	Posición de memoria	Comentario
0000	$r0$		Para pasar el valor de un registro a otro.
0001	$r1$		Para incrementar el índice $i$ .
$nT$	$r2$		Número de elementos de la tabla.
$T1(i)$	$r3$		Elemento de la tabla $T1$ .
$T2(i)$	$r4$		Elemento de la tabla $T2$ .
$T3(i)$	$r5$		Elemento de la tabla $T3$ .
$i$	$rC$		Registro de indexación.
Programa		0010	Dirección de carga del programa.

**Tabla 4.5.** Asignación de registros y de memoria del programa de suma de dos tablas.

#### c) Redacción del programa en nemónicos

Teniendo en cuenta el repertorio de instrucciones de CODE-2 (Tabla 4.4) y la asignación de memoria, el programa en nemónicos es el que se incluye en la tercera columna de la Tabla 4.6. La primera columna especifica simbólicamente las referencias de salto, de acuerdo con la notación utilizada en el organigrama y la segunda columna las direcciones de memoria en que se ubicaría cada instrucción. También en la quinta columna se incluye un breve comentario sobre la instrucción.

Refer. salto	Dirección memoria	Instrucción máquina		Explicación
		Nemónico	Hexadecimal	
(a)	0010	LLI $r0,00$	2000	Inicializar $r0$ a 0.
	0011	LLI $r1,01$	2101	Inicializar $r1$ a 1.
	0012	LLI $r2,20$	2220	Tamaño de tabla: $nT = H'20$ .
	0013	LLI $rC,00$	2C00	Inicialización del índice: $i = 0$ .
	0014	ADDS $rD,rC,r0$	6DC0	Pasar el índice al registro de dirección.
	0015	LD $r3,[40]$	0340	Llevar a $r3$ el elemento de $T1$ .
	0016	LD $r4,[80]$	0480	Llevar a $r4$ el elemento de $T2$ .
	0017	ADDS $r5,r3,r4$	6534	Suma de elementos de $T1$ y $T2$ en $r5$ .
	0018	ST $[C0],r5$	15C0	Almacenar nuevo elemento de $T3$ .
	0019	ADDS $rC,rC,r1$	6CC1	Incrementar el índice.
	001A	LLI $rD,1F$	2DAF	Cargar en $rD$ dirección de salto (final).
	001B	SUBS $rF,r2,rC$	7F2C	Comparar $nT$ con $i$ .
	001C	BS	C200	Si $S = 1$ el programa concluye, salta a (b).
	001D	LLI $rD,14$	2D14	Cargar en $rD$ dirección de salto (continuar).
(b)	001E	BR	C000	Salto a (a).
	001F	HALT	F000	Final del programa.

**Tabla 4.6.** Programa de suma de dos tablas.

#### d) Codificación del programa en código máquina

La cuarta columna de la Tabla 4.6 contiene el programa codificado en código máquina (hexadecimal). Conviene indicar que el programa en lenguaje máquina está constituido tan solo por la segunda columna (dirección de memoria) y la cuarta columna (código máquina) de la tabla indicada. Las otras columnas sirven como ayuda para facilitar la redacción del programa y mejorar su legibilidad.

## 4.4. Lenguaje ensamblador

El lenguaje máquina tiene dos notables ventajas:

- a) Es directamente interpretable y ejecutable por el procesador.
- b) Los programas se adaptan completamente a los elementos y peculiaridades del computador ya que se programan directamente los elementos físicos de éste: registros, puertos, memoria, etcétera; con lo que se puedan desarrollar programas muy eficientes.

No obstante, las ventajas anteriores quedan oscurecidas por lo siguiente:

- a) Las instrucciones en lenguaje máquina se han de dar en binario o en hexadecimal, siendo su redacción compleja y el programa resultante muy poco *legible*.
- b) El programador ha de realizar la **asignación de registros** y la **asignación de memoria**, decidiendo en qué posiciones colocar las diferentes unidades de que consta un programa (programa principal, subrutinas y cada uno de los datos).

Para solucionar estos dos inconvenientes, manteniendo casi todas las ventajas del lenguaje máquina, se utiliza el **lenguaje ensamblador**. En éste cada instrucción viene identificada mediante un nombre simbólico (nemónico) que, al igual que el código de operación en el lenguaje máquina, identifica las acciones a realizar por la instrucción. Además, en vez de tener necesariamente que especificar cada variable o cada constante indicando el registro o la posición de memoria donde se encuentra, se puede dar un nombre simbólico a cada una de ellas, reduciéndose así notablemente el problema de asignación de registros y de memoria para los datos. Para un usuario los programas resultan así mucho más fáciles de entender y desarrollar. En la Sección 12.12 se incluyen más detalles sobre los lenguajes ensambladores, y en la referencia bibliográfica [17] se describe un ensamblador para CODE-2.

## 4.5. Microprocesadores y microcontroladores

Según indicamos en la Sección 1.2.1, un **microprocesador** es un procesador (CPU) implantado en un circuito integrado (*chip*).

Las funciones que realiza un microprocesador son las típicas de un procesador; es decir:

1. Almacena temporalmente las instrucciones.
2. Decodifica los códigos de operación de las instrucciones y genera las señales de control, tanto para los circuitos internos del propio microprocesador como para los circuitos y dispositivos externos a él.
3. Genera las secuencias de tiempo que sincronizan los intercambios de información entre el microprocesador y su exterior, y que temporizan globalmente al sistema de que forma parte.

4. Contiene un conjunto de registros para el almacenamiento temporal de datos y direcciones.
5. Efectúa las operaciones aritméticas y lógicas típicas de una ALU.

Un **microcontrolador** es un circuito integrado que contiene, total o parcialmente, los cinco elementos básicos de un computador completo (unidad de control, camino de datos y puertos de entrada/salida), estando proyectados para aplicaciones de supervisión, monitorización, gestión y control en sistemas tales como aparatos telefónicos, electrodomésticos, instrumentación médica, control de robots, líneas de ensamblado, etc. Se diferencian de los microprocesadores en que: 1) contienen en su interior no sólo el procesador, sino también otros elementos como puertos de entrada/salida y memoria principal (ampliable externamente) y 2) están orientados a aplicaciones específicas de control. Como ejemplo, el circuito integrado 8051 dispone de 40 terminales (patillas) de entrada/salida, una CPU de 12 MHz, memoria ROM de 4 KB y memoria RAM de 128 Bytes, y 32 líneas de E/S. Usualmente estos circuitos se utilizan para aplicaciones tales como control de semáforos, control en máquinas lavadoras, etc.

Los microprocesadores y microcontroladores son circuitos que, por ser muy versátiles, pueden construirse en grandes series a muy bajo precio. Además de utilizarse para construir computadores (microcomputadores) se utilizan, al igual que los microcontroladores, *embebidos* en multitud de sistemas, de hecho la mayor parte de los sistemas electrónicos digitales actuales se construyen con microprocesadores o microcontroladores, por ser una opción muy económica. Un **sistema embebido** es un sistema controlado por un microprocesador o un microcontrolador cuyo hardware y software están especialmente diseñados y optimizados para resolver un problema concreto. Por lo general estos sistemas interactúan continuamente con su entorno con objeto de monitorizar o controlar algún proceso. Su hardware usualmente se diseña utilizando circuitos integrados comerciales y la mínima circuitería adicional requerida para la aplicación concreta. El término embebido se refiere al hecho de que los circuitos con microprocesador o microcontrolador se encuentran incluidos (embebidos o empotrados) en el interior del sistema que controlan, sin ser externamente aparente su existencia. Hay sistemas embebidos en multitud de electrodomésticos (lavadoras de ropa, lavavajillas, hornos de microondas, etc.), equipos musicales y de vídeo (lectoras/grabadoras de casetes, CD, televisores, vídeos, máquinas de fotos, etc.), juguetes electrónicos (consola de videojuegos), sistemas de control industrial, terminales punto de ventas, comunicaciones de datos (teléfonos inalámbricos, teléfonos móviles, módem, etc.), equipos militares y armamento, etc. Un automóvil moderno puede contener del orden de 100 microprocesadores y microcontroladores embebidos que controlan funciones tales como: encendido, desplazamiento de la transmisión, control de potencia, antibloqueo del freno, control de tracción y seguridad (disparo de la bolsa de aire, etc.). También los computadores de uso general, además de su microprocesador central, contienen diversos microprocesadores en forma de sistemas embebidos que están, por ejemplo, en el teclado, el monitor o en los distintos periféricos.

A diferencia de un computador de uso general, un sistema embebido no es programable por el usuario final de la aplicación. El programa o programas (generalmente grabados en memoria ROM) forma parte integral del sistema y usualmente se ejecuta bajo el control de un sistema operativo extremadamente sencillo.

## 4.6. Procesadores RISC y CISC

Hasta mediados de la década de los ochenta el diseño de los procesadores se realizaba de forma que el repertorio de instrucciones máquina fuese lo más completo posible, ampliándolo sucesivamente, con objeto de que los traductores de lenguajes fuesen lo más sencillos posibles y, además, con la creencia de que también así la ejecución de los programas sería más rápida por la reducción del número de instrucciones a ejecutar. Esta tendencia era seguida por la totalidad de los diseños, incluyendo los computadores DEC VAX y las familias de microprocesadores Intel  $80 \times 86$  y Motorola  $680 \times 0$ . Los re-

peritorios contenían del orden de 200 a 300 instrucciones, muchas de ellas sofisticadas, consumiendo la ejecución de cada una de ellas múltiples ciclos de reloj. Un procesador que sigue esta tendencia se denomina **procesador CISC** (*Complex Instruction Set Computer*, **computador con repertorio de instrucciones complejo**). No obstante, estudios estadísticos sobre la utilización por los traductores de las distintas instrucciones máquina muestran que muchas de ellas apenas se utilizan, por lo que resultaba aconsejable reconsiderar la idea de que un procesador es más potente y rápido cuanto más complejo es. Surge así la tendencia de **procesadores RISC** (*Reduced Instruction Set Computer*, **computador con repertorio de instrucciones reducido**) cuyas peculiaridades más significativas son:

1. En los repertorios se incluyen un reducido número de instrucciones que realizan operaciones muy básicas.
2. Los formatos de las instrucciones son muy regulares (igual longitud, igual tamaño y posición dentro de la instrucción de los distintos campos. Código de operación, dirección, etc.).
3. El procesador contiene un número adecuado de registros (de 16 a 64 o más), realizándose las operaciones de la ALU con los datos de dichos registros. Los intercambios de datos entre los registros y la memoria se efectúan sólo con instrucciones específicas de carga en registro (*load*) y de memorizar el contenido de un registro (*store*).
4. Debido a las propiedades anteriores el desarrollo de la unidad de control RISC es mucho más sencillo que en el caso de un CISC, y se logra que el tiempo de diseño de un nuevo procesador se reduzca notablemente, y que pueda ejecutar, por término medio, una instrucción en tan sólo un ciclo de reloj.

La tendencia RISC es ampliamente utilizada en la actualidad en el diseño de muchos microprocesadores (RISC, MIPS, SPARC, PA-RISC, Alpha, PowerPC, etc.), y en realidad muchas de las ideas surgidas con su desarrollo se utilizan también en los procesadores CISC actuales.

## 4.7. Conclusión

Este capítulo se ha dedicado a estudiar la estructura y funcionamiento básicos de un procesador. La mayor parte de los conceptos se han introducido utilizando un Computador Didáctico Elemental (CODE-2), que a pesar de su sencillez contiene los elementos fundamentales de cualquier computador. Se ha analizado CODE-2 tanto en los niveles de micromáquina como de máquina convencional (lenguaje máquina) (Sección 1.5). Más detalles de este computador, tales como su utilización (carga de programas, etc.), su lenguaje ensamblador y su diseño completo pueden verse en la referencia bibliográfica [17].

### Test



**T4.1.** El biestable de indicador de cero (Z) de una ALU se pone a cero:

- a) Al llevar un dato de valor cero de la memoria principal a un registro del camino de datos.
- b) Al hacer una operación en la ALU cuyo resultado sea cero.
- c) Tanto si se lleva un dato de valor cero de la memoria principal a un registro del camino de datos, como al hacer una operación en la ALU cuyo resultado sea cero.

d) Cuando el valor del contador de programa o del puntero de pila se hacen cero.

**T4.2.** En general, los registros del banco de registros del camino de datos se utilizan (o pueden utilizarse):

- a) Sólo para almacenar datos o resultados con los que opera la ALU.
- b) Para almacenar tanto datos como direcciones.
- c) Para almacenar datos, direcciones e instrucciones del programa.

- d) Para almacenar datos y las direcciones de retorno en las llamadas a subrutinas (caso de pila hardware).

**T4.3.** Suponiendo un computador con una memoria de 128 MBytes de capacidad y que direcciona palabras de memoria de 32 bits, su contador de programa (*PC*) tiene:

- a) 27 bits.
- b) 25 bits.
- c) 32 bits.
- d) Ninguna de las contestaciones anteriores es correcta.

**T4.4.** Suponiendo un computador cuyo puntero de pila (*SP*) tiene una longitud de 32 bits, y cuya memoria está organizada en palabras de 32 bits, ¿cuál sería la capacidad máxima de memoria direccionable?

- a) 4 GBytes.
- b) 16 MBytes.
- c) 16 GBytes.
- d) Ninguna de las contestaciones anteriores es correcta.

**T4.5.** En un computador que tiene una memoria de longitud de palabra de 16 bits y su capacidad máxima puede ser de 256 MB, su puntero de pila (*SP*) debe tener:

- a) 20 bits.
- b) 27 bits.
- c) 28 bits.
- d) Ninguna de las contestaciones anteriores es correcta.

**T4.6.** ¿Cuál de las siguientes afirmaciones es correcta?

- a) Sea cual sea la instrucción, siempre hay una fase de captación de instrucción y otra de ejecución.
- b) Todas las instrucciones tienen fase de ejecución, excepto las de saltos y las de retorno de subrutina.
- c) Todas las instrucciones tienen fase de ejecución, excepto las de salto incondicional.
- d) Todas las instrucciones tienen fase de ejecución, excepto las de salto y llamadas a subrutinas condicionales, en el caso de que no se cumpla la condición.

**T4.7.** Las microoperaciones de la fase de captación de una instrucción:

- a) Dependen del código de operación de la instrucción que se encuentra en el registro instrucción.
- b) Dependen de los biestables indicadores y del código de operación de la instrucción que se encuentra en el registro instrucción.
- c) Dependen del valor del contador de programa.
- d) No depende de nada (son comunes para todas las instrucciones).

**T4.8.** Las microoperaciones generadas en la fase de ejecución de una instrucción:

- a) Dependen del código de operación de la instrucción que se encuentra en el registro instrucción.
- b) Dependen de los valores almacenados en la pila.
- c) Dependen del valor del contador de programa.
- d) No depende de nada (son comunes para todas las instrucciones).

**T4.9.** El procesador utiliza el puntero de pila (*SP*):

- a) En las instrucciones de llamadas y retornos de subrutinas, caso de cumplirse la condición de salto o llamada.
- b) En todo tipo de instrucciones de saltos, y llamadas y retornos a subrutinas.
- c) En todas las instrucciones que tengan al menos dos accesos a memoria.
- d) En todas las instrucciones.

**T4.10.** En el computador descrito en este capítulo, el puntero de pila (*SP*) indica:

- a) La dirección de memoria donde debe saltar el programa después de ejecutarse la instrucción de retorno correspondiente.
- b) La dirección de memoria donde se encuentra la dirección donde debe saltar el programa después de ejecutarse la instrucción de retorno correspondiente.
- c) La dirección de memoria a donde se ha producido el último salto.
- d) La dirección de memoria donde se encuentra la dirección a donde se ha producido la última llamada a una subrutina.

**T4.11.** La instrucción LD *rx*,[*v*] de CODE-2 utiliza 9 ciclos de reloj. Suponiendo que la frecuencia de su reloj es de 1,6 GHz, ¿cuánto se tarda en ejecutar dicha instrucción?

- a) 14,4 nanosegundos.
- b) 8,382 nanosegundos.
- c) 5,625 nanosegundos.
- d) 14,4 microsegundos.

**T4.12.** Suponiendo que el registro *rD* contuviese el valor H'35A0, con la instrucción LD *rA*,[3C] se accedería a la dirección de memoria:

- a) 45A0.
- b) 003C.
- c) 35DC.
- d) 71A0.

**T4.13.** ¿En qué posición de memoria se encontraría el dato que se carga en el registro *rB* al ejecutarse las dos siguientes instrucciones?

LLI *rD*,H'A0  
LD *rB*,[6C]

- a) 00A0.
- b) 004C.
- c) 4CA0.
- d) 010C.

**T4.14.** ¿Qué dato se cargará en el registro *rA* al ejecutarse las dos siguientes instrucciones?

Dirección	Nemónico	Máquina
010B 010C	LLI <i>rD</i> ,H'A0 LD <i>rA</i> ,[6C]	2DA0 0B6C

- a) 010C.
- b) 0B6C.
- c) 4CA0.
- d) No es posible conocerlo con la información que se suministra en el enunciado.

**T4.15.** ¿A que dirección saltará un programa después de ejecutar las siguientes instrucciones?

LLI *rD*,A0  
LHI *rD*,7C  
BR

- a) 7CA0.
- b) A07C.
- c) 011C.
- d) A7C0.

**T4.16.** Suponiendo que el contenido del puerto IP3 es H'AF4F, ¿cuál es el resultado las siguientes cuatro instrucciones?

LLI *rD*,73  
LHI *rD*,5C  
IN *rF*,IP3  
ST [45],*rF*

- a) Almacenar 5CB8 en la posición AF4F de memoria.
- b) Almacenar 5C73 en la posición 45 de memoria.
- c) Almacenar AF4F en la posición 45 de memoria.
- d) Almacenar AF4F en la posición 5CB8 de memoria.

**T4.17.** ¿Qué valor se obtiene en *rF* al ejecutarse las siguientes instrucciones?

LLI *r3*,0F  
LHI *r3*,80  
LLI *r4*,7F  
LHI *r4*,F5  
SUBS *rF*,*r3*,*r4*

- a) 8A90.
- b) 8B90, pero con desbordamiento.
- c) 8A8F.
- d) 8A8F, pero con desbordamiento.

**T4.18.** ¿Qué valor se obtiene en *rF* al ejecutarse las siguientes instrucciones?

LLI *r3*,0F  
LLI *r4*,7F  
ADDS *rF*,*r3*,*r4*

- a) 0080.
- b) 0070.
- c) 008E.
- d) Ninguna de las otras contestaciones es correcta.

**T4.19.** ¿Qué valor se obtiene en *rF* al ejecutarse las siguientes instrucciones?

LLI *r3*,0F  
LLI *r4*,7F  
NAND *rF*,*r3*,*r4*

- a) FFF0.
- b) 00F0.
- c) 000F.
- d) Ninguna de las otras contestaciones es correcta.

**T4.20.** ¿Qué resultados se obtienen al ejecutarse las siguientes instrucciones?

LLI *r3*,FF  
LHI *r3*,FF  
SHL *r3*

- a) *r3* = FFFF; *Z* = 0, *S* = 1, *C* = 1.
- b) *r3* = 7FFF; *Z* = 0, *S* = 0, *C* = 1.
- c) *r3* = 7FFF; *Z* = 0, *S* = 1, *C* = 1.
- d) *r3* = FFFE; *Z* = 0, *S* = 1, *C* = 1.

**T4.21.** ¿Qué resultados se obtienen después de ejecutarse las siguientes instrucciones?

LLI *r3*,FF  
LHI *r3*,FF  
SHR *r3*

- a) *r3* = FFFF; *Z* = 0, *S* = 1, *C* = 1.
- b) *r3* = 7FFF; *Z* = 0, *S* = 0, *C* = 1.
- c) *r3* = 7FFF; *Z* = 0, *S* = 1, *C* = 1.
- d) *r3* = FFFE; *Z* = 0, *S* = 1, *C* = 1.

**T4.22.** ¿Qué resultados se obtienen después de ejecutarse las siguientes instrucciones?

LLI *r3*,FF  
LHI *r3*,FF  
SHRA *r3*

- a) *r3* = FFFF; *Z* = 0, *S* = 1, *C* = 1.
- b) *r3* = 7FFF; *Z* = 0, *S* = 0, *C* = 1.
- c) *r3* = 7FFF; *Z* = 0, *S* = 1, *C* = 1.
- d) *r3* = FFFE; *Z* = 0, *S* = 1, *C* = 1.

**T4.23.** Suponiendo que el contenido del puerto IP2 es H'07AC, y que las siguientes instrucciones se encuentran a partir de la dirección H'AAAA, ¿qué valor tiene el contador de programa después de ejecutar la última instrucción?

IN *rD*,IP2  
BR

- a) H'AAAC.
- b) H'07AC.
- c) H'AAAB.
- d) Ninguna de las otras contestaciones es correcta.

**T4.24.** Suponiendo que las siguientes instrucciones se encuentran a partir de la dirección H'AAFF, ¿qué valor tiene el contador de programa después de ejecutar la última instrucción?

SUBS *rD*,*rA*,*rA*  
CALLZ

- a) H'AB00.
- b) H'AB01.
- c) H'0000.
- d) No es posible conocerlo con tan solo la información que se suministra en el enunciado.

**T4.25.** Suponiendo que las siguientes instrucciones se encuentran almacenadas a partir de la posición H'0010 de me-



moria, ¿qué valor se encuentra almacenado en la posición H'00FE de memoria después de ejecutarlas?

LLI *rE*,FF  
LLI *rD*,00  
CALLR

- a) 0013.
- b) 0012.
- c) 0000.
- d) No es posible conocerlo con tan solo la información que se suministra en el enunciado.

**T4.26.** CODE-2 para ejecutar la instrucción LD *rx*,*v*], que se encuentra en memoria, debe acceder a memoria en total:

- a) Ninguna vez.
- b) Una vez.
- c) Dos veces.
- d) Tres veces.

**T4.27.** CODE-2 para ejecutar la instrucción ST [*v*],*rx*, que se encuentra en memoria, debe acceder a memoria en total:

- a) Ninguna vez.
- b) Una vez.
- c) Dos veces.
- d) Tres veces.

**T4.28.** CODE-2 para ejecutar la instrucción LLI *rx*,*v*, que se encuentra en memoria, debe acceder a memoria en total:

- a) Ninguna vez.
- b) Una vez.
- c) Dos veces.
- d) Tres veces.

**T4.29.** CODE-2 para ejecutar la instrucción IN *rx*,*IPv*, que se encuentra en memoria, debe acceder al bus de datos externo al procesador:

- a) Ninguna vez.
- b) Una vez.
- c) Dos veces.
- d) Tres veces.

**T4.30.** CODE-2 para ejecutar la instrucción SHL *rx*, que se encuentra en memoria, debe acceder a memoria en total:

- a) Ninguna vez.
- b) Una vez.
- c) Dos veces.
- d) Tres veces.

**T4.31.** CODE-2 para ejecutar la instrucción BR, que se encuentra en memoria, debe acceder a memoria en total:

- a) Ninguna vez.
- b) Una vez.
- c) Dos veces.
- d) Tres veces.

**T4.32.** CODE-2 para ejecutar la instrucción CALLR, que se encuentra en memoria, debe acceder a memoria en total:

- a) Ninguna vez.
- b) Una vez.
- c) Dos veces.
- d) Tres veces.

**T4.33.** CODE-2 para ejecutar la instrucción RET, que se encuentra en memoria, debe acceder a memoria en total:

- a) Ninguna vez.
- b) Una vez.
- c) Dos veces.
- d) Tres veces.

**T4.34.** CODE-2 para ejecutar la instrucción HALT, que se encuentra en memoria, debe acceder a memoria en total:

- a) Ninguna vez.
- b) Una vez.
- c) Dos veces.
- d) Tres veces.

**T4.35.** Un microprocesador se define como un circuito integrado que contiene:

- a) La ALU y la memoria caché de un computador.
- b) El camino de datos y la unidad de control de un computador.
- c) La unidad de control y la memoria caché de un computador.
- d) La unidad de control de un computador.

**T4.36.** Un microcontrolador se define como un circuito integrado que contiene:

- a) El camino de datos y parte de la memoria de un computador.
- b) El camino de datos y la unidad de control de un computador.
- c) La unidad de control y parte de la memoria de un computador.
- d) El procesador, parte de la memoria y puertos de E/S de un computador.

**T4.37.** Cuanto más amplio es el repertorio de instrucciones máquina de un procesador:

- a) Más rápido resulta ser.
- b) Más sencillos serán sus traductores de lenguajes de alto nivel a lenguaje máquina.
- c) Más fácil y rápidamente se diseñará dicho procesador.
- d) Menos espacio ocupará dicho procesador.

**T4.38.** Cuanto más reducido y regular es el repertorio de instrucciones máquina de un procesador:

- a) Mayor potencia se logrará.
- b) Más sencillos serán sus traductores de lenguajes de alto nivel.
- c) Más complejo será el diseño de dicho procesador.
- d) Menos espacio ocupará dicho procesador.

## Problemas resueltos



## ELEMENTOS DE UN PROCESADOR

**P4.1.** Un procesador dispone de los siguientes elementos: registro de dirección de memoria (*AR*) de 16 bits, registro de memoria (*DR*) de 8 bits, contador de programa (*PC*), puntero de pila (*SP*), registro de instrucción (*IR*), registros de uso general (*R0*, *R7*) y un registro temporal *RT* para las operaciones con la ALU. Indicar:

- a) Número de hilos (bits) de los buses de datos y de direcciones.
- b) Tamaño en bytes de la memoria principal.
- c) Tamaño en bits del registro *PC*.
- d) Tamaño en bits del registro *SP*.

## SOLUCIÓN

- a) Como el registro de memoria es de 8 bits, y a él está conectado el bus de datos, este bus tendrá 8 hilos. Como el registro de dirección de memoria tiene 16 bits, quiere decir que las direcciones de memoria son de 16 bits, por lo que el bus de dirección de memoria es de 16 bits.
- b) Como las direcciones de memoria se dan con 16 bits, el número máximo de combinaciones (o direcciones) que podemos tener es de:  $2^{16}$  palabras =  $2^6 \cdot 2^{10}$  palabras = 64 Kpalabras. Como cada palabra de datos es de 8 bits,  $C_{MP}(\text{máxima}) = 64 \text{ KB}$ .
- c) Como el registro *PC* (*contador de programa*) almacena una dirección de memoria (donde se encuentra la próxima instrucción a ejecutar), será de 16 bits.
- d) Como el registro *SP* (*puntero de pila*) almacena una dirección de memoria (donde se encuentra la cabecera de la pila), será de 16 bits.

## EJECUCIÓN DE INSTRUCCIONES

**P4.2.** El procesador de un computador de longitud de palabra de 16 bits dispone de los siguientes elementos: registro de dirección de memoria (*DM*), registro de memoria (*RM*), contador de programa (*PC*), puntero de pila (*SP*), registro instrucción (*IR*), conjunto de registros (*R0* a *R7*) y registro interno auxiliar (*RT*). Suponga que la instrucción máquina de resta (código 35C0) se realiza entre *R0* (que actúa de acumulador) y un dato de memoria cuya dirección se encuentra en *R7*, que la resta se realiza en complemento a dos, y que internamente las operaciones de la ALU se realizan con los registros *R0* y *RT*.

Hay los siguientes contenidos iniciales:

M(370A)	=	35C0 ( <i>instrucción</i> )
M(48A0)	=	B732
R0	=	0037
R7	=	48A0

- a) Realice una tabla donde se indiquen las distintas microoperaciones que se realizan al ejecutar la instrucción de la posición 370A de memoria y los contenidos y cambios en: *PC*, *DM*, *RM*, *IR*, *RT*, *R0*, *R7*.
- b) Indicar los contenidos nuevos de los biestables indicadores *CY* (acarreo), *S* (signo), *Z* (cero), *P* (paridad par), *O* (desbordamiento).

## SOLUCIÓN

- a) Las microoperaciones que se generan son las que muestra la tabla siguiente:



$\mu$ operaciones	<i>PC</i>	<i>DM</i>	<i>RM</i>	<i>IR</i>	<i>RT</i>	<i>R0</i>	<i>R7</i>
$DM \leftarrow PC$	370A	370A	—	—	—	0037	48A0
$PC \leftarrow PC + 1$	370B	370A	—	—	—	0037	48A0
$RM \leftarrow M(DM)$	370B	370A	35C0	—	—	0037	48A0
$IR \leftarrow RM$	370B	370A	35C0	35C0	—	0037	48A0
$DM \leftarrow R7$	370B	48A0	35C0	35C0	—	0037	48A0
$RM \leftarrow M(DM)$	370B	48A0	B732	35C0	—	0037	48A0
$Ra \leftarrow RM$	370B	48A0	B732	35C0	B732	0037	48A0
$R0 \leftarrow R0 - Ra$	370B	48A0	B732	35C0	B732	4905	48A0

El resultado final en el acumulador es:

$$\begin{aligned}
 0037 - B732 &= 0037 + C_{(2)}(B732) = 0000\ 0000\ 0011\ 0111 + C_{(2)}(1011\ 0111\ 0011\ 0010) = \\
 &= 1011\ 0111\ 0011\ 0010 + C_{(1)}(1011\ 0111\ 0011\ 0010) + 1 = \\
 &= 1011\ 0111\ 0011\ 0010 + 0100\ 1000\ 1100\ 1101 + 1 = 0100\ 1001\ 0000\ 0101 = 4905_H
 \end{aligned}$$

b) *CY*: No ha habido acarreo  $\Rightarrow CY = 0$ .

*S*:  $R0 > 0 \Rightarrow S = 0$ .

*Z*:  $R0 \neq 0 \Rightarrow Z = 0$ .

*P*: n.º de unos: 5, impar  $\Rightarrow P = 0$ .

*O*: no ha habido desbordamiento  $\Rightarrow O = 0$ .

**P4.3.** El procesador de un ordenador de longitud de palabra 8 bits, representación de enteros en complemento a dos, y criterio del extremo menor, dispone de los siguientes elementos: registro de dirección de memoria (*DM*), registro de memoria (*RM*), contador de programa (*PC*), puntero de pila (*SP*), registro auxiliar de direcciones (*RDA*), registro de instrucción (*IR*), registros de uso general (*R0* a *R15*, actuando *R0* como acumulador) y un acumulador temporal, *At*. Los circuitos de la ALU realizan las operaciones entre *R0* y el acumulador temporal, depositando el resultado en *R0*. La instrucción *resta directa* es una instrucción de 3 bytes, el primer byte es el código de operación *3F* y los dos siguientes contienen la dirección del sustraendo. La instrucción de resta consiste en:

$$R0 \leftarrow R0 - M[dir]$$

Dada la siguiente situación inicial:

$$\begin{aligned}
 PC &= 443D; DM = FFFF; RDA \leftarrow FFFF; RM \leftarrow FF; IR \leftarrow FF; R0 \leftarrow 14; At \leftarrow 00; \\
 V &\leftarrow 1; S \leftarrow 0; Z \leftarrow 1
 \end{aligned}$$

indicar las microoperaciones a realizar, cómo se ven afectados los registros internos del procesador con la ejecución de cada microinstrucción y el valor final de los biestables de estado: desbordamiento (*v*), negativo (*s*) y cero (*z*). Suponer los siguientes contenidos de memoria:

$$\begin{aligned}
 M[443D] &= 3F \\
 M[443E] &= 3D \\
 M[443F] &= 44 \\
 M[4441] &= 17
 \end{aligned}$$

**SOLUCIÓN**

En la tabla que se muestra en la página siguiente se indican las microinstrucciones y cambios.

La operación aritmética que finalmente se hace es:

$$14 - 3F = 14 + 41 = 55$$

quedando los tres biestables indicadores a cero.

**P4.4.** Suponga que en CODE-2 en un momento dado el valor del puntero de pila es *FFFF*, y se hacen las llamadas a subrutinas que se indican resumida y ordenadamente en la tabla. Realizar un esquema que mues-

	$\mu$ operación	PC	AR	DR	IR	RDA	R0	At	V	S	Z
		443D	0000	FF	FF	0000	14	00	1	0	1
Captación de la instrucción	$AR \leftarrow PC$	443D	443D	FF	FF	0000	14	00	1	0	1
	$DR \leftarrow M(AR); PC \leftarrow PC + 1$	443E	443D	3F	FF	0000	14	00	1	0	1
	$IR \leftarrow DR$	443E	443D	3F	3F	0000	14	00	1	0	1
	$AR \leftarrow PC$	443E	443E	3F	3F	0000	14	00	1	0	1
	$DR \leftarrow M(AR); PC \leftarrow PC + 1$	443F	443E	3D	3F	0000	14	00	1	0	1
	$RDA(7:0) \leftarrow DR$	443F	443E	3D	3F	003D	14	00	1	0	1
	$AR \leftarrow PC$	443F	443F	3D	3F	003D	14	00	1	0	1
	$DR \leftarrow M(AR); PC \leftarrow PC + 1$	4440	443F	44	3F	003D	14	00	1	0	1
	$RDA(15:8) \leftarrow DR$	4440	443F	44	3F	443D	14	00	1	0	1
Captación del operando	$AR \leftarrow RDA$	4440	443D	44	3F	443D	14	00	1	0	1
	$DR \leftarrow M(AR)$	4440	443D	3F	3F	443D	14	00	1	0	1
	$At \leftarrow DR$	4440	443D	3F	3F	443D	14	3F	1	0	1
Ejecución	$R0 \leftarrow R0 - At$	4440	443D	3F	3F	443D	55	3F	0	0	0

tre los cambios en el contador de programa, y cómo se iría rellenando y vaciando la pila en las llamadas y retornos, respectivamente.

	Valor de $rD$ en el momento de la llamada	Dirección donde se produce la llamada (CALL)	Dirección donde se produce el retorno (RET)
Programa principal	4CD4	AB00	(programa principal)
Primera subrutina	6F22	6C73	6CA9
Segunda subrutina	BCD4	8FA5	8FCD
Tercera subrutina	(no llama a otra)	(no llama a otra)	BCF6

#### SOLUCIÓN

En la Tabla 4.7 de la página siguiente se detallan todos los pasos seguidos en las llamadas y en los retornos.

- P4.5.** Un computador tiene una memoria principal, organizada en Bytes, siendo su capacidad máxima de memoria 32 MB, y se quiere que el nivel de anidamiento máximo entre subrutinas sea de 256. Estando la memoria pila en las posiciones altas de memoria, indicar en hexadecimal las posiciones de memoria concretas que habrá que reservar para la pila, suponiendo que ésta sólo se utilizará en las llamadas y retornos de subrutinas.

#### SOLUCIÓN

Primero interesa determinar la longitud de las direcciones de memoria.

Como  $32 \text{ MB} = 32 \cdot 2^{20} \text{ B} = 2^{25} \text{ Bytes}$ , las direcciones ocupan 25 bits  $= 25/8 = 3.125 \text{ Bytes}$ .

Como a la memoria se accede por Bytes se ocuparan 4 Bytes por dirección, en lugar de 3.125.

En la memoria pila, suponiendo que sólo se utiliza para almacenar direcciones de memoria, se necesitarán:  $256 \text{ direcciones} \cdot 4 \text{ Bytes/dirección} = 1.024 \text{ Bytes}$  o posiciones de memoria.

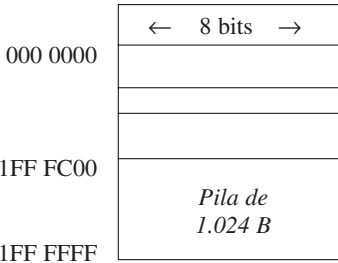
Al ser la capacidad de memoria de 32 Mbytes, sus direcciones irán de la posición  $000\ 0000)_{16}$  a la  $2^{25} - 1 = 1\text{FF FFFF}$ , y como  $1.024)_{10} = 100\ 0000\ 0000)_{2} = 400)_{16}$ , la dirección de comienzo de la memoria pila será:

$$\begin{array}{r}
 1\text{FF FFFF} \\
 - \quad 3\text{FF} \\
 \hline
 1\text{FF FC00}
 \end{array}$$

	Antes de la llamada			Después de la llamada		
	PC	SP	Pila	PC	SP	Pila
Primera llamada	AB01	EFFF	<div><div></div><div></div><div></div><div>EFFE</div><div>XXXX</div></div>	4CD4	EFFE	<div><div></div><div></div><div>EFFF</div><div>AB01</div><div>XXXX</div><div>..</div></div>
Segunda llamada	6C74	EFFE	<div><div></div><div></div><div></div><div>EFFE</div><div>XXXX</div></div>	6F22	EFFD	<div><div></div><div>6C74</div><div>AB01</div><div>XXXX</div><div>..</div></div>
Tercera llamada	8FA6	EFFD	<div><div></div><div>6C74</div><div>AB01</div><div>XXXX</div></div>	BCD4	EFFC	<div><div>8FA6</div><div>6C74</div><div>AB01</div><div>XXXX</div><div>..</div></div>
Primer retorno	BCF7	EFFC	<div><div>8FA6</div><div>6C74</div><div>AB01</div><div>XXXX</div></div>	8FA6	EFFD	<div><div>8FA6</div><div>6C74</div><div>AB01</div><div>XXXX</div><div>..</div></div>
Segundo retorno	8FCE	EFFD	<div><div>1FA6</div><div>AC74</div><div>AB01</div><div>XXXX</div></div>	AC74	EFFE	<div><div>1FA6</div><div>AC74</div><div>AB01</div><div>XXXX</div><div>..</div></div>
Tercer retorno	6CAA	EFFE	<div><div>8FA6</div><div>6C74</div><div>AB01</div><div>XXXX</div></div>	AB01	EFFF	<div><div>8FA6</div><div>6C74</div><div>AB01</div><div>XXXX</div><div>..</div></div>

Tabla 4.7. Seguimiento de llamadas y retornos de subrutinas (P4.4).

En resumen, la organización de las direcciones de la memoria es como se indica en la figura.



CONCEPTOS BÁSICOS DE LENGUAJE MÁQUINA

P4.6. Suponga un computador que dispone de un repertorio de 8 instrucciones, 4 registros de datos (R0, R1, R2 y R3) y que puede direccionar 256 posiciones de memoria. En el repertorio se incluyen instrucciones que

direccionan simultáneamente un registro y una posición de memoria (LD *R1,250* o ADDS *R0,126*; por ejemplo). ¿Cuál sería el tamaño mínimo de estas instrucciones?

#### SOLUCIÓN

El tamaño de las instrucciones vendrá por la suma de los de sus campos; es decir:

- Campo de código de operación: 3 bits, ya que  $2^3 = 8$  instrucciones.
- Campo de dirección de registro: 2 bits, ya que  $2^2 = 4$  registros.
- Campo de dirección de memoria: 8 bits, ya que  $2^8 = 256$  instrucciones.

Con lo que el tamaño mínimo de las instrucciones será:

$$3 + 2 + 8 = 13 \text{ bits}$$

### TRATAMIENTO DE BITS Y PROCESAMIENTO DE TEXTOS

**P4.7.** De la dirección  $I = A000$  a  $F = AFFF$  de la memoria de CODE se encuentra una tabla de datos, que empiezan todos con un bit 0 (es decir, el bit más significativo, MSB, es 0). Hacer un programa en código máquina, a partir de la dirección  $H'0000$ , que cambie los datos de la tabla incluyendo como bit MSB un bit de paridad (criterio par).

#### SOLUCIÓN

##### Algoritmo y organigrama

La idea consiste en contar el número de unos de cada dato, desplazando el dato un bit a la izquierda y comprobando el bit de acarreo. Si el número de unos es impar (es decir, el número de datos acaba en uno), colocamos un 1 en el bit más significativo del dato. Esta última operación la podemos realizar, sencillamente, sumándole al dato original  $H'8000$ . Una vez puesto a 1 el bit más significativo, lo almacenamos en la tabla.

Llamaremos  $pT$  a una variable que recorra las posiciones de la tabla (puntero de tabla).

$D$  al dato que se está analizando.

$NU$  al número de unos.

$ND$  número de desplazamientos realizados.

$I$  a la dirección inicial de la tabla.

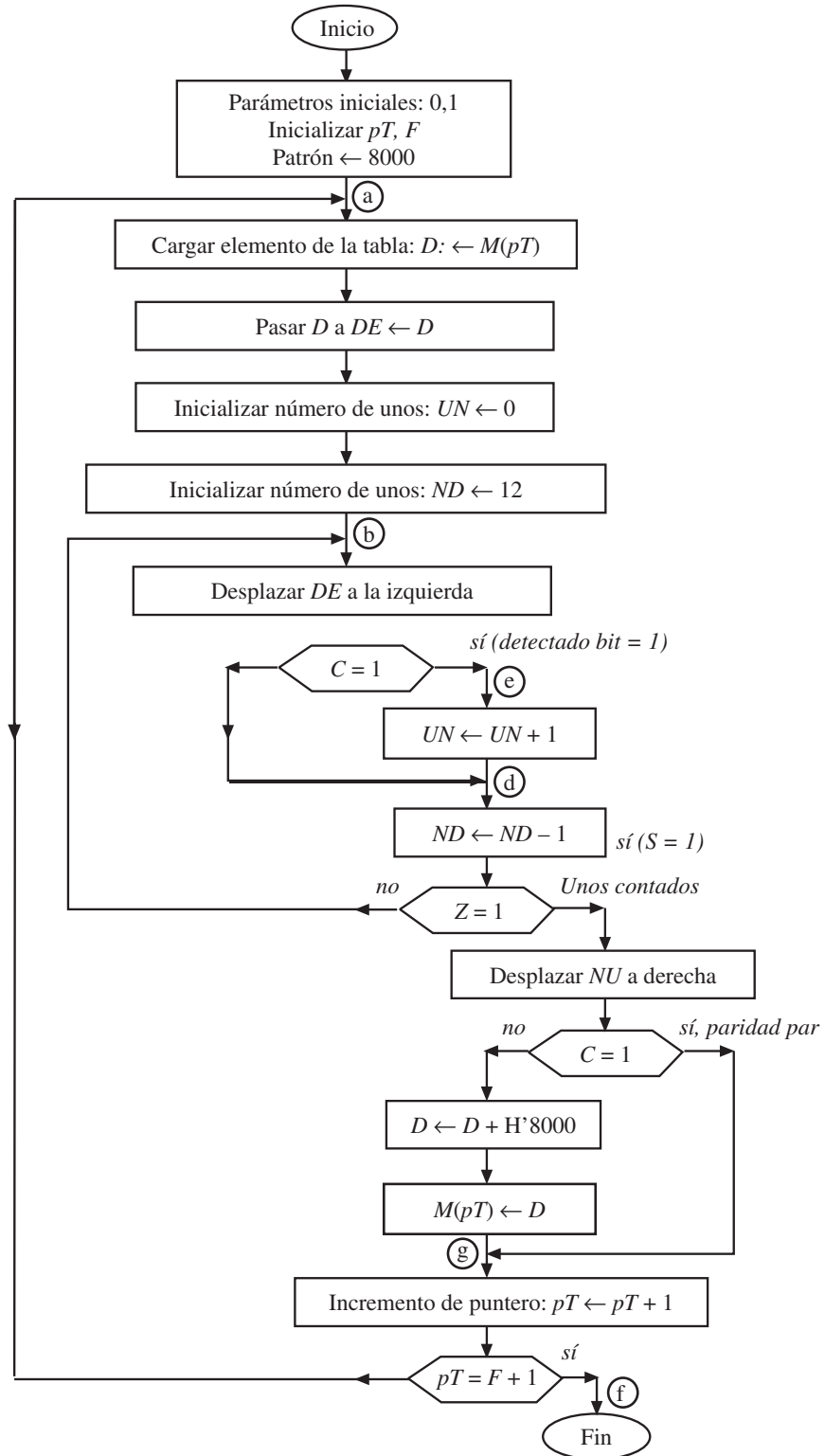
$FI$  a la dirección final de la tabla más 1.

En la Figura 4.14 se muestra el organigrama, que se explica por sí mismo. En la Tabla 4.8 se incluye la asignación de registros y memoria a constantes y variables, y en la Tabla 4.9 el programa correspondiente.

**P4.8.** A partir de la posición  $B000$  de la memoria de CODE se tiene una cadena de caracteres Unicode, que finaliza con un carácter  $CR$  ("retorno de carro"). Hacer un programa que sustituya en la cadena los caracteres punto y coma (;) por coma (,).

Parámetro o variable	Registro	Posición de memoria	Comentario
0000	<i>r0</i>		Para pasar el valor de un registro a otro.
0001	<i>r1</i>		Para incrementar el índice <i>i</i> .
<i>pT</i>	<i>r2</i>		Inicializar con la dirección de inicio de la tabla.
<i>D</i>	<i>r3</i>		Dato de la tabla en análisis.
<i>DE</i>	<i>r4</i>		Dato para contar el número de unos.
<i>ND</i>	<i>r5</i>		Número de desplazamientos que se están haciendo a <i>DE</i> .
<i>NU</i>	<i>r6</i>		Número de unos del dato analizado.
<i>FI</i>	<i>r8</i>		Constante para detectar el final de la tabla, $B000$ .
$H'8000$	<i>r9</i>		Patrón a sumar al dato con número de unos impar para tener paridad par.
Programa		0000	Dirección de carga del programa.

**Tabla 4.8.** Asignación de registros y de memoria del programa del bit de paridad (P4.7).



**Figura 4.14.** Organigrama del programa de introducción de bit de paridad (P4.7).

Refer. salto	Dirección memoria	Instrucción máquina		Explicación
		Nemónico	Hex	
	0000	LLI $r0,00$	2000	Inicializar $r0$ a 0.
	0001	LLI $r1,01$	2101	Inicializar $r1$ a 1.
	0002	LLI $r2,00$	2D10	Dirección de inicio de tabla en $rD$ .
	0003	LHI $r2,A0$	32A0	
	0004	LLI $r8,00$	2800	Constante para detectar el final de la tabla.
	0005	LHI $r8,A0$	38A0	
	0006	LLI $r9,00$	2900	Constante para sumar a números para paridad par.
	0007	LHI $r9,80$	3980	
(a)	0008	ADDS $rD,r2,r0$	6D20	$rD \leftarrow r2$ .
	0009	LD $r3,[00]$	0200	Carga del dato $D$ en $r3$ .
	000A	ADDS $r4,r3,r0$	6430	$r4 \leftarrow r3$ .
	000B	LLI $r5,0C$	250C	$r5 \leftarrow 0C = D'12$ ; inicializar n.º de desplazamientos.
	000C	LLI $r6,00$	2600	$r6 \leftarrow 00$ ; inicializar n.º de unos del dato.
(b)	000D	SHL $r4$	9400	Desplazar $DE$ un lugar a izquierda.
	000E	LLI $rD,1D$	2D1D	Dirección de salto (e).
	000F	BC	C300	Saltar a (e) si $C = 1$ .
(d)	0010	SUBS $r5,r5,r1$	7551	$r5 \leftarrow r5 - 1$ .
	0011	LLI $rD,0D$	2D0D	Dirección de salto (b).
	0012	BZ	C100	Saltar a (b) si no se ha acabado de contar unos.
	0013	SHR $r6$	A600	Desplazar $NU$ para ver si es impar.
	0014	LLI $rD,19$	2D19	Dirección de salto (g).
	0015	BC	C300	Saltar a (g) si $C = 1$ .
	0016	ADDS $r3,r3,r9$	6339	Poner a 1 el MSB; $D \leftarrow D + H'8000$ .
	0017	ADDS $rD,r2,r0$	6D20	$rD \leftarrow r2$ .
	0018	ST $[00],r3$	1300	Almacenar el dato con su bit de paridad (uno).
(g)	0019	ADDS $r2,r2,r1$	6221	$pT \leftarrow pT + 1$ .
	001A	SUBS $rF,r2,r8$	7F28	Si la resta es 0 hemos acabado.
	001B	LLI $rD,22$	2D22	Dirección de salto (f).
	001C	BZ	C100	Ir al final.
	001D	LLI $rD,08$	2DA3	Dirección de (a).
	001E	BR	C000	Salto incondicional a (a).
(e)	001F	ADDS $r6,r6,r1$	6661	Se detectó un uno, nacemos $NU \leftarrow NU + 1$ .
	0020	LLI $rD,10$	2B10	Dirección de (d).
	0021	BR	C000	Salto incondicional a (d).
(f)	0022	HALT	FFFF	Final.

Tabla 4.9. Programa de introducción de bit de paridad (P4.7).

## SOLUCIÓN

En la Tabla 4.10 indicamos las constantes y variables a utilizar.

Parámetro o variable	Significado
0000	Constante cero.
0001	Para incrementar el puntero de la cadena.
$pT$	Inicializar con la dirección de inicio de la cadena $B000$ .
$C$	Carácter en análisis.
$CR$	Código de carácter de retorno de carro: $000C$ .
$PYC$	Código del carácter punto y coma: $003B$ .
$CM$	Código del carácter coma: $002C$ .

Tabla 4.10. Asignación de registros y de memoria del programa de cambio de caracteres (P4.8).

En la Figura 4.15 se muestra el organigrama, que se explica por sí mismo. En la Tabla 4.11 se incluye la asignación de registros y memoria a constantes y variables.

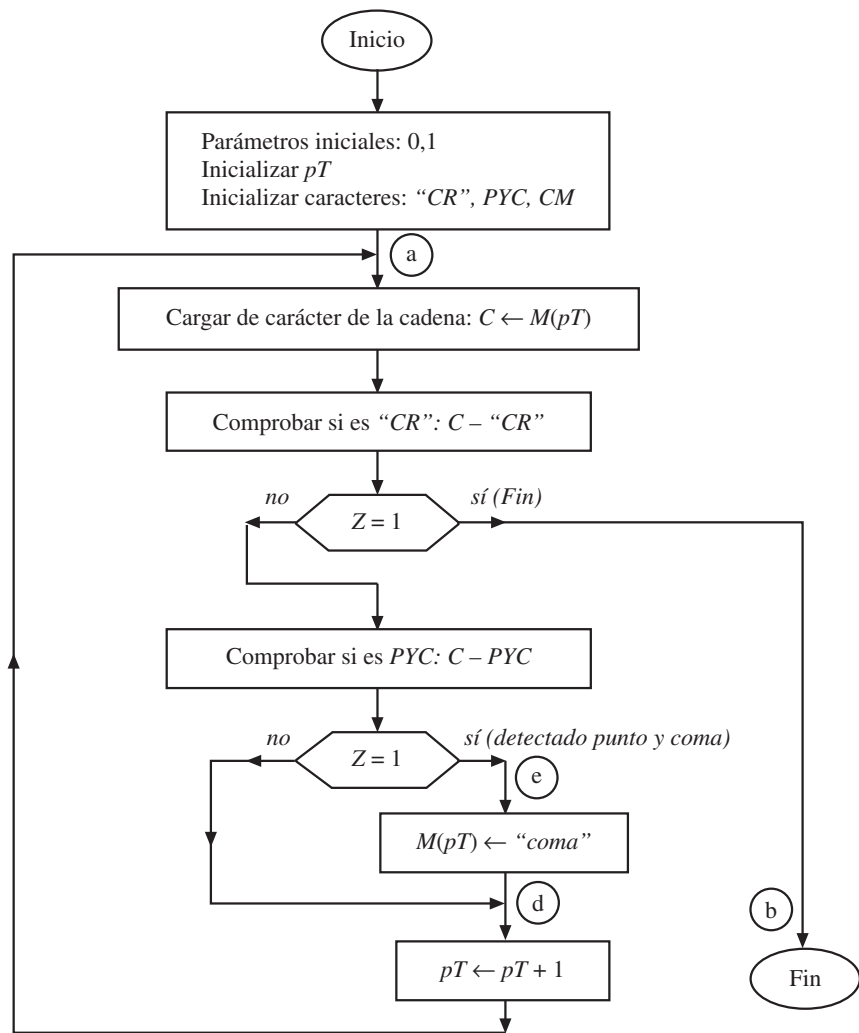


Figura 4.15. Organigrama del programa de cambio de caracteres (P4.8).

Parámetro o variable	Registro	Posición de memoria	Comentario
0001 pT C CR PYC CM	r1 r2 r3 r5 r6 r8		Para incrementar el puntero de la cadena. Inicializar con la dirección de inicio de la cadena B000. Carácter en análisis. Código de carácter de retorno de carro: 000C. Código del carácter punto y coma: 003B. Código del carácter coma: 002C.
Programa		0000	Dirección de carga del programa.

Tabla 4.11. Asignación de registros y de memoria del programa de cambio de caracteres (P4.8).

El programa en nemónicos y en lenguaje máquina puede verse en la Tabla 4.12.

Refer. salto	Dirección memoria	Instrucción máquina		Explicación
		Nemónico	Hex	
	0000	LLI $r0,00$	2000	Inicializar $r0$ a 0000.
	0001	LLI $r1,01$	2101	Inicializar $r1$ a 0001.
	0002	LLI $r2,00$	2D10	Dirección (B000) de inicio de tabla en $r2$ .
	0003	LHI $r2,B0$	32A0	
	0004	LLI $r5,0C$	250C	Retorno de carro.
	0005	LHI $r6,3B$	263B	Punto y coma.
	0006	LLI $r8,2C$	282C	Coma.
(a)	0007	ADDS $rD,r2,r0$	6D20	$rD \leftarrow pT$ .
	0008	LD $r3,[00]$	0300	Carga del carácter $r3$ .
	0009	SUBS $rF,r3,r5$	7F35	Comprobar si es carácter "CR".
	000A	LLI $rD,16$	2D16	Dirección de salto (b).
	000B	BZ	C100	Salta a fin (b).
	000C	SUBS $rF,r3,r6$	7F36	Comprobar si carácter es punto y coma.
	000D	LLI $rD,12$	2D12	Dirección de salto (e).
(d)	000E	BZ	C100	Salta a (e).
	000F	ADDS $r2,r2,r1$	6221	$pT \leftarrow pT + 1$ .
	0010	LLI $rD,07$	2D07	Dirección de salto (a).
(e)	0011	BR	C000	Salto incondicional a (a).
	0012	ADDS $rD,r2,r0$	6D20	$rD \leftarrow pT$ .
	0013	ST $[00], r8$	1800	Cambiar en la cadena punto y coma por coma.
	0014	LLI $rD,0F$	2D0F	Dirección de salto (d).
(b)	0015	BR	C000	Salto incondicional a (d).
	0016	HALT	FFFF	Fin.

**Tabla 4.12.** Programa de cambio de caracteres (P4.8).

## TABLAS

**P4.9.** Dada una tabla de números que se inicia en la posición  $5F3C$  y acaba en la posición  $F000$ , hacer un programa para CODE-2 que proporcione por el puerto de salida  $OP2$  el máximo y el mínimo de esta tabla, y por el puerto de salida  $OP1$  los mensajes 0000 y FFFF, respectivamente.

### SOLUCIÓN

En la Tabla 4.13 indicamos las constantes y variables a utilizar.

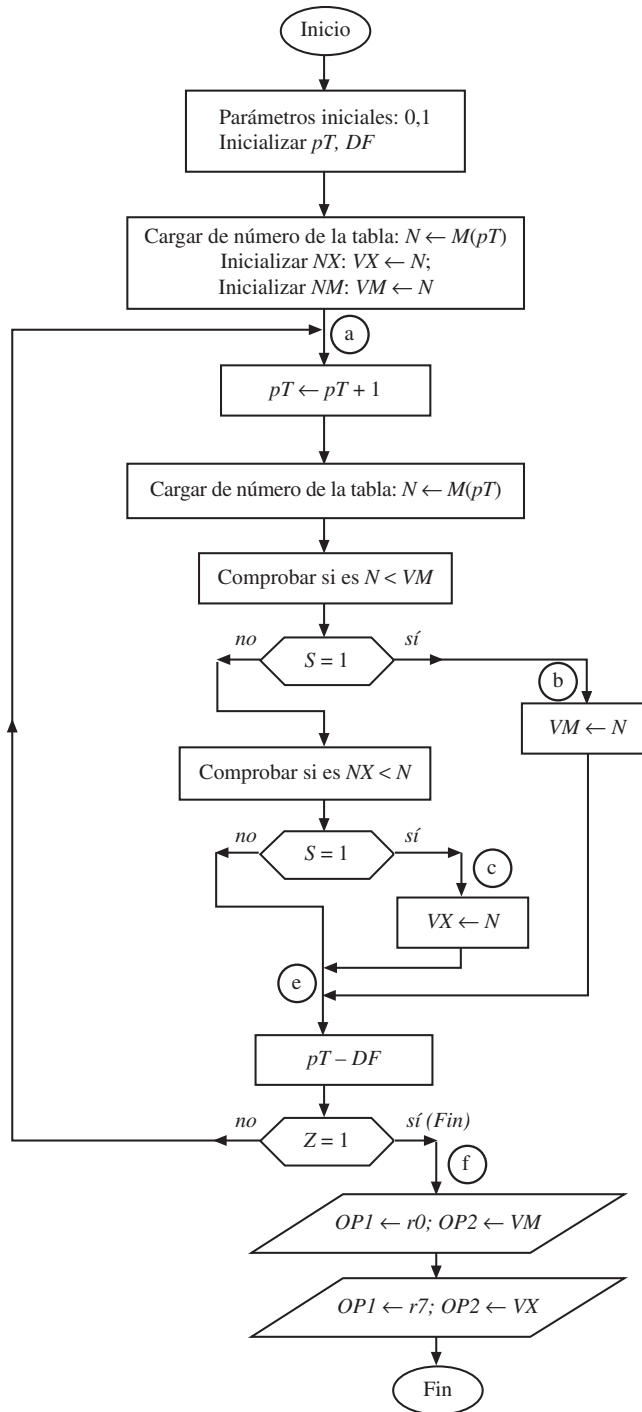
El algoritmo consiste en definir dos variables  $NM$  y  $NX$  que almacenen el valor máximo y mínimo de la tabla, respectivamente. Los dos valores los inicializamos con el primer número de la tabla. Luego reco-

Parámetro o variable	Comentario
0000	Constante cero.
0001	Para incrementar el puntero de la cadena.
$pT$	Inicializar con la dirección de inicio de la cadena $5F3C$ .
$N$	Número en análisis.
$VX$	Valor máximo.
$VM$	Valor mínimo.
$DF$	Dirección final, $F000$ .
FFFF	Patrón de máximo.
Programa	Dirección de carga del programa.

**Tabla 4.13.** Asignación de registros y de memoria del programa del máximo y del mínimo.



remos uno a uno los elementos de la tabla, comprobando si son menores que el mínimo o mayores que el máximo, en caso que se cumpla una u otra condición cambiamos los valores correspondientes de  $NM$  o  $NX$ , respectivamente. En la Figura 4.16 se muestra el organigrama, que se explica por sí mismo. En la Tabla 4.14 se incluye la asignación de registros y memoria a constantes y variables.



**Figura 4.16.** Organigrama del programa del máximo y del mínimo (Problema P4.9).

Parámetro o variable	Registro	Posición de memoria	Comentario
0000	$r0$		Constante cero.
0001	$r1$		Para incrementar el puntero de la cadena.
$pT$	$r2$		Inicializar con la dirección de inicio de la cadena $5F3C$ .
$N$	$r3$		Número en análisis.
$VX$	$r4$		Valor máximo.
$VM$	$r5$		Valor mínimo.
$DF$	$r6$		Dirección final, $F000$ .
FFFF	$r7$		Patrón de máximo, $FFFF$ .
Programa		0000	Dirección de carga del programa.

**Tabla 4.14.** Asignación de registros y de memoria del programa del máximo y del mínimo.

El programa en némonicos y en lenguaje máquina puede verse en la Tabla 4.15.

Refer. salto	Dirección memoria	Instrucción máquina		Explicación
		Nemónico	Hex	
	0000	LLI $r0,00$	2000	Inicializar $r0$ a 0000.
	0001	LLI $r1,01$	2101	Inicializar $r1$ a 0001.
	0002	LLI $r2,3C$	2D3C	Dirección de inicio de tabla de números: $5F3C$ .
	0003	LHI $r2,5F$	325F	
	0004	LLI $r6,00$	2500	Dirección final de la tabla: $F000$ .
	0005	LHI $r6,F0$	36F0	
	0006	ADDS $rD,r2,r0$	6D20	$rD \leftarrow pT$ .
	0007	LD $r4,[00]$	0400	Cargar primer número en $r4$ , como $VX$ .
	0008	ADDS $r5,r4,r0$	6540	Cargar primer número como $VM$ .
(a)	0009	ADDS $r2,r2,r1$	6221	$pT \leftarrow pT + 1$ .
	000A	ADDS $rD,r2,r0$	6D20	$rD \leftarrow pT$ .
	000B	LD $r3,[00]$	0300	Cargar número de la tabla en $r3$ .
	000C	SUBS $rF, r3, r5$	7F35	$N - VM$ .
	000D	LLI $rD,17$	2D17	Dirección de salto (b).
	000E	BS	C200	Salta a (b) si hay nuevo mínimo.
	000F	SUBS $rF, r4, r3$	7F43	$VX - N$ .
	0010	LLI $rD,1A$	2D1A	Dirección de salto (c).
(e)	0011	BS	C200	Saltar a (c) si hay nuevo máximo.
	0012	SUBS $rF,r2,r6$	7F26	$r2 - r6$ .
	0013	LLI $rD,1D$	2D1D	Dirección de salto (f) (final).
	0014	BZ	C100	Saltar a (f) si se ha llegado al final de la tabla.
	0015	LLI $rD,09$	2D09	Dirección de salto (a).
(b)	0016	BR	C 000	Salto incondicional a (a).
	0017	ADDS $r5,r3,r0$	6530	Nuevo mínimo: $VM \leftarrow N$ .
	0018	LLI $rD,12$	2D12	Dirección de salto (e).
(c)	0019	BR	C000	Salto incondicional a (e).
	001A	ADDS $r4,r3,r0$	6430	Nuevo máximo: $VX \leftarrow N$ .
	001B	LLI $rD,12$	2D12	Dirección de salto (e).
(f)	001C	BR	C000	Salto incondicional a (e).
	001D	OUT OP1, $r0$	5001	Aviso de mínimo.
	001E	OUT OP2, $r5$	5502	Salida de mínimo.
	001F	OUT OP1, $r7$	5701	Aviso de máximo.
	0020	OUT OP2, $r4$	5402	Salida de máximo.
	0021	HALT	FFFF	Fin.

**Tabla 4.15.** Programa de obtención del máximo y del mínimo de una tabla de números.

## ORDENACIÓN Y BÚSQUEDAS

- P4.10.** Entre la dirección *C000* y *CD88* de la memoria de CODE hay almacenada una lista de caracteres Unicode. Hacer un programa que busque la cadena “ANA” y muestre en el puerto de salida OP1 la dirección donde comienza.

## SOLUCIÓN

El algoritmo consiste en ir recorriendo con un puntero, *pL*, los elementos de la lista o cadena de caracteres y detectar si el carácter leído es una *A*, caso de que así lo sea, almacenamos en una variable, *dC* (dirección de cadena) la dirección del puntero en ese momento, que es donde podría empezar la cadena. En el caso de que el primer carácter analizado no sea *A* se incrementa el puntero de la lista. Caso de haber encontrado una *A* y no ser los siguientes *N* y *A*, se pasa a analizar si el carácter es una *A*, repitiéndose la iteración.

En la Tabla 4.16 indicamos las constantes y variables a utilizar.

Parámetro o variable	Comentario
0000	Constante cero.
0001	Para incrementar el puntero de la cadena.
<i>pL</i>	Inicializar con la dirección de inicio de la lista <i>C000</i> .
<i>C</i>	Carácter en análisis.
<i>A</i>	Código Unicode de la “A”, <i>0041</i> .
<i>N</i>	Código Unicode de la “N”, <i>004E</i> .
<i>dF</i>	Dirección final, <i>CD88</i> .
FFFF	Patrón para indicar que no se ha encontrado la cadena “ANA”.
<i>dC</i>	Dirección de inicio de la cadena “ANA”.

**Tabla 4.16.** Variables y constantes utilizadas en el programa de búsqueda de cadena “ANA”.

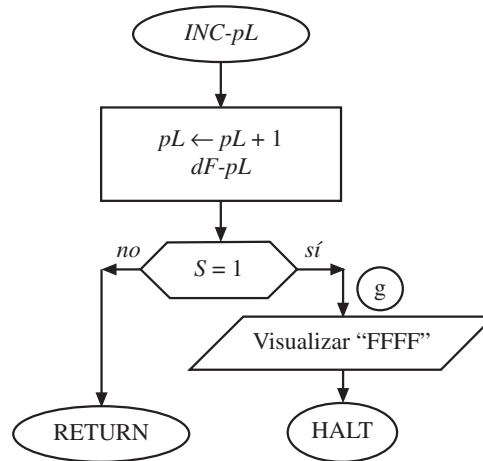
En la Figura 4.17 se muestra el organigrama, que se explica por sí mismo. Como en tres puntos distintos hay que incrementar el puntero de la lista *pL* y detectar si se ha llegado al final (*pL* > *dF*), hemos incluido estas operaciones en la subrutina *INC-pL* (Figura 4.18 de la página 148). Caso de que se llegue al final de la lista incrementando el puntero, quiere decir que en la lista no se encuentra la cadena “ANA”, hecho que ponemos de manifiesto escribiendo en el puerto de salida OP02 el texto FFFF.

En la Tabla 4.17 se incluye la asignación de registros y memoria a constantes y variables.

Parámetro o variable	Registro	Posición de memoria	Comentario
0000	<i>r0</i>		Constante cero.
0001	<i>r1</i>		Para incrementar el puntero de la cadena.
<i>pL</i>	<i>r2</i>		Inicializar con la dirección de inicio de la lista <i>C000</i> .
<i>C</i>	<i>r3</i>		Carácter en análisis.
<i>A</i>	<i>r4</i>		Código unicode de la “A”, <i>0041</i> .
<i>N</i>	<i>r5</i>		Código unicode de la “N”, <i>004E</i> .
<i>dF</i>	<i>r6</i>		Dirección final, <i>CD88</i> .
FFFF	<i>r7</i>		Patrón para indicar que no se ha encontrado la cadena “ANA”.
<i>dC</i>	<i>r8</i>		Dirección de inicio de la cadena “ANA”.
Programa		0000	Dirección de carga del programa.

**Tabla 4.17.** Asignación de registros y de memoria en el programa de búsqueda de cadena “ANA”.





**Figura 4.18.** Organigrama de la subrutina del programa de búsqueda de una cadena (Problema P4.10).

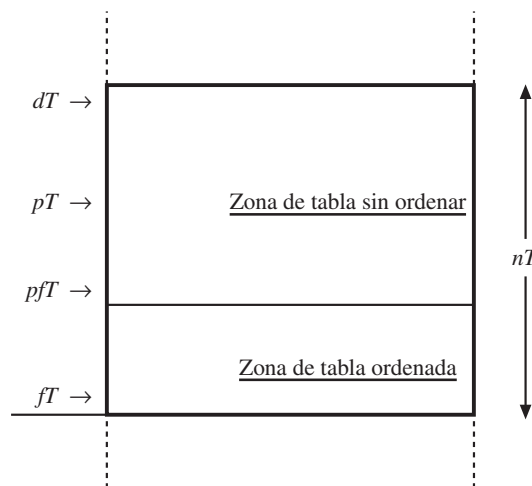
El programa en némonicos y en lenguaje máquina puede verse en la Tabla 4.18.

- P4.11.** Realizar una rutina para CODE-2 que ubique los elementos de una tabla en orden creciente, de menor a mayor. En la posición  $H'0010$  de memoria se encuentra la dirección de comienzo de la tabla, y en  $H'0011$  su tamaño, y la rutina debe empezar en  $H'00A0$ .

#### SOLUCIÓN

##### a) Descripción del algoritmo

Vamos a resolver este problema utilizando el **algoritmo de ordenación por burbujas**, que consiste en detectar el elemento mayor de la tabla y ubicarlo al final de ella, con lo que éste queda ordenado; después se vuelven a analizar el resto de elementos no ordenados de la tabla y el nuevo máximo se coloca al final, justo inmediatamente antes de la zona previamente ordenada, y así sucesivamente (Figura 4.19). Cada vez que se encuentra un máximo de la zona desordenada, aumenta en una posición la zona ordenada y disminuye en dicha unidad la zona desordenada. Si la tabla contiene  $nT$  elementos habrá que recorrer la tabla  $nT - 1$  veces. Para seleccionar cada máximo, sencillamente se compara, desde el principio, cada elemento,  $T(i)$ ,  $i = 2, \dots, nT$ , con el anterior,  $T(i - 1)$ , y si  $T(i - 1)$  es mayor que  $T(i)$  se intercambian en la memoria, de esta forma al recorrer los elementos no ordenados de la tabla, el último será el nuevo máximo y quedará por tanto en su sitio, como primero de la nueva zona ordenada.



**Figura 4.19.** Significado de las variables utilizadas en el programa de ordenación.

Refer. salto	Dirección memoria	Instrucción máquina		Explicación
		Nemónico	Hex	
	0000	LLI $r0,00$	2000	Inicializar $r0$ a 0000.
	0001	LLI $r1,01$	2101	Inicializar $r1$ a 0001.
	0002	LLI $r2,3C$	2D00	Dirección de inicio de lista de caracteres: C000.
	0003	LHI $r2,5F$	32C0	
	0004	LLI $r6,00$	2588	Dirección final de la tabla: CD88.
	0005	LHI $r6,F0$	36CD	
	0006	LLI $r4,41$	2441	Carácter “A”, 0041.
	0007	LLI $r5,4E$	254E	Carácter “N” 004E.
	0008	LLi $rE,FF$	2EFF	Inicializar el puntero de pila (registro $rE$ ).
	0009	LHI $rE,FF$	3EFF	
	000A	LLI $r7,FF$	6430	Letrero “FFFF”.
	000B	LHI $r7,FF$	37FF	
(a)	000C	ADDS $rD,r2,r0$	6D20	$rD \leftarrow pL$ .
	000D	LD $r3,[00]$	0300	Cargar carácter en $r3$ .
(b)	000E	SUBS $rF,r3,r4$	7F34	Comprobar si es “A”.
	000F	LLI $rD,13$	2D15	Dirección de salto (c).
	0010	BZ	C100	Si es “A” saltar a (c).
	0011	LLI $rD,30$	2D30	Dirección de la subrutina.
	0012	CALLR	D000	Llamar a subrutina de gestión de $pL$ .
	0013	LLI $rD,0A$	2D0C	Dirección de salto (a).
	0014	BR	C000	Salto incondicional a (a).
(c)	0015	ADDS $r8,r2,r0$	6820	$dC \leftarrow pL$ .
	0016	LLI $rD,30$	2D30	Dirección de la subrutina.
	0017	CALLR	D000	Llamar a la subrutina de gestión de $pL$ .
	0018	ADDS $rD,r2,r0$	6D20	$rD \leftarrow pL$ .
	0019	LD $r3,[00]$	0300	Cargar carácter en $r3$ .
	001A	SUBS $rF,r3,r5$	7F35	Comprobar si es “N”.
	001B	LLI $rD,1D$	2D1F	Dirección de salto (d).
	001C	BZ	C100	Si es “N” saltar a (d).
	001D	LLI, $rD,0C$	2D0E	Dirección de salto (b).
	001E	BR	C000	Salto incondicional a (b).
(d)	001F	LLI $rD,30$	2D30	Dirección de la subrutina.
	0020	CALLR	D000	Llamar a la subrutina de gestión de $pL$ .
	0021	ADDS $rD,r2,r0$	6D20	$rD \leftarrow pL$ .
	0022	LD $r3,[00]$	0300	Cargar carácter en $r3$ .
	0023	SUBS $rF,r3,r4$	7F34	Comprobar si es “A”.
	0024	LLI $rD,26$	2D28	Dirección de salto (f).
	0025	BZ	C100	Si es “A” saltar a (f).
	0026	LLI $rD,0C$	2D0E	Dirección de salto (b).
	0027	BR	C000	Salto incondicional a (b).
(f)	0028	OUT $OP02,r8$	5802	Visualizar en $OP02$ la dirección de inicio de la cadena).
	0029	HALT	FFFF	Fin del programa.

Subrutina de gestión del puntero de la lista ( $INC-pL$ )

$INC-pL$	0030	ADDS $r2,r2,r1$	6221	$pL \leftarrow pL + 1$ .
	0031	SUBS $rF,r6,r2$	7F62	$dF - pL$ .
	0032	LLI $rD,35$	2D35	Dirección de salto (g).
	0033	BS	C200	Saltar si ha llegado al final.
	0034	RET	E000	Retornar al punto de llamada.
(g)	0035	OUT $OP02,r7$	5702	Visualizar el letrero “FFFF” (cadena no encontrada).
	0036	HALT	FFFF	Fin del programa.

**Tabla 4.18.** Programa de búsqueda de la cadena “ANA” (Problema P4.10).

Las variables que vamos a utilizar son (Figura 4.19):

$dT$ : Posición inicial de la tabla.

$pT$ : Puntero de la tabla (posición en memoria del elemento de la tabla al que se accede:  $dT + i$ ).

$pfT$ : Puntero de la posición del último elemento de la zona desordenada, que inicialmente será el último elemento de la tabla; es decir:  $pfT = dT + nT$ .

En la Figura 4.20 puede verse un organigrama del algoritmo a implementar. Al comienzo de la rutina se incluye un conjunto de instrucciones que sirven para especificar los parámetros iniciales del programa. En este caso se introducen las constantes 0 y 1, y se inicializan las variables  $dT$  (leyéndola de la posición 0010 de memoria) y  $pfT$  (sumando a la posición inicial de la tabla su tamaño). A continuación se entra en el bucle principal del programa, que se ejecutará  $nT - 1$  veces, una por cada vez que se obtiene un nuevo máximo de la zona no ordenada. El bucle interior recorre con el puntero  $pT$  la zona no ordenada de la tabla; es decir, de la posición  $dT$  a la  $pfT$ , dejando el nuevo máximo en esta última posición ( $pfT$ ).

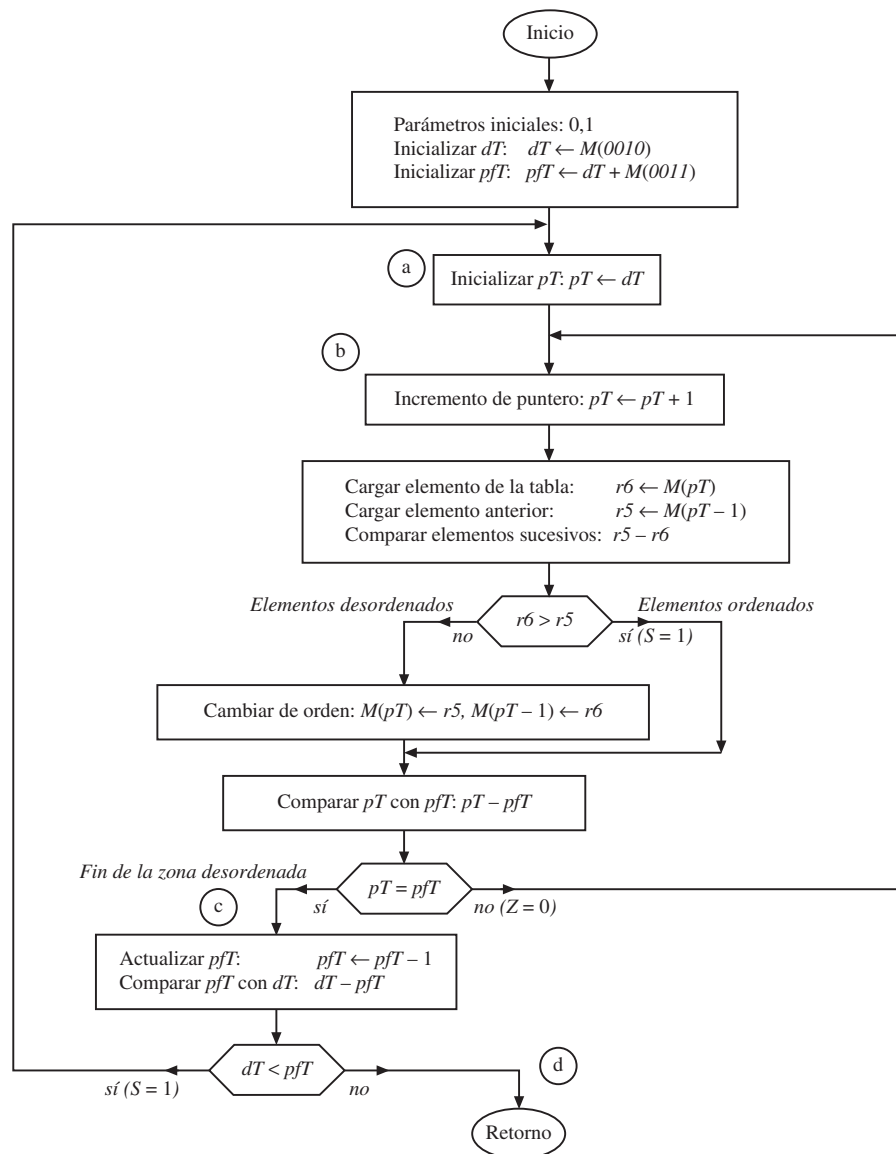


Figura 4.20. Organigrama del programa de ordenación.

Como el programa se configura como una rutina, finaliza con una instrucción RET. De acuerdo con el enunciado del problema, el programa que llame a esta rutina debe ubicar en la posición  $H'0010$  de memoria la dirección de comienzo de la tabla, y en  $H'0011$  su tamaño. Una vez realizado lo anterior se llamará a la rutina con las dos siguientes instrucciones: LLI  $rD, A0$  y CALLR.

b) *Asignación de registros y de memoria*

En la Tabla 4.19 se incluye la asignación de registros y de posiciones de memoria para los parámetros, variables y el código máquina del programa.

Parámetro o variable	Registro	Posición de memoria	Comentario
0000	$r0$		Para pasar el valor de un registro a otro.
0001	$r1$		Para incrementar el índice $i$ .
$dT$	$r2$		Dirección de inicio de la tabla.
$pfT$	$r3$		Dirección del final de la zona no ordenada.
$pT$	$r4$		Puntero que recorre la zona no ordenada.
$M(pT - 1)$	$r5$		Variable que contiene un elemento de la tabla.
$M(pT)$	$r6$		Variable que contiene el siguiente elemento de la tabla.
Programa		00A0	Dirección de carga del programa.

**Tabla 4.19.** Asignación de registros y de memoria del programa de ordenación.

c) *Redacción del programa en nemónicos*

Teniendo en cuenta el repertorio de instrucciones de CODE-2 (Tabla 4.4) y la asignación de memoria, el programa en nemónicos es el que se incluye en la tercera columna de la Tabla 4.20. La primera columna especifica simbólicamente las referencias de salto, de acuerdo con la notación utilizada en el organigrama y la segunda columna las direcciones de memoria en que se ubicaría cada instrucción. También en la quinta columna se incluye un breve comentario sobre la instrucción.

Refer. salto	Dirección memoria	Instrucción máquina		Explicación
		Nemónico	Hex	
	00A0	LLI $r0, 00$	2000	Inicializar $r0$ a 0.
	00A1	LLI $r1, 01$	2101	Inicializar $r1$ a 1.
	00A2	LLI $rD, 10$	2D10	Cargar dirección de inicio de tabla en $rD$ .
	00A3	LD $r2, [00]$	0200	Cargar $dT$ en $r2$ .
	00A4	LD $r3, [01]$	0301	Cargar tamaño de tabla en $r3$ .
	00A5	ADDS $r3, r3, r2$	6332	Obtener dirección final de tabla ( $pfT$ ).
(a)	00A6	ADDS $r4, r2, r0$	6420	Iniciar $pT$ para la búsqueda de un máximo.
(b)	00A7	ADDS $r3, r3, r1$	6331	Incrementar $pT$ .
	00A8	ADDS $rD, r4, r0$	6D40	Llevar $pT$ a registro de dirección
	00A9	LD $r6, [00]$	0600	Cargar elemento $pT$ de tabla en $r6$ .
	00AA	SUBS $rD, rD, r1$	7DD1	Decrementar la dirección.
	00AB	LD $r5, [00]$	0500	Cargar elemento $pT-1$ de tabla en $r5$ .
	00AC	LLI $rD, B3$	2DA7	Llevar a $rD$ dirección (e) de salto.
	00AD	SUBS $rF, r5, r6$	7F56	Comparar elementos $pT-1$ y $pT$ de la tabla.
	00AE	BS	C200	Saltar a (e) si están ordenados.
	00AF	ADDS $rD, r4, r0$	6D40	Llevar $pT$ a registro de dirección.
	00B0	ST $[00], r5$	1500	Cambio de orden en la tabla.
	00B1	SUBS $rD, rD, r1$	7DD1	Obtener $pT$ en $rD$ .
	00B2	ST $[00], r6$	1600	Cambio de orden en la tabla.

**Tabla 4.20.** Programa de ordenación por el método de las burbujas.



Refer. salto	Dirección memoria	Instrucción máquina		Explicación
		Nemónico	Hex	
(e)	00B3	SUBS $rF, r3, r4$	7F34	Comparara $pfT$ con $pT$ .
	00B4	LLI $rD, B8$	2DB8	Cargar en $rD$ dirección (c) de salto.
	00B5	BZ	C100	Saltar a (c) si $pfT = pT$ .
	00B6	LLI $rD, A7$	2DA7	Cargar en $rD$ dirección (b).
	00B7	BR	C000	Saltar a (b).
(c)	00B8	SUBS $r3, r3, r1$	7331	Actualizar $pfT$ .
	00B9	SUBS $rF, r2, r3$	7F23	Comparar $dT$ con $pfT$ .
	00BA	LLI $rD, A6$	2DA6	Cargar en $rD$ dirección (a) para salto.
	00BB	BS	C200	Saltar a (a) si $pfT > dT$ .
(d)	00BC	RET	E000	Retorno al programa de llamada.

**Tabla 4.20.** Programa de ordenación por el método de las burbujas (*continuación*).

d) Codificación del programa en código máquina

La cuarta columna de la Tabla 4.20 contiene el programa codificado en código máquina (hexadecimal).

## CAMBIOS DE CÓDIGO

**P4.12.** En posiciones contiguas de la memoria de CODE se encuentra un conjunto de números enteros representados en signo y magnitud. Construir un programa en nemónicos que cambie en la tabla los números por su representación en complemento a dos. Suponer que las direcciones del primer y último número de la tabla están almacenadas en dos posiciones,  $IT = A000$  y  $FT = A001$ , respectivamente, de memoria.

### SOLUCIÓN

Para transformar un número,  $N$ , en signo y magnitud a la representación de complemento a dos hay que hacer lo siguiente:

- Si el número es positivo, no cambia.
- Si es negativo:
  - Hay que complementar todos los bits (menos el de signo); es decir, cambiarlos de ceros a unos y de unos a ceros. El bit de signo no se cambia, porque al ser en número  $N$  negativo, ya su bit de signo es 1.
  - Hay que sumar uno al resultado anterior.

Con las instrucciones de CODE-2, la forma mejor de complementar los bits de  $N$  es con la operación NAND, que se da en la tabla. Observamos que tenemos que hacer la operación NAND con el valor 1.

a	b	a NAND b
0	0	1
0	1	1
1	0	1
1	1	0

Para el bit de signo, como queremos que quede a 1, hacemos la operación NAND con 0. En definitiva, si  $N$  es negativo, tenemos que hacer la operación NAND con el patrón de bits ( $PB$ ):  $0111\ 1111\ 1111\ 1111 = EFFF$ .

De acuerdo con lo anterior, en la Tabla 4.21 se incluye la asignación de registros realizada, y en la Figura 4.21 un organigrama del programa.

Objetivo de la variable	Nombre simbólico	Registro o posición	Valor inicial
Dato a analizar	$N$	$r3$	
Puntero de la tabla	$P$	$rA$	$M(A000)$
Posición final de la tabla	$FT$	$rB$	$M(A001)$
Valor 0		$r0$	0000
Valor 1		$r1$	0001
Patrón de bits	$PB$	$r2$	FFFF

Tabla 4.21

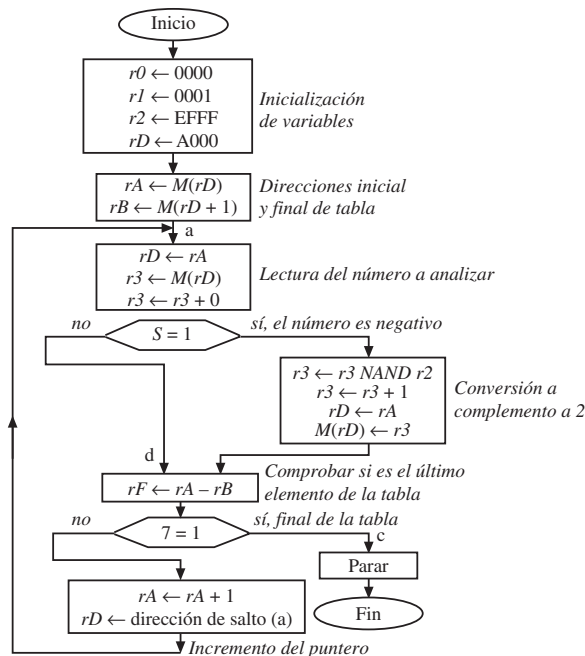


Figura 4.21. Organigrama del programa de paso a complemento a dos.

El programa en nemónicos y en código máquina se muestra en la Tabla 4.22.

## PROGRAMAS ARITMÉTICOS

**P4.13.** En el registro  $rB$  de CODE-2 se tiene un dato determinado. Indicar las instrucciones necesarias para poner a cero los bits que ocupan una posición impar dentro de dicho registro y dejar los otros como están.

### SOLUCIÓN

Según podemos ver en la Tabla 4.23 podemos hacer un bit cero, sea cual sea su valor inicial, realizando la operación lógica de intersección (AND); pero en el repertorio de instrucciones de CODE-2 sólo existe la operación lógica NAND. Ahora bien, si complementamos (invertimos) la salida de una operación NAND tenemos la intersección; en efecto se verifica:

$$[(a \cdot b)'] = a \cdot b$$

No disponemos en el repertorio de la inversión, pero si observamos la tabla de la operación NAND, si hacemos las entradas  $a$  y  $b$  iguales, a la salida tenemos el valor invertido (Tabla 4.24). Es decir, si hacemos la operación NAND de un registro consigo mismo, se realiza el complemento (cambian los ceros por unos, y los unos por ceros).

Refer.	Dirección memoria	Nemónico	Explicación
	00A0 00A1 00A2 00A3 00A4 00A5	LLI <i>r0,00</i> LLI <i>r1,01</i> LLI <i>r2,FF</i> LHI <i>r2,EF</i> LLI <i>rD,00</i> LHI <i>rD,00</i>	Inicializaciones.
	00A6 00A7	LD <i>rA,[00]</i> LD <i>rB,[01]</i>	Direcciones inicial y final de la tabla.
(a)	00A8 00A9 00AA	ADDS <i>rD,rA,r0</i> LD <i>r3,[00]</i> ADDS <i>rF,r3,r0</i>	Dirección de dato en la tabla. Carga del número a analizar. Activar biestables indicadores.
	00AB 00AC	LLI <i>rD</i> BS	Cargar en <i>rD</i> dirección de salto (b). Saltar a (b) si <i>N</i> < 0.
(d)	00AD 00AE 00AF	SUBS <i>rF,rA,rB</i> LLI <i>rD</i> BZ	Comparar dirección del puntero con la final de la tabla. Cargar en <i>rD</i> dirección de salto (c). Saltar a (c) si <i>Z</i> = 0 (fin de tabla).
	00B0 00B1 00B2	ADDS <i>rA,rA,r1</i> LLI <i>rD</i> BR	Cargar en <i>rD</i> dirección de salto (a). Salto incondicional a (a).
(b)	00B3 00B4 00B5 00B5 00B7 00B8	NAND <i>r3,r3,r2</i> ADDS <i>r3,r3,r1</i> ADDS <i>rD,rA,r0</i> ST <i>[00],r3</i> LLI <i>rD</i> BR	Complemento de <i>N</i> . Sumar 1. Llevar a <i>rD</i> el puntero. Almacenar en tabla el complemento a 2. Cargar en <i>rD</i> dirección de salto (d). Salto incondicional a (d).
(c)	00B9	HALT	

Tabla 4.22. Programa de paso a complemento a dos.

a b	AND	NAND
	$a \cdot b$	$(a \cdot b)'$
0 0	0	1
0 1	0	1
1 0	0	1
1 1	1	0

Tabla 4.23. Operaciones AND y NAND.

		Inversor
a b		$(a \cdot b)$
0 0		1
1 1		0

Tabla 4.24. Inversor con un NAND.

En consecuencia, lo que tenemos que hacer es dos operaciones NAND, la primera con una máscara que tenga todos los bits a uno, salvo los que se quieran poner a cero, y la segunda del resultado anterior consigo mismo:

LLI <i>rF,55</i> LHI <i>rF,55</i> NAND <i>rB,rB,rF</i> NAND <i>rB,rB,rB</i>	Introducimos en <i>rF</i> una máscara con ceros en las posiciones impares: 0101 0101 0101 0101. $rF \leftarrow (rB \cdot rF)'$ $rB \leftarrow (rB)'$
--	---

**P4.14.** Dada una tabla de números enteros positivos que se inicia en la posición *B000* y finaliza en la *BFFF*, hacer un programa para CODE-2 que indique por el puerto *OPI* el número de datos de la tabla que son múltiplos de 4.

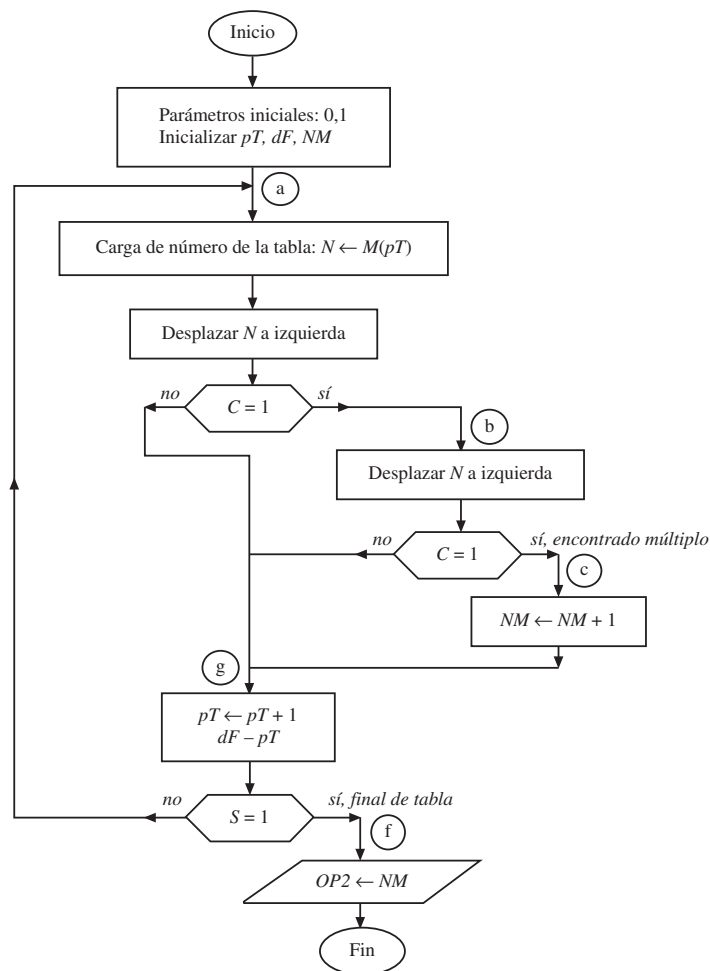
## SOLUCIÓN

*Algoritmo:* Un número binario es múltiplo de cuatro cuando acaba en dos ceros. En consecuencia el algoritmo debe recorrer uno a uno los elementos de la tabla y contabilizar los que acaban en dos ceros. Podemos determinar si el bit menos significativo es cero con un desplazamiento a la derecha (SHR) y comprobando el bit de acarreo (*C*) resultante. Las variables y parámetros utilizados se muestran en la Tabla 4.25.

Parámetro o variable	Comentario
0000	Constante cero.
0001	Para incrementar el puntero de la cadena.
<i>pT</i>	Inicializar con la dirección de inicio de la cadena <i>B000</i> .
<i>N</i>	Número en análisis.
<i>NM</i>	Número de múltiplos (inicializar a 0000).
<i>dF</i>	Dirección final, <i>BFFF</i> .

**Tabla 4.25.** Asignación de registros y de memoria del programa de los múltiplos de 4.

En la Figura 4.22 se muestra el organigrama, que se explica por sí mismo. Por otra parte, la asignación de registros y memoria se muestra en la Tabla 4.26.



**Figura 4.22.** Organigrama del programa para detección de múltiplos de 4.

Parámetro o variable	Registro	Posición de memoria	Comentario
0000	<i>r0</i>		Constante cero.
0001	<i>r1</i>		Para incrementar el puntero de la cadena.
<i>pT</i>	<i>r2</i>		Inicializar con la dirección de inicio de la cadena <i>B000</i> .
<i>N</i>	<i>r3</i>		Número de análisis.
<i>NM</i>	<i>r4</i>		Número de múltiplos (inicializar a 0000).
<i>dF</i>	<i>r5</i>		Dirección final, <i>BFFF</i> .
Programa		0000	Dirección de carga del programa.

**Tabla 4.26.** Asignación de registros y de memoria del programa de los múltiplos de 4.

El programa en némonicos y en lenguaje máquina puede verse en la Tabla 4.27.

Refer. salto	Dirección memoria	Instrucción máquina		Explicación
		Nemónico	Hex	
	0000	LLI <i>r0,00</i>	2000	Inicializar <i>r0</i> a 0000.
	0001	LLI <i>r1,01</i>	2101	Inicializar <i>r1</i> a 0001.
	0002	LLI <i>r2,3C</i>	2D00	Dirección de inicio de tabla de números: <i>B000</i> .
	0003	LHI <i>r2,5F</i>	32B0	
	0004	LLI <i>r6,00</i>	25FF	Dirección final de la tabla: <i>BFFF</i> .
	0005	LHI <i>r6,F0</i>	36BF	
	0006	LLI <i>r4,00</i>	2400	<i>NM</i> ← 0000.
(a)	0007	ADDS <i>rD,r2,r0</i>	6D20	<i>rD</i> ← <i>pT</i> .
	0008	LD <i>r3,[00]</i>	0300	Cargar número de la tabla en <i>r3</i> .
	0009	SHR <i>r3</i>	A300	Desplazar a derecha el número.
	000A	LLI <i>rD,12</i>	2D12	Dirección de salto (b).
	000B	BC	C300	Salta a (b) si acaba en cero.
(g)	000C	ADDS <i>r2,r2,r1</i>	6221	<i>pT</i> ← <i>pT</i> + 1.
	000D	SUBS <i>rF, r5, r2</i>	7F52	<i>dF</i> ← <i>pT</i> .
	000E	LLI <i>rD,1A</i>	2D1A	Dirección de salto (f).
	000F	BS	C200	Saltar a (f) si se llega a final de tabla.
	0010	LLI <i>rD,07</i>	2D07	Dirección de salto (a).
	0011	BR	C000	Salto incondicional a (a).
(b)	0012	SHR <i>r3</i>	A300	Desplazar a derecha el número.
	0013	LLI <i>rD,17</i>	2D17	Dirección de salto (c).
	0014	BC	C300	Salta a (c) si acaba en cero.
	0015	LLI <i>rD,0C</i>	2D0C	Dirección de salto (g).
	0016	BR	C000	Salto incondicional a (g).
(c)	0017	ADDS <i>r4,r4,r1</i>	6441	<i>NM</i> ← <i>NM</i> + 1.
	0018	LLI <i>rD,0C</i>	2D0C	Dirección de salto (g).
	0019	BR	C000	Salto incondicional a (g).
(f)	001A	OUT <i>OP2, r4</i>	5402	Salida de máximo.
	001B	HALT	FFFF	Fin.

**Tabla 4.27.** Programa detección de múltiplos de 4.

**P4.15.** Hacer una subrutina que genere a partir de la posición E000 de memoria los cuadrados de los números enteros del 0 al 10.

#### SOLUCIÓN

*Algoritmo:* Como no disponemos de instrucción de multiplicar, podemos hacer el cuadrado de un número *n* sumando *n*, *n* veces. En consecuencia, hacemos un lazo de 2 a 10, en que se realicen las citadas

sumas. En la Tabla 4.28 se indican las variables y parámetros que utilizaremos, y en la Figura 4.23 un organigrama, que se explica por sí mismo.

Parámetro o variable	Comentario
0000	Constante cero.
0001	Para incrementar el puntero de la cadena.
<i>d</i>	Dirección para almacenar el cuadrado; inicializar con <i>E000</i> .
<i>N</i>	Número al que hay que realizar el cuadrado.
<i>NT</i>	Número (registro) temporal.
<i>NC</i>	Cuadrado del número.
<i>NF</i>	Último número a realizar el cuadrado; inicializar a 10.

Tabla 4.28. Variables y parámetros del programa de obtención de los cuadrados.

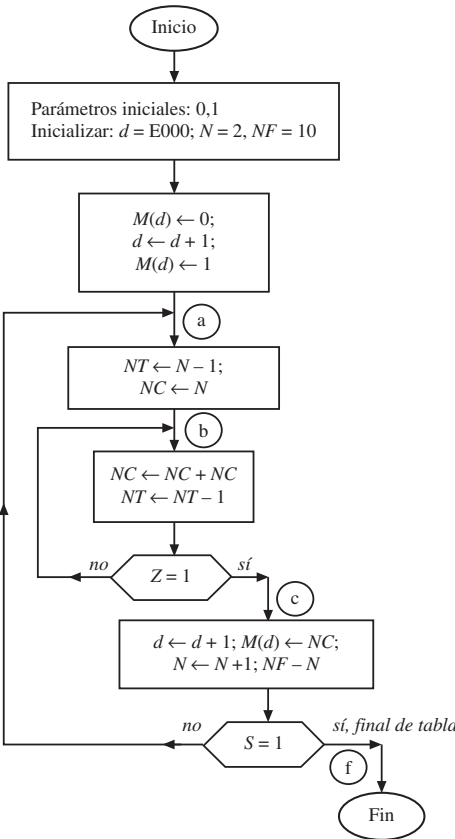


Figura 4.23. Organigrama del programa de obtención de cuadrados.

La asignación de variables a registros se incluye en la Tabla 4.29, y el programa en nemónicos y en código máquina en la Tabla 4.30.

**P4.16.** En posiciones consecutivas de la memoria de CODE-2, a partir de la *H'1000*, se encuentran una serie de registros que contienen los datos personales de 8 alumnos de la asignatura Introducción a los Computadores. Cada uno de estos registros está compuesto por los siguientes campos:

- a) Nombre y apellidos: 30 caracteres Unicode.
- b) Nota de teoría: 1 entero.
- c) Nota de prácticas: 1 entero.
- d) Nota final: 1 entero.

Parámetro o variable	Registro	Posición de memoria	Comentario
0000	<i>r0</i>		Constante cero.
0001	<i>r1</i>		Para incrementar el puntero de la cadena.
<i>d</i>	<i>r2</i>		Dirección para almacenar el cuadrado; inicializar con <i>E000</i> .
<i>N</i>	<i>r3</i>		Número al que hay que realizar el cuadrado, inicializar a <i>0002</i> .
<i>NT</i>	<i>r4</i>		Número (registro) temporal.
<i>NC</i>	<i>r5</i>		Cuadrado del número.
<i>NF</i>	<i>r6</i>		Último número a realizar el cuadrado; inicializar a 10.
Programa		0000	Dirección de carga del programa.

**Tabla 4.29.** Asignación de registros y de memoria del programa de obtención de los cuadrados.

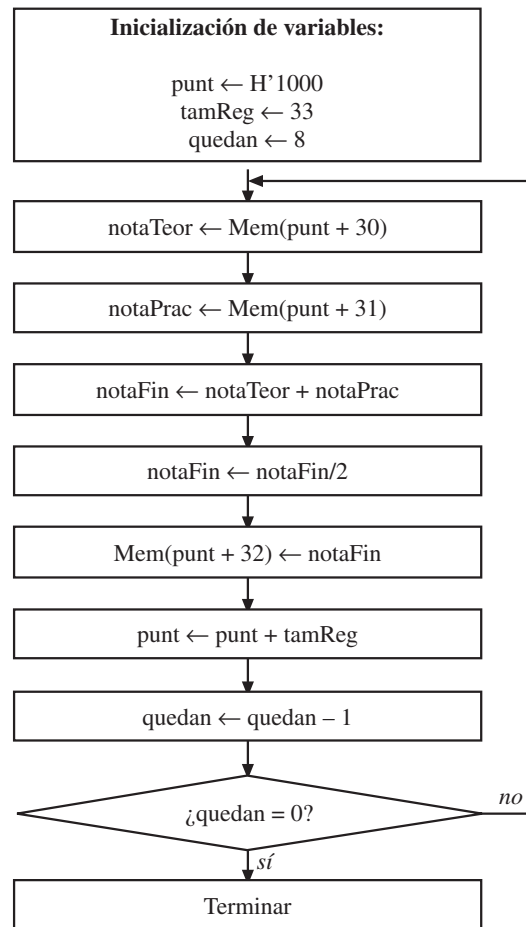
Refer. salto	Dirección memoria	Instrucción máquina		Explicación
		Nemónico	Hex	
	0000	LLI <i>r0,00</i>	2000	Inicializar <i>r0</i> a 0000.
	0001	LLI <i>r1,01</i>	2101	Inicializar <i>r1</i> a 0001.
	0002	LLI <i>r2,00</i>	2D00	Dirección de inicio de tabla de números: <i>E000</i> .
	0003	LHI <i>r2,E0</i>	32B0	
	0004	LLI <i>r6,0A</i>	25FF	$NF \leftarrow D'10$ .
	0005	LLI <i>r3,02</i>	36BF	$N \leftarrow 2$ .
	0006	ADDS <i>rD,r2,r0</i>	6D20	$rD \leftarrow r2$ .
	0007	ST [00], <i>r0</i>	1000	$M(d) \leftarrow 0000$ .
	0008	ADDS <i>rD,rD,r1</i>	6DD1	$d \leftarrow d + 1$ .
	0009	ST [00], <i>r1</i>	0100	$M(d) \leftarrow 0001$ .
	000A	ADDS <i>rD,rD,r1</i>	6DD1	$d \leftarrow d + 1$ .
(a)	000B	ADDS <i>r4,r3,r0</i>	6430	$NT \leftarrow N$ .
	000C	SUBS <i>r4,r4,r1</i>	7441	$NT \leftarrow N - 1$ .
	000D	ADDS <i>r5,r3,r0</i>	6530	$NC \leftarrow N$ .
(b)	000E	ADDS <i>r5,r5,r5</i>	6555	$NC \leftarrow NC + NC$ .
	000F	SUBS <i>r4,r4,r1</i>	7441	$NT \leftarrow NT - 1$ .
	0010	LLI <i>rD,14</i>	2D14	Dirección de salto (c).
	0011	BZ	C100	Saltar a (c) si $NT = 0$ .
	0012	LLI <i>rD,0E</i>	2D0E	Dirección de salto (b).
	0013	BR	C000	Salto incondicional a (b).
(c)	0014	ADDS <i>r2,r2,r1</i>	6221	$d \leftarrow d + 1$ .
	0015	ADDS <i>rD,r2,r0</i>	6D20	$rD \leftarrow d$ .
	0016	ST [00], <i>r5</i>	1500	$M(d) \leftarrow NC$ .
	0017	ADDS <i>r3,r3,r1</i>	6331	$N \leftarrow N + 1$ .
	0018	SUBS <i>rF,r6,r3</i>	7F63	$NF \leftarrow N$ .
	0019	LLI <i>rD,1D</i>	2D1D	Dirección de salto (f).
	001A	BS	C200	Saltar a (f) si $N > NF$ (si se ha acabado la tabla).
	001B	LLI <i>rD,0B</i>	2D0B	Dirección de salto (a).
	001C	BR	C000	Salto incondicional a (a).
(f)	001D	HALT	FFFF	Fin.

**Tabla 4.30.** Programa de obtención de una tabla con los cuadrados de números del 0 al 10.

Escribir un programa que comience a partir de la dirección de memoria H'0100, que rellene el campo nota final de cada alumno con el valor resultante de calcular la media aritmética de sus notas de teoría y prácticas.

#### SOLUCIÓN

En la Figura 4.24 se muestra un organigrama del programa.



**Figura 4.24.** Organigrama del programa de nota media de alumnos.

Para poder realizar el programa necesitaremos tener almacenados en un par de registros los valores 0 y 1, de forma que se puedan copiar valores entre registros e incrementar el puntero que va recorriendo los datos. Para este uso reservaremos los registros *r0* y *r1*, respectivamente.

También necesitamos almacenar el desplazamiento que hay que aplicar para saltarse el nombre y los apellidos. Para ello usaremos el registro *r2*.

El registro *r3* se usará para almacenar el número de alumnos que quedan por procesar; para recorrer la tabla de registros se usará el registro *r4* y para almacenar temporalmente las notas se usarán los registros *r5*, *r6* y *r7*.

Por tanto, inicialmente se supone la siguiente asignación de registros:

Registro	Variable	Valor inicial
<i>r0</i>		0
<i>r1</i>		1
<i>r2</i>	tamReg	30
<i>r3</i>	quedan	8
<i>r4</i>	punt	H' 1000
<i>r5</i>	notaTeor	
<i>r6</i>	notaPrac	
<i>r7</i>	notaFin	



El programa en nemónicos y en código máquina se muestra en la Tabla 4.31.

Dirección	Nemónico	Código	Comentario
0100	ADDS $rD, r4, r0$	6D40	$rd \leftarrow \text{punt.}$
0101	LD $r5, rD[1E]$	051E	$\text{notaTeor} \leftarrow \text{Mem}(\text{punt} + 30).$
0102	LD $r6, rD[1F]$	061F	$\text{notaPrac} \leftarrow \text{Mem}(\text{punt} + 31).$
0103	ADDS $r7, r5, r6$	6756	$\text{notaFin} \leftarrow \text{notaTeor} + \text{notaPrac}.$
0104	SHRA $r7$	B700	$\text{notaFin} \leftarrow \text{notaFin}/2.$
0105	ST $rD[20], r7$	1720	$\text{Mem}(\text{punt} + 32) \leftarrow \text{notaFin}.$
0106	LLI $rD, 0E$	2D0E	Preparo la dirección de salto si el programa debe terminar.
0107	LHI $rD, 01$	3D01	programa debe terminar.
0108	ADDS $r4, r4, r2$	6442	$\text{punt} \leftarrow \text{punt} + \text{tamReg}.$
0109	SUBS $r3, r3, r1$	7331	$\text{quedan} \leftarrow \text{quedan} - 1.$
010A	BZ	C100	Si no quedan, salto al final.
010B	LLI $rD, 00$	2D00	Preparo la dirección de salto para la siguiente iteración.
010C	LHI $rD, 01$	3D01	guiente iteración.
010D	BR	C000	A por el siguiente alumno.
010E	HALT	F000	Termina el programa.

**Tabla 4.31.** Programa de nota media de alumnos.

**P4.17.** Se desea realizar un programa en lenguaje máquina de CODE-2 que, teniendo en los registros  $r2$  y  $r3$  dos números positivos, realice la multiplicación de ambos números, almacene el resultado en la posición de memoria contenida en  $r4$  y lo visualice en  $OPI$ . Suponer que el programa se cargará a partir de la dirección  $H'0000$ .

- Realizar un organigrama del programa.
- Efectuar la asignación de registros y de memoria.
- Redactar el programa en nemónicos y código máquina.
- Calcular el tiempo que tardaría en ejecutarse el programa si  $R2 = 5$  y  $R3 = 2$  si la frecuencia de reloj de CODE-2 fuese de 10 GHz.

*Nota:* El programa debe prever que uno de los datos a multiplicar sea 0, y que se produzca un desbordamiento, en este último caso, el resultado que debe proporcionarse por el puerto OP2 será  $0F0F$ .

#### SOLUCIÓN

##### a) Organigrama del programa

La multiplicación la realizamos sumando el multiplicando tantas veces como indique el multiplicador. Al hacer cada suma parcial comprobamos si se produce desbordamiento ( $V = 1$ ). El organigrama se muestra en la Figura 4.25.

##### b) Asignación de registros y de memoria

- El programa se carga a partir de la dirección  $M(0000)$ .
- En  $M(R4)$  se deja el resultado.

$R0$	Valor 0	0000
$R1$	Valor uno	0001
$R2$	Multiplicando	
$R3$	Multiplicador	
$R4$	Dirección resultado	
$R5$	Patrón <i>overflow</i>	0F0F
$R6$	Resultado multiplicación	0000

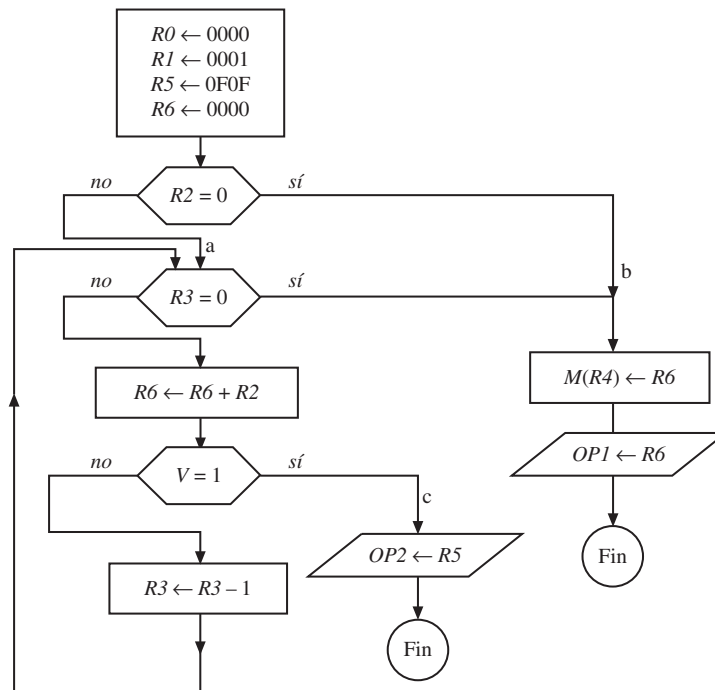


Figura 4.25. Organigrama del programa de multiplicar.

c) Programa en nemónicos y código máquina. Puede verse en la Tabla 4.32.

Salto	Dirección	Nemónico	Hexadecimal	Comentarios
	0000 0001 0002 0003 0004	LLI R0,00 LLI R1,01 LLI R5,0F LHI R5,0F LLI R6,00	2000 2101 250F 350F 2600	Inicializaciones
	0005 0006 0007	SUBS RF,R2,R0 LLI RD,11 BZ	7F20 2D11 C100	¿R2 = 0? Salto a (b)
(a)	0008 0009 000A	SUBS RF,R3,R0 LLI RD,11 BZ	7F30 2D11 C100	¿R3 = 0? Salto a (b)
	000B 000C 000D	ADDS R6,R6,R2 LLI RD,14 BV	6662 2D14 C400	Paso de multiplicación Salto a (c) si overflow
	000E 000F 0010	SUBS R3,R3,R1 LLI RD,08 BR	7331 2D08 C000	Salto a (a)
(b)	0011 0012 0013	ST [0 + R4],R6 OUT OP1,R6 HALT	1600 5601 F000	Memorizar resultado Sacar el resultado Fin
(c)	0014 0015	OUT OP2,R5 HALT	5502 F000	Salida de overflow Fin

Tabla 4.32. Programa de multiplicar enteros positivos.

- d) Tiempo que tardaría en ejecutarse el programa si  $r2 = 5$  y  $r3 = 2$  si la frecuencia de reloj de CODE-2 fuese de 10 GHz

Suponiendo que  $R2 = 5$  y  $R3 = 2$ , el programa ejecutará las siguientes instrucciones:

Ciclos	Instrucción	Número de veces	Total ciclos
6	LLI	5	30
7	SUBS	1	7
6	LLI	1	6
6	BZ	1	6
7	SUBS	$R2 = 2, 1, 0$ (3 veces)	21
9	LLI	$R2 = 2, 1, 0$ (3 veces)	27
6	BZ	$R2 = 2, 1, 0$ (3 veces)	18
7	ADDS	$R2 = 2, 1$ (2 veces)	14
6	LLI	$R2 = 2, 1$ (2 veces)	12
6	BV	$R2 = 2, 1$ (2 veces)	12
7	SUBS	$R2 = 1, 0$ (2 veces)	14
6	LLI	$R2 = 1, 0$ (2 veces)	12
6	BR	$R2 = 1, 0$ (2 veces)	12
9	ST	1	9
8	OUT	1	8
6	HALT	1	6
TOTAL			214

El período de reloj (tiempo de ciclo) es:

$$T = \frac{1}{F} = \frac{1}{10 \times 10^9} = 100 \mu s$$

con lo que el tiempo de ejecución del programa es:

$$t = T \cdot N = 100 \mu s \times 214 = 21 \mu s$$

## PROGRAMAS DE ESTIMACIONES DE TIEMPO

- P4.18.** Realizar una rutina en nemónicos, para ubicar a partir de la dirección 00E0, que produzca un retardo de 1 ms, suponiendo una implementación de CODE-2 cuya frecuencia de reloj sea 1 GHz.

### SOLUCIÓN

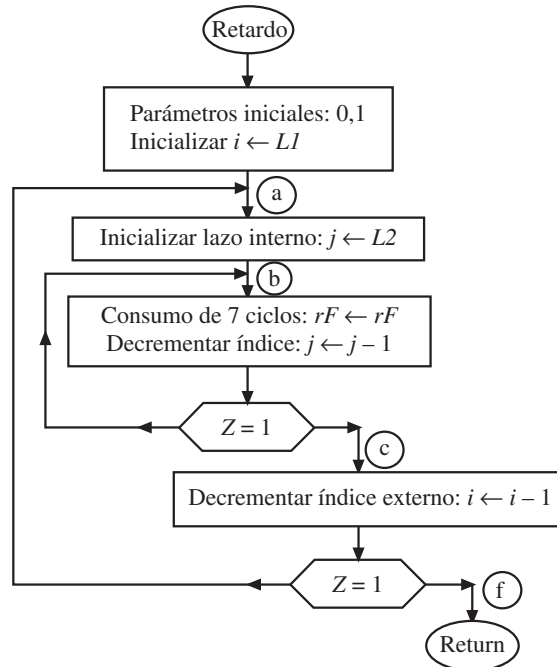
Si la frecuencia de CODE-2 es de  $F = 1$  GHz, el período de reloj (o tiempo de ciclo) será:

$$T = \frac{1}{F} = \frac{1}{10^9} = 10^{-9} \text{ seg}$$

Con lo que para consumir 1 ms necesitamos el siguiente número de ciclos:

$$N = \frac{10^{-3} \text{ seg}}{10^{-9} \text{ seg/ciclo}} = 10^6 \text{ ciclos}$$

La idea consiste en hacer un ciclo, y calcular el número de iteraciones a realizar de forma que la rutina consuma  $10^6$  ciclos. Como el número positivo mayor de que disponemos es 32.767, y tendríamos que hacer del orden de 100.000 iteraciones es mejor utilizar dos ciclos, uno interno y otro externo, de esta forma el número de iteraciones será el producto de los dos. La Figura 4.26 muestra un organigrama de la rutina. Hemos denominado  $i$  y  $L1$  al índice y valor máximo del lazo exterior; y  $j$  y  $L2$ , a los del lazo interior. Los valores de  $L1$  y  $L2$  los calcularemos con la condición de que se produzcan un millón de ciclos en total.



**Figura 4.26.** Organigrama de la rutina que provoca un retardo de 1 ms.

La asignación de parámetros constantes y variables a registros es la que se muestra en la Tabla 4.33.

Parámetro o variable	Registro	Posición de memoria	Comentario
0000	r0		Constante cero.
0001	r1		Para incrementar el puntero de la cadena.
i	r2		Índice del lazo exterior; inicializar a L1.
j	r3		Índice del lazo interior.
L2	r4		Valor máximo del índice del lazo interior.
Programa		00E0	Dirección de carga del programa.

**Tabla 4.33.** Asignación de registros y de memoria de la subrutina de retardo.

El programa en nemónicos y en lenguaje máquina es el que se muestra en la Tabla 4.34. Para obtener los valores de L1 y L2, hemos incluido después de los nemónicos de cada instrucción el número de ciclos que consume cada una de ellas (ver Tabla 4.4). Sumando todos los tiempos se tiene que el número de ciclos consumido por la subrutina es:

$$N_{\text{retardo}} = 48 + 44 \cdot L1 + 38 \cdot L2 \cdot L1$$

Haciendo  $L1 = 1.000$  iteraciones, se tiene que:

$$N_{\text{retardo}} = 48 + 44.000 + 38.000 \cdot L2$$

De donde, despejando L2, se obtiene que:

$$L2 = 25, 156632 \text{ iteraciones}$$

En consecuencia hacemos  $L2 = 25$  iteraciones. Obsérvese que se obtiene un error de:

$$t_{\text{error}} = 0,156632 \cdot 38.000 \cdot 10^{-9} = 0,006 \text{ ms}$$

Este error podría reducirse eligiendo más adecuadamente los valores de L1 y L2.

En consecuencia, hacemos  $L1 = D'1000 = H'03C8$ ; y  $L2 = D'25 = H'0019$ .

Refer. salto	Dirección memoria	Instrucción máquina		Explicación
		Nemónico	Ciclos	
	0000	LLI $r0,00$	6	Inicializar $r0$ a 0000.
	0001	LLI $r1,01$	6	Inicializar $r1$ a 0001.
	0002	LLI $r2,C8$	6	Inicializar lazo exterior, $L1 = 03C8$ .
	0003	LHI $r2,03$	8	
	0004	LLI $r4,19$	6	Valor máximo del índice del lazo interior: $L2 = 0019$ .
	0005	LHI $r4,00$	8	
(a)	0006	ADDS $r3,r4,r0$	$7 \cdot L1$	Inicio lazo exterior; $j \leftarrow L2$ .
(b)	0007	ADDS $rF,rF,r0$	$7 \cdot L2 \cdot L1$	Consumo de 7 ciclos.
	0008	SUBS $r4,r4,r1$	$7 \cdot L2 \cdot L1$	$j \leftarrow j - 1$ .
	0009	LLI $rD$	$6 \cdot L2 \cdot L1$	Dirección de salto (c).
	000A	BZ	$6 \cdot L2 \cdot L1$	Saltar a (c) si $j = 0$ .
	000B	LLI $rD$	$6 \cdot L2 \cdot L1$	Dirección de salto (b).
	000C	BR	$6 \cdot L2 \cdot L1$	Salto incondicional a (b).
(c)	000D	SUBS $r2,r2,r1$	$7 \cdot L1$	$i \leftarrow i - 1$ .
	000E	LLI $rD$	$6 \cdot L1$	Dirección de salto (f).
	000F	BZ	$6 \cdot L1$	Saltar a (f) si $i = 0$ .
	0010	LLI $rD$	$6 \cdot L1$	Dirección de salto (a).
	0011	BR	$6 \cdot L1$	Salto incondicional a (a).
(f)	0012	RET	8	

**Tabla 4.34.** Programa de obtención de una tabla con los cuadrados de números del 0 al 10.

## COMPILACIÓN DE INSTRUCCIONES DE ALTO NIVEL

**P4.19.** Se tiene la siguiente instrucción *switch* de lenguaje C:

```
switch (k) {
    case 0: g = j + n; break;
    case 1: g = h + i; break;
    case 2: g = h - i; break;
    case 3: g = j - n; break;
};
```

Hacer una rutina en nemónicos de CODE-2 que implemente la instrucción anterior, suponiendo que los variables  $g$ ,  $h$ ,  $i$ ,  $j$ ,  $k$  y  $n$  tienen asignadas las posiciones de memoria  $A000$ ,  $A001$ ,  $A002$ ,  $A003$  y  $A004$ ,  $A0005$ , respectivamente.

### SOLUCIÓN

Dada la sencillez del problema vamos a realizar directamente el organigrama. Hacemos la asignación de variables y parámetros que se indica en la Tabla 4.35.

Parámetro o variable	Registro
0000	$r0$
0001	$r1$
0002	$r2$
0003	$r3$
$g$	$r4$
$h$	$r5$
$i$	$r6$
$j$	$r7$
$k$	$r8$
$n$	$r9$

**Tabla 4.35.** Asignación de registros al programa de implementación de una instrucción *switch*.

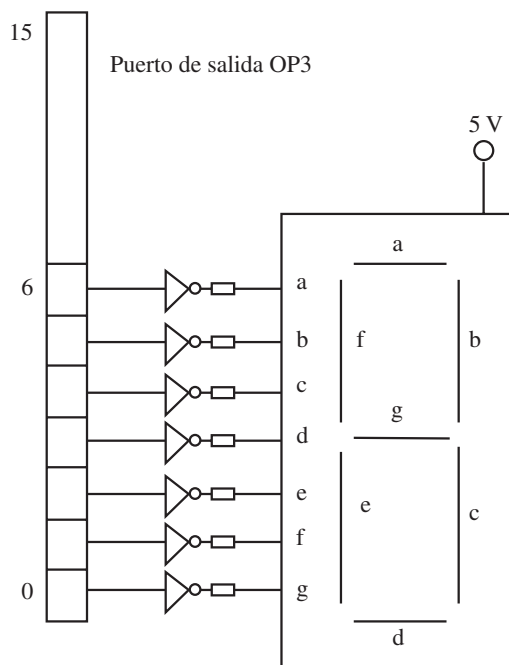
El programa correspondiente se muestra en la Tabla 4.36.

Salto	Dirección memoria	Nemónico	Explicación
	0000	LLI $r0,00$	$r0 \leftarrow 0000$ .
	0001	LLI $r1,01$	$r1 \leftarrow 0001$ .
	0002	LLI $r2,02$	$r2 \leftarrow 0002$ .
	0003	LLI $r3,03$	$r3 \leftarrow 0003$ .
	0004	LLI $rD,01$	
	0005	LHI $rD,A0$	$rD \leftarrow A001$ (dirección de h).
	0006	LD $r5,[00]$	$r5 \leftarrow h$ .
	0007	ADDS $rD,rD,r1$	$rD \leftarrow rD + 1$ .
	0008	LD $r6,[00]$	$r6 \leftarrow i$ .
	0009	ADDS $rD,rD,r1$	$rD \leftarrow rD + 1$ .
	000A	LD $r7,[00]$	$r7 \leftarrow j$ .
	000B	ADDS $rD,rD,r1$	$rD \leftarrow rD + 1$ .
	000C	LD $r8,[00]$	$r8 \leftarrow k$ .
	000D	ADDS $rD,rD,r1$	$rD \leftarrow rD + 1$ .
	000E	LD $r9,[00]$	$r9 \leftarrow m$ .
	000F	SUBS $rF,r8,r0$	$\zeta k = 0?$
	0010	LLI $rD$	Dirección c0.
	0011	BZ	Saltar a case 0.
	0012	SUBS $rF,r8,r1$	$\zeta k = 1?$
	0013	LLI $rD$	Dirección c1.
	0014	BZ	Saltar a case 1.
	0015	SUBS $rF,r8,r2$	$\zeta k = 2?$
	0016	LLI $rD$	Dirección c2.
	0017	BZ	Saltar a case 2.
	0018	SUBS $rF,r8,r3$	$\zeta k = 3?$
	0019	LLI $rD$	Dirección c3.
	001A	BZ	Saltar a case 3.
	001B	LLI $rA,EE$	Error, se da por el puerto O01 el mensaje EEEE.
	001C	LHI $rA,EE$	
	001D	OUT $O01,rA$	
	001E	HALT	
c0	001F	ADDS $r4,r7,r9$	Caso 0.
	0020	LLI $rD$	Dirección de salto <i>continuar</i> .
	0021	BR	Salto incondicional a cont.
c1	0022	ADDS $r4,r5,r6$	Caso 1.
	0023	LLI $rD$	Dirección de salto <i>continuar</i> .
	0024	BR	Salto incondicional a cont.
c2	0025	SUBS $r4,r5,r6$	Caso 2.
	0026	LLI $rD$	Dirección de salto <i>continuar</i> .
	0027	BR	Salto incondicional a cont.
c3	0028	SUBS $r4,r7,r9$	Caso 3.
<i>continuar</i>	0029	LLI $rD,00$	Dirección de memoria donde hay que almacenar g. Almacenar en memoria el valor de g. <i>CONTINUAR</i>
	002A	LHI $rD,A0$	
	002B	ST $[00],r4$	
	002C		

**Tabla 4.36.** Programa de implementación de una instrucción *switch*.

## PROGRAMAS DE ENTRADA/SALIDA

**P4.20.** Suponiendo que los siete bits menos significativos del puerto de salida *OP3* de CODE-2 se encuentran conectados a un indicador de siete segmentos (*LED*) de ánodo común, tal como se indica en la Figura 4.27, hacer una subrutina que esté ubicada a partir de la posición *E000*, que muestre por el indicador los valores hexadecimales correspondientes a los 4 bits menos significativos del dato contenido en el registro *rB*. *Información complementaria y sugerencias:* El indicador de 7 segmentos (Figura 4.27) funciona de tal manera que si la entrada *a* es 0 (o, lo que es lo mismo, si  $OP3(6) = 1$ ), el segmento *a* se activa, emitiendo luz. De esta forma se sugiere utilizar una **tabla de consulta** (*Lookup table*) en memoria (a partir de la posición *EB00*) de forma que en la posición 0 de la tabla los bits 6:0 contengan el patrón de valores a aplicar a los indicadores para que se activen los elementos que hagan que se visualice un 0 (elementos *a*, *b*, *c*, *d*, *e*, *f*), en la posición 1 los correspondientes a la visualización de un 1 (elementos *b* y *c*), y así sucesivamente. (*Nota:* En el problema P3.9 se muestra cómo realizar la misma función por hardware.)



**Figura 4.27.** Indicador LED de 7 segmentos conectado al puerto *OP3* de CODE-2.

## SOLUCIÓN

Suponemos que a partir de la posición *BAS = EB00* de memoria tenemos una tabla de consulta, que contiene directamente los valores que hacen que cada segmento del indicador emita o no luz (si el bit es 1 se emite luz). La tabla sería:

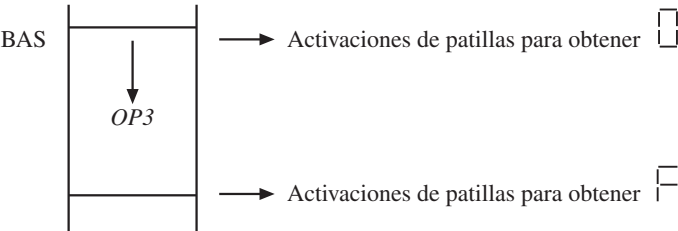
Tabla de consulta

<i>dirección</i>	A	b	c	d	e	f	g	
<i>BAS = EB00</i>	----	1	1	1	1	1	0	→ 0
<i>EB01</i>	----	0	1	1	0	0	0	→ 1
<i>EB02</i>	----	1	1	0	1	1	0	→ 2
.		.						
.		.						
.		.						
<i>EB0F</i>	----	1	0	0	0	1	1	→ F

"-" → valor indiferente

Como tenemos que visualizar el valor correspondiente a los 4 últimos bits del valor en *rB*, primero conviene extraer esos cuatro últimos bits, y los llevaremos a *rC*. De esta forma si en *rB* hay, por ejemplo, *F7C5*, en *rC* deseamos obtener 0005.

Una vez que tenemos en *rC* los 4 últimos bits de *rB*, los sumamos a *EB00* (posición inicial de la tabla de consulta) y obtendremos la dirección de la tabla donde se encuentra el valor que llevado al puerto *OP3* hace que se visualice la cifra hexadecimal correspondiente. En efecto, al realizar la suma: *0005 + EB00*, obtenemos *EB05*, que contiene el valor que activa los segmentos correspondientes al 5 (segmentos *a*, *f*, *g*, *c*, *d*).



Para extraer los 4 últimos bits de *rB* (es decir, convertir XXXH en 000H), podemos realizar dos operaciones NAND, con los patrones 000F y FFFF, según se indica a continuación:

<i>rB</i>	x x x x . . . . . x h h h h	XXXH	
<i>rF</i>	0 0 0 0 . . . . . 0 1 1 1 1	000F	NAND
<i>rC</i>	1 1 1 1 . . . . . 1 h'h'h'h'	FFFF	
<i>rF</i>	1 1 1 1 . . . . . 1 1 1 1 1		NAND
<i>rC</i>	0 0 0 0 . . . . . 0 h h h h	000H	

También se podría hacer con 12 desplazamientos a la izquierda y 12 a la derecha, pero es más complicado.

El organigrama de todo el proceso es el que se indica en la Figura 4.28 y Tabla 4.37.

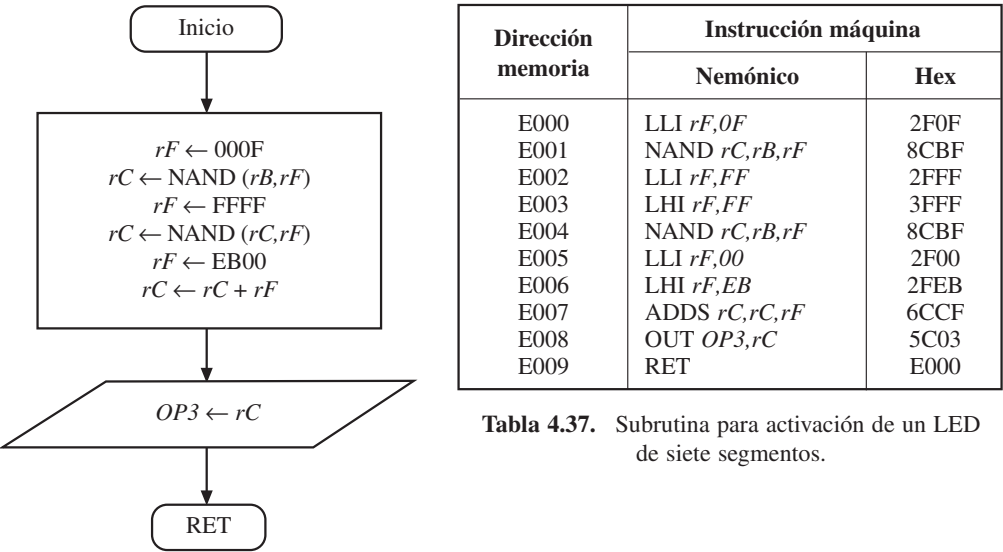
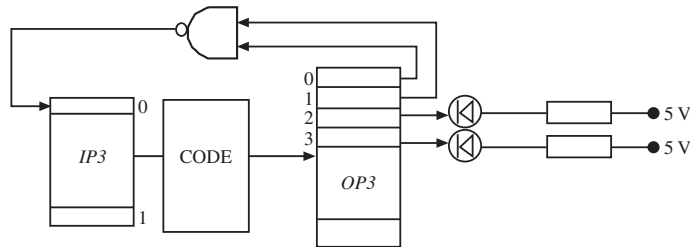


Tabla 4.37. Subrutina para activación de un LED de siete segmentos.

Figura 4.28. Organigrama del programa de activación de un LED de siete segmentos.



- P4.21.** En los puertos de salida *OP3* y de entrada *IP3* de CODE se conectan los circuitos que se indican en la Figura 4.29. Se trata de comprobar el correcto funcionamiento de puertas NAND. Uno de los fotodiodos (el conectado al bit 3 de *OP3*) es verde y debe iluminarse si el circuito funciona correctamente, el otro (bit 2) es rojo y debe iluminarse si la puerta NAND no funciona correctamente. Hacer un organigrama para CODE-2 de test (éste debe probar el adecuado funcionamiento de la puerta NAND para todas las posibles combinaciones de entrada).



**Figura 4.29.** Montaje para test automático del comportamiento de un circuito NAND.

### SOLUCIÓN

De acuerdo con el esquema del circuito, para que se encienda cada uno de los LED el bit del puerto de salida al que está conectado debe ser 0; si es 1 no se activa.

*Algoritmo:* Damos en la salida sucesivamente en los bits *OP3(0:1)* todas las combinaciones de entrada que se indican en la Tabla 4.38, y después de cada una de ellas comprobamos si el bit *IP3(0)* coincide con la salida correcta; si es así para todas las combinaciones, llevamos a la salida el patrón:

$$bien \leftarrow 0000\ 0000\ 0000\ 0100 = 0004,$$

con lo que el diodo verde se encenderá y el rojo permanecerá apagado. Caso de que se detecte un error se dará por la salida el patrón:

$$mal \leftarrow 0000\ 0000\ 0000\ 1000 = 0008,$$

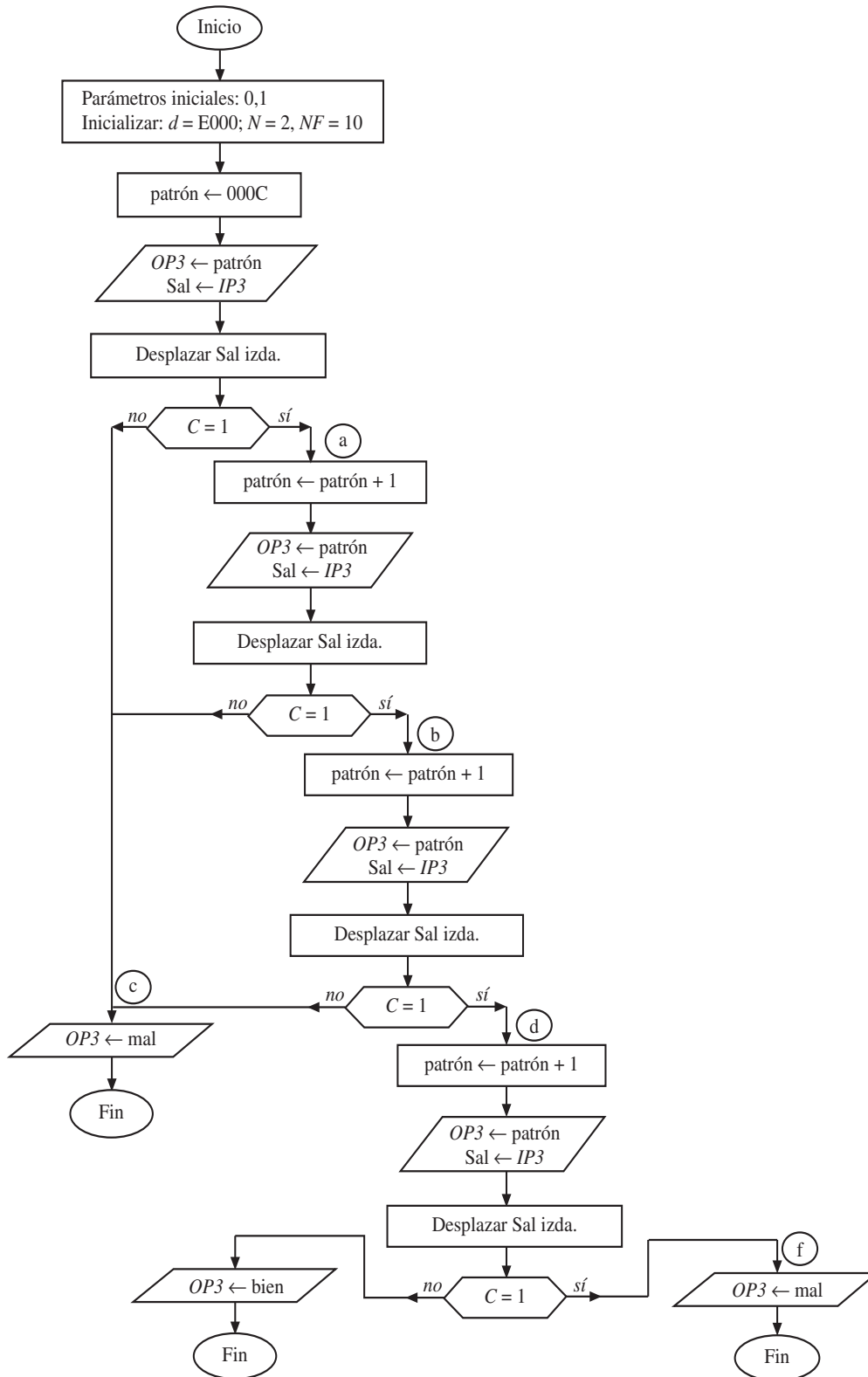
con lo que se encenderá el LED rojo.

a b	NAND	Patrones de prueba (a dar por <i>OP3</i> )	
	$(a \cdot b)'$	Binario	Hex
0 0	1	0000 0000 0000 1100	000C
0 1	1	0000 0000 0000 1101	000D
1 0	1	0000 0000 0000 1110	000E
1 1	0	0000 0000 0000 1111	000F

**Tabla 4.38.** Patrones de prueba a generar para probar una puerta NAND.

Los patrones a generar y dar por la salida *OP3* son los que se indican en la Tabla 4.38. Obsérvese que siempre mantenemos los bits 2 y 3 al valor 1, para que no se activen los LED. Los valores de entrada al circuito NAND se dan en los bits de orden 0 y 1 del patrón.

En la Figura 4.30 se muestra un organigrama del algoritmo que resuelve el problema.



**Figura 4.30.** Organigrama de programa para test automático de una puerta NAND.



## Problemas propuestos

### ELEMENTOS DE UN PROCESADOR

**P4.22.** Un procesador dispone, entre otros, de los siguientes elementos: registro de dirección (*AR*) de 32 bits, registro de datos (*DR*) de 16 bits, contador de programa (*PC*) y puntero de pila (*SP*). Indicar:

1. Número de bits del bus de datos.
2. Número de bits del bus de direcciones.
3. Tamaño máximo posible de la memoria principal (en MB o GB).
4. Tamaño en bits del registro *PC*.
5. Tamaño en bits del registro *SP*.

### EJECUCIÓN DE INSTRUCCIONES

**P4.23.** Suponga que CODE-2 dispusiese de una instrucción *memorizar*, **ST r1**, que almacena en la posición *rD* de memoria el contenido del registro *r1*. La instrucción tiene de código (en hexadecimal) 1100. Suponiendo que esta instrucción se encuentra en la posición A777 de la memoria, que en *rD* se encuentra el valor 5ACD y *r1* contiene FFFF, realice una tabla donde se indiquen las distintas microoperaciones que deben generarse durante la ejecución de la instrucción, los valores que tienen en cada momento los registros *PC*, *AR*, *DR* e *IR*, y los cambios producidos en la memoria.

**P4.24.** Suponga la instrucción *entrada* (**IN r0,IP**) de CODE-2 que almacena en el registro *r0* el contenido del puerto de entrada *IP3*. El código (en hexadecimal) de la instrucción es 4003. Suponiendo que en un momento esta instrucción se encuentra en la posición de memoria A778 y que en el puerto *IP3* se encuentra el valor FFFF, realice una tabla donde se indiquen las distintas microoperaciones que deben generarse durante la ejecución de la instrucción, los valores que tienen en cada momento los registros *PC*, *AR*, *DR* e *IR*, y los cambios producidos en el banco de registros.

μoperación	PC	DM	RM	IR	R0	SP	M(FFAA)	M(FFAB)	M(FFA9)
Valores iniciales		A732	3459	7321	3486	FFAA	0079	0054	00A3

- b) ¿Qué restricción impone el formato de la instrucción indicada, en cuanto a las subrutinas que pueden ser llamadas?

**P4.28.** Se dispone de una memoria de 4 GBytes, organizada en palabras de 16 bits. Obtener:

- a) La longitud de los buses de datos y de dirección de la memoria.
- b) Suponiendo que un proceso utiliza una pila para poder almacenar hasta 1.024 direcciones en las posiciones altas

**P4.25.** Suponga que en la posición de memoria A779 de CODE-2 se encuentra la instrucción de *salida* **H'5105**. Esta instrucción (*OUT OP5,r1*) lleva al puerto de salida *OP5* el contenido del registro *r1*. Suponiendo que en un momento dado en *r1* se encuentra el valor FFFF, realice una tabla donde se indiquen las distintas microoperaciones que deben generarse durante la ejecución de la instrucción, los valores que tienen en cada momento los registros *PC*, *AR*, *DR* e *IR*, y los cambios producidos en los puertos de salida.

**P4.26.** Suponga un computador de 8 GB de memoria principal, longitud de palabra 16 bits y con la pila de gestión de subrutinas en la memoria principal. En la posición de memoria 37BC A73D se encuentra la instrucción *F000*, que es una instrucción *RET* (retorno de subrutina). Indique detalladamente las microoperaciones que tendrían que generar la unidad de control para que se ejecute esta instrucción.

**P4.27.** El procesador de un computador de longitud de palabra de 16 bits dispone de los siguientes elementos: registro de dirección de memoria (*DM*), registro de memoria (*RM*), contador de programa (*PC*), puntero de pila (*SP*), registro instrucción (*IR*), conjunto de registros (*R0* a *R7*) y registro interno auxiliar (*Ra*). Suponga que la instrucción máquina de llamada a subrutina se forma yuxtaponiendo el código de operación (que es *A4*) y un número, *ds*, de 8 bits que especifica la dirección de memoria donde comienza la subrutina a la que hay que saltar. Suponiendo que en la dirección de memoria 37AB se encuentra la instrucción *A433*:

- a) Indicar en una tabla como la que se da a continuación las microoperaciones que se realizan y cómo varían los registros y posiciones de memoria que se indican, durante la ejecución de la instrucción ubicada en la posición 37AB (en la primera fila se dan los valores iniciales que no pueden deducirse del enunciado del problema).

de memoria, indicar en hexadecimal cuáles serían las direcciones físicas reservadas para esa hipotética pila.

### PROGRAMAS DE ALMACENAMIENTO EN MEMORIA

**P4.29.** Realizar un programa para CODE-2 que lea una serie de números del puerto de entrada *IPI*, y almacene los números pares en una zona de memoria que comience en la direc-

ción  $P$ , y los números impares en otra zona que comience en la dirección  $I$ . El programa finaliza cuando se dé por la entrada el carácter de control Unicode de final de texto ( $ETX$ ), teniendo entonces que facilitar por el puerto  $OP2$  los números,  $NP$  y  $NI$ , de datos pares e impares, respectivamente, leídos. Suponer que las direcciones de inicio de las tablas se encuentran en las posiciones  $A000$  y  $A001$  de memoria.

## TRATAMIENTO DE BITS Y PROCESAMIENTO DE TEXTOS

**P4.30.** A partir de la posición  $A000$  de memoria se encuentra una cadena de 4 caracteres, almacenada con el criterio del extremo mayor. Hacer un programa para almacenar dicha cadena a partir de la posición  $B000$  con el criterio del extremo menor.

**P4.31.** A partir de la dirección  $I = A000$  de la memoria de CODE hay una cadena de caracteres Unicode. Hacer un programa que dé por el puerto de salida  $OPI$  las direcciones de memoria donde se encuentre un carácter “igual” ( $=$ ) y por el  $OP2$  el número total de ellos, dentro de la cadena. *Nota:* El último carácter de la cadena es el código de fin de texto ( $ETX$ ).

**P4.32.** A partir de la posición  $PT = A000$  de la memoria de CODE se encuentra una cadena de caracteres ( $T$ ), siendo el último de ellos el carácter de control retorno de carro ( $CR$ ). Realizar un programa que lea del puerto de entrada  $IPI$  un carácter,  $C$ , y proporcione por el puerto de salida  $OPI$  las posiciones relativas dentro de la cadena donde se encuentra dicho carácter.

**P4.33.** A partir de la posición  $A000$  de la memoria de CODE-2 se encuentra un texto Unicode que concluye con un carácter de control “fin de texto” ( $ETX$ ). El texto se compone de líneas, cada una de ellas concluyendo con un carácter de control “retorno de carro” ( $CR$ ). Generar a partir de la dirección de memoria  $E000$  una tabla, cuyo primer elemento sea el número de líneas del texto, y en posiciones sucesivas se dé el número de caracteres de cada línea del texto original.

**P4.34.** A partir de la posición  $A000$  de la memoria de CODE-2 se encuentra un texto Unicode que concluye con un carácter de control “fin de texto” ( $ETX$ ). Sustituir en el texto las posibles letras que haya en mayúsculas por sus correspondientes minúsculas.

**P4.35.** En la memoria de CODE-2 se encuentra una cadena de caracteres Unicode. Hacer una subrutina que ubique en la posición  $595)_{10}$  el número de caracteres correspondientes a cifras decimales que se encuentran en la tabla. La dirección inicial de la tabla se encuentra en la posición de memoria  $600)_{10}$ , y finaliza con el carácter de control “fin de texto” ( $ETX$ ).

## TABLAS

**P4.36.** Dada una tabla de números que se inicia en la posición  $5F3C$  y su último elemento tiene de contenido  $5FFF$

(que no aparece como dato en cualquier otra posición de la tabla), hacer un programa para CODE-2 que indique por el puerto  $OPI$  las direcciones relativas dentro de la tabla donde se encuentra el dato ubicado en la posición  $A000$  de la memoria.

**P4.37.** Dada una tabla de  $N$  valores enteros (positivos y negativos) almacenados en posiciones contiguas de la memoria de CODE, a partir de la posición  $PI$ , construir un programa que determine el número de cambios de signo entre datos consecutivos en la tabla. El programa debe leer del puerto de entrada  $IPI$  los parámetros  $N$  y  $PI$ , y dar a través del puerto de salida ( $OPI$ ) el número pedido.

**P4.38.** A partir de la dirección  $PI = A000$  de CODE se encuentra una tabla,  $TO$ , de valores que finaliza con un carácter fin de texto ( $ETX$ ). Hacer un programa en ensamblador que genere dos tablas  $TP$  y  $TI$ , a ubicar a partir de las direcciones  $PIP = B000$  y  $PII = C000$ , de forma que  $TP$  contenga los valores de  $TO$  que estén en posición par (en  $TO$ ) y en  $TI$  los que estén en posición impar (en  $TO$ ).

**P4.39.** Disponemos a partir de la posición  $H'00B1$  de una tabla de calificaciones de D'15 alumnos, cuyas valoraciones van de 0 ( $H'0000$ ) a 100 ( $H'0064$ ), en la que después de la nota se encuentra un código numérico que identifica al alumno (palabra de 16 bits). Realizar una subrutina en CODE-2 que copie consecutivamente los códigos de los alumnos aprobados a partir de la posición  $H'00C0$ . Suponga que la nota de corte para aprobar se proporciona en la posición  $H'00B0$ . La subrutina se debe cargar a partir de la posición  $H'0000$ . Calcular el tiempo que tardaría en ejecutarse el programa si hubiesen aprobado 10 alumnos y si la frecuencia de reloj de CODE-2 fuese de 10 GHz.

## ORDENACIÓN Y BÚSQUEDAS

**P4.40.** Hacer una subrutina para CODE-2 que ubique en la posición  $0B02$  de memoria el primer dato que se repita 3 veces en una tabla  $T$  de números. La dirección inicial y la longitud de la tabla  $T$  se encuentran, respectivamente, en las posiciones de memoria  $0B00$  y  $0B01$ .

**P4.41.** Dada una tabla de números desordenada, hacer un programa para CODE que permita buscar en dicha tabla un número dado. Si el número está en la tabla, el programa debe indicar en qué posición está; y si el número no está en la tabla, debe incluirlo al final de la misma.

**P4.42.** A partir de la dirección de memoria indicada por el registro  $rA$  hay una tabla,  $TA$ , de caracteres Unicode, cuya longitud se da en el registro  $rC$ . Hacer un programa que genere una tabla,  $TB$ , a partir de la dirección que se da en el registro  $rB$ , que contenga los caracteres de la tabla  $TA$ , pero ordenados de menor a mayor. (*Nota:* Utilizar un método directo; es decir, buscar primero en  $TA$  el elemento menor, y colocarlo como primer elemento en la tabla  $TB$ , y así sucesivamente.)

**P4.43.** Dada una tabla ordenada (de menor a mayor, por ejemplo) de  $N$  elementos, un procedimiento de búsqueda (*bús-*

*queda dicotómica o binaria*) de un elemento dado consiste en comparar ese elemento con el que ocupa la posición  $N/2$  (si  $N/2$  no es entero, se redondea al entero siguiente o al anterior). Si el elemento que se busca es mayor que el  $N/2$ , la búsqueda se restringe a la mitad superior de la tabla, y si es menor, a la mitad inferior. A continuación se repite el proceso en la mitad correspondiente, y así sucesivamente, hasta que la búsqueda finaliza por haber encontrado el elemento que se busca, o hasta que se restrinja a un solo elemento de la tabla y se puede decidir que el elemento buscado no está en la tabla. Realizar un programa para CODE-2, que busque el valor que se encuentra en el registro  $R2$ , en la tabla ordenada de menor a mayor cuya dirección de inicio se encuentra en  $R3$  y su longitud en  $R4$ .

## CAMBIOS DE CÓDIGO

**P4.44.** A partir de la posición  $I = A000$  de la memoria de CODE hay una tabla de valores que tiene en las direcciones pares la representación binaria ( $BD$ ) de los 10 dígitos decimales, y en las impares consecutivas siguientes un código ( $CD$ ) que sustituye al  $BD$ . Hacer un programa para CODE que se almacene a partir de la dirección  $A015$ , que lea del puerto de entrada  $IP1$  cifras decimales (de  $0000$  a  $0009$ ) y escriba en el puerto de salida  $OP1$  los valores transformados. Si la entrada no corresponde a un dígito decimal  $BD$ , por el dispositivo de salida  $O$  debe visualizarse  $FFFF$ .

**P4.45.** Hacer una subrutina que transforme un carácter Unicode correspondiente a un dígito decimal en su correspondiente valor de número entero. Suponer que el carácter se da en el registro  $rA$ , y su representación como número entero hay que proporcionarla en el registro  $rB$ . Procurar que el programa sea lo más rápido posible.

## LISTAS

**P4.46.** A partir de la dirección  $5A70$  de la memoria de CODE-2 se tiene una lista encadenada. Cada elemento de la lista contiene 2 palabras: la primera es un dato y la segunda constituye el puntero al dato siguiente (contienen la dirección del siguiente elemento de la lista). El último elemento de la lista tiene de puntero  $0000$ . Los datos de la lista se encuentran dentro de ella ordenados de menor a mayor. En la dirección  $3000$  de memoria se encuentra un dato. Hacer un programa en ensamblador para incluir dicho dato en la lista, en el lugar que por su valor le corresponda. Suponer que el dato en  $3000$  siempre es mayor que el primer elemento de la lista y menor que el último elemento de la lista. (*Sugerencia:* No es necesario cambiar de posición ningún dato de la lista, ni la posición del dato que está en  $3000$ .)

## PROGRAMAS ARITMÉTICOS

**P4.47.** En el registro  $rB$  de CODE-2 se tiene un dato determinado. Indicar las instrucciones necesarias para poner a

uno los bits que ocupan una posición par dentro de dicho registro.

**P4.48.** En el registro  $rB$  de CODE-2 se tiene un dato determinado. Indicar las instrucciones necesarias para cambiar (de cero a uno y de uno a cero) los 8 bits menos significativos de dicho registro.

**P4.49.** Hacer un programa para CODE que lea un conjunto de  $N$  datos y contabilice y dé como resultado en el dispositivo de salida el número de datos positivos, negativos y cero, existentes en el conjunto.

**P4.50.** Hacer una subrutina para CODE-2 que ubique en la posición  $0A02$  de memoria el número de datos con valor comprendido entre  $-7$  y  $+7$  (inclusive) que hay en una tabla  $T$  de números enteros. La dirección inicial y la longitud de la tabla  $T$  se encuentran, respectivamente, en las posiciones de memoria  $0A00$  y  $0A01$ .

**P4.51.** Supóngase que en una zona de la memoria de CODE hay una tabla de valores numéricos que son grados Fahrenheit. Hacer un programa para CODE que convierta esta tabla de valores en otra de grados Celsius, quedando escrita en una zona de la memoria previamente determinada por el usuario del programa por medio de una instrucción de entrada en el puerto  $IP1$ .

**P4.52.** A partir de la posición  $D000$  de la memoria de CODE se encuentra una Tabla  $A$  de 100 valores enteros comprendidos entre 0 y 10 (ambos inclusive). Realizar un programa que sustituya en la propia Tabla  $A$  los valores anteriores por su cuadrado. Para hacer el cuadrado se dispone (previamente ya almacenada en memoria a partir de la posición  $E000$ ) de otra tabla (Tabla  $B$ ) que contiene sucesivamente los cuadrados de los números enteros del 0 al 10. A partir de la posición  $D000$  de la memoria de CODE se encuentra una Tabla  $A$  de 100 valores enteros comprendidos entre 0 y 10 (ambos inclusive). Realizar un programa que sustituya en la propia Tabla  $A$  los valores anteriores por su cuadrado. Para hacer el cuadrado se dispone (previamente ya almacenada en memoria a partir de la posición  $E000$ ) de otra tabla (Tabla  $B$ ) que contiene sucesivamente los cuadrados de los números enteros del 0 al 10 (ver Problema P4.15).

## PROGRAMAS DE ESTIMACIONES DE TIEMPO

**P4.53.** En posiciones sucesivas de la memoria de CODE, a partir de la dirección  $DI = H'9000$  se almacena una tabla de  $H'30$  datos. Realizar un programa que recorra la tabla, lleve, uno a uno, cada dato al puerto de salida  $OP1$  y mantenga allí el dato durante unos 2 segundos, antes de sobrescribirlo con el siguiente de la tabla. Suponer que la frecuencia de CODE-2 es de 1 GHz. (*Sugerencia:* Redactar una subrutina de retardo que realice la espera de 1 segundos (ver Problema P4.18).)

## COMPILACIÓN DE INSTRUCCIONES DE ALTO NIVEL

**P4.54.** Escribir una subrutina para CODE que permita ejecutar repetidas veces las sentencias escritas entre dos posiciones de memoria especificadas al llamar a la subrutina en los registros  $r8$  y  $r9$ . El número de veces a ejecutar ese bloque de sentencias viene determinado por tres valores numéricos que se dan en otros tres registros:  $rA$  (valor inicial),  $rB$  (valor final) y  $rC$  (incremento). Cada vez que se ejecuta el ciclo, el valor en  $rA$  se incrementa en  $rC$ , y siempre que  $rA \leq rB$ , se ejecuta el ciclo (es decir, el retorno de la subrutina se debe producir cuando  $rA > rB$ ).

## SUBROUTINAS

**P4.55.** Hacer una rutina para CODE-2 que proporcione el cuadrado de un número entero comprendido entre 0 y 9. El dato inicial se encuentra en  $A001$ , y la rutina debe suministrar su cuadrado en la posición  $A002$ . *Sugerencia:* Utilizar una tabla de consulta que contenga los cuadrados de los 10 números y obtener los cuadrados consultando dicha tabla.

**P4.56.** Realizar un programa que genere una tabla  $TC$  cuyos elementos sean el resultado de multiplicar los elementos correspondientes de dos tablas,  $TA$  y  $TB$  de números enteros de 8 bits. Las direcciones iniciales de las tablas ( $PITA$ ,  $PITB$ ,  $PITC$ ), así como su longitud ( $N$ ) deben leerse del puerto de entrada  $IP1$ . Utilizar una subrutina que se encuentra a partir de la posición  $E000$  y que recibe en  $rA$  el multiplicando, en  $rB$  el multiplicador y proporciona el resultado en  $rC$ .

**P4.57.** Dada una tabla de  $N$  datos enteros positivos almacenada desde la posición  $PI$ , hacer una subrutina para CODE ubicada a partir de la dirección  $E000$  que seleccione los datos mayores que  $LI$  y menores que  $LS$  (es decir, que cumplan  $LI < X < LS$ ), y los copie en una segunda lista que se almacenará a partir de la posición  $PINT$ . A la subrutina se le pasan los parámetros en los registros  $r8$  ( $N$ ),  $r9$  ( $PI$ ),  $rA$  ( $LI$ ),  $rB$  ( $LS$ ),  $rC$  ( $PINT$ ). Realizar un programa, que llame a la subrutina anterior, para obtener una tabla a partir de la posición  $DA00$ , con los datos de otra tabla de 30 elementos ubicados a partir de  $D000$  que cumplan la condición  $2.501 < X \leq 3.526$ .

**P4.58.** Hacer una subrutina para CODE-2 que proporcione por el puerto de salida  $OP1$  el número de veces que se encuentra repetido un dato en la misma dentro de una tabla que finaliza con el carácter de control fin de texto ( $ETX$ ). La dirección inicial de la tabla se proporciona a la subrutina en el registro  $rA$ , y el dato a comprobar en el registro  $rB$ . Hacer un programa, que llame a la subrutina anterior, para comprobar las veces que aparece el valor  $0000$  en una tabla que comienza en la posición  $B000$ .

## PROGRAMAS DE ENTRADA/SALIDA

**P4.59.** Hacer un programa para CODE que lea del puerto de entrada  $IP1$  un conjunto de  $N$  datos, siendo el último el carac-

ter fin de texto, y contabilice y dé como resultado sucesivamente en el puerto de salida  $OP2$  el número de datos positivos, negativos y cero, existentes en el conjunto.

**P4.60.** Hacer un programa para CODE-2 que muestre por el indicador de siete segmentos de la Figura 4.1 los valores hexadecimales correspondientes a los 4 bits menos significativos de los elementos de una tabla que comienza en la dirección  $PT = B000$ , y cuyo último valor (que no aparece como dato en cualquier otra posición de la tabla) es  $FFFF$ . Utilizar la subrutina definida en el ejercicio anterior.

**P4.61.** Se desea diseñar un sistema de control de temperatura con un procesador CODE-2. Los 8 bits más significativos del puerto de entrada  $IP3$  están conectados a la salida de un conversor A/D que proporciona la temperatura actual del recinto. Los 8 bits menos significativos de dicho puerto se utilizan para introducir el valor de la temperatura deseada ("consigna" de temperatura). El sistema debe disponer de 4 salidas, conectadas todas ellas al puerto de salida  $OP3$ , con las siguientes conexiones y funciones (entre paréntesis se especifica el bit correspondiente):

- O1(11): Si se proporciona en este bit 1 se acciona una electroválvula que abre la conducción de aire; por el contrario con 0 se cierra.
- O1(10): Si se saca en él 1 se cierra un relé que conecta el calefactor (calor), y debe estar a 0 para que se abra el relé; es decir, se desconecte el calefactor.
- O1(9): Actúa igual que el bit de salida O1(10), pero para conectar o desconectar un refrigerador.
- O(6:0): Estos 7 bits están conectados directamente a un indicador de 7 segmentos como el de la Figura 4.1 (cuando la entrada a un elemento del indicador es 1, éste se activa). Se desea que en el indicador aparezcan las siguientes letras:  
A, si el sistema de caldeo/refrigeración está apagado.  
F, si el sistema está enfriando.  
C, si el sistema está calentando.

Realizar un programa en ensamblador que realice el control de temperatura, teniendo en cuenta lo siguiente:

- a) El sistema debe actuar calentando o enfriando cuando la temperatura de consigna se diferencie (positiva o negativamente, respectivamente) en 1 o más grados con respecto a la temperatura ambiente.
- b) La conexión de aire debe actuar si y sólo si hay refrigeración o caldeo.
- c) El indicador de siete segmentos debe marcar la acción que se esté realizando.

## MICROPROCESADORES

**P4.62.** Realizar una tabla de microprocesadores comercializados por Intel a partir del año 2000, y que incluya las siguientes columnas: año de comercialización, modelo, ancho del bus de datos, memoria direccionable (principal y cachés), frecuencia de reloj y co-



mentarios sobre alguna característica relevante. *Sugerencia:* Utilizar Internet para obtener esta información; pueden encontrarse enlaces en:

*[http://atc.ugr.es/intro\\_info\\_mcgraw.html](http://atc.ugr.es/intro_info_mcgraw.html), o en  
<http://atc.ugr.es>*

- P4.63.** Realizar una tabla de microprocesadores de la serie PowerPC a partir del año 2000, y que incluya las si-

guientes columnas: año de comercialización, modelo, ancho del bus de datos, memoria direccionable (principal y cachés), frecuencia de reloj y comentarios sobre alguna característica relevante. *Sugerencia:* Utilizar Internet para obtener esta información; pueden encontrarse enlaces en:

*[http://atc.ugr.es/intro\\_info\\_mcgraw.html](http://atc.ugr.es/intro_info_mcgraw.html), o en  
<http://atc.ugr.es>*

# La memoria

En este capítulo presentamos distintos aspectos sobre la memoria, tanto interna como externa, de un computador. En el primer caso nos centramos en la descripción de la estructura y funcionamiento de la memoria principal (Sección 5.1.1), y de la memoria caché (Sección 5.1.2) como medio para mejorar notablemente los tiempos de acceso del procesador al sistema de memoria. Posteriormente (Sección 5.2) describimos los distintos sistemas de memoria, considerados en su globalidad, y mostrando cómo existe una auténtica jerarquía, existiendo un compromiso entre velocidad y capacidad, y apoyándose el funcionamiento de los más rápidos en los más capaces de su nivel inmediato inferior.

En cuanto a la memoria externa, se pasa revisión a los sistemas que tienen una mayor proyección: discos magnéticos (Sección 5.3.2), cintas magnéticas (Sección 5.3.3) y discos ópticos (Sección 5.3.4).

## 5.1. Memoria interna

En esta sección se considerarán las unidades de memoria semiconductoras (esto es, formadas por circuitos integrados) y que tienen mayor relación directa con el procesador: la memoria principal (Sección 5.1.1) y la memoria caché (Sección 5.1.2).

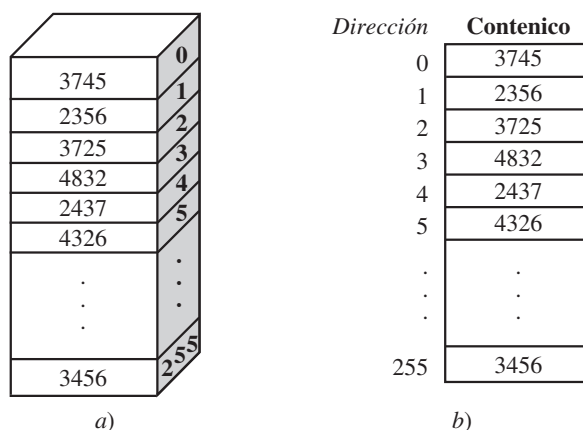
### 5.1.1. MEMORIA PRINCIPAL

La memoria principal de un computador está organizada en grupos de elementos de memoria denominados **palabras de memoria**. Una palabra es el conjunto de bits que se puede leer o memorizar en un instante dado, y su número de bits,  $n$ , se denomina **longitud de palabra**. La información, ya sean instrucciones o datos, se almacena en palabras de memoria, y ésta la podemos imaginar (Figura 5.1) como un conjunto ordenado de palabras, a cada una de las cuales se puede acceder, para recuperar o memorizar, indicando su **dirección** (posición relativa). Las longitudes usuales de palabras de memoria son 8, 16, 32 o 64 bits.

#### EJEMPLO 3.1

En una memoria de longitud de palabra de 16 bits, en cada palabra se puede almacenar una instrucción máquina (o una parte de ella) de 16 bits, o un dato numérico (o parte de él) de 16 bits o 1 carácter Unicode o dos caracteres ASCII de 8 bits.





**Figura 5.1.** Diagrama simplificado de una memoria principal de 256 palabras (en realidad tanto las direcciones como los contenidos son números binarios).

La Figura 5.2 muestra un esquema más detallado de la memoria principal de un computador. Para acceder (leer o escribir) a las posiciones de memoria, ésta dispone de un **bus de direcciones** de  $m$  hilos y dos **buses de datos**<sup>1</sup> de  $n$  bits, uno de entrada y otro de salida. Además, hay una entrada de control,  $R/W'$ , para especificar si se desea leer ( $R/W' = 1$ ) o escribir ( $R/W' = 0$ ) en ella. También, como se vio en la Sección 4.10, puede haber una señal para seleccionar la memoria que denominaremos  $IO/M'$ . El funcionamiento de la memoria puede resumirse así:

- **Lectura de memoria:** La unidad de control debe hacer llegar al bus de direcciones ( $AB$ ) la dirección de la posición de memoria donde se encuentra la información a leer. Además, debe generar las señales de control  $IO/M' = 0$ ,  $R/W' = 1$ ; al cabo de un cierto tiempo denominado **tiempo de acceso a memoria** se obtiene la información solicitada en el bus de salida ( $DBO$ ). A veces la memoria genera una señal de estado denominada  $MFC$  (*Memory Function Completed*) que indica cuándo la información de salida es válida. Como vimos en la Sección 4.1 el procesador suele disponer de dos registros, uno de dirección ( $AR$ ) y otro de datos ( $DR$ ), a través de los cuales se efectúan los intercambios con la memoria. Una operación de lectura de memoria se simboliza de la siguiente manera:

$$DR \leftarrow M(AR) \quad [5.1]$$

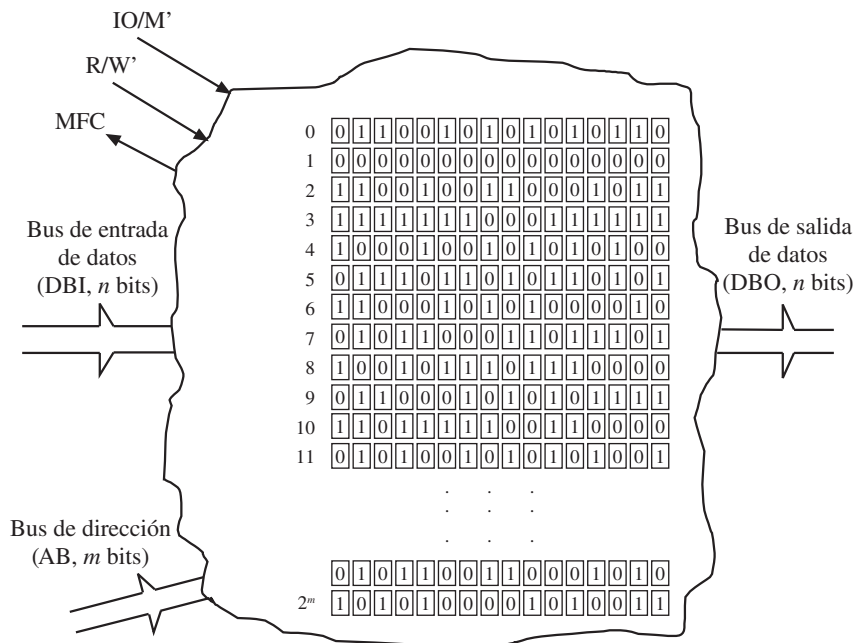
- **Escritura de memoria:** La unidad de control debe hacer llegar al bus de direcciones ( $AB$ ) la dirección de la posición de memoria donde se desea escribir, y en bus de entrada de datos ( $DBI$ ) la información a escribir. Además, debe generar las señales de control  $IO/M' = 0$ ,  $R/W' = 0$ . Transcurrido el tiempo de acceso a memoria se tiene garantía de que la información se ha almacenado correctamente ( $MFC = 1$ ). Una operación de escritura en memoria se simboliza así:

$$M(AR) \leftarrow DR \quad [5.2]$$

Si una memoria tiene  $m$  hilos en el bus de direcciones se dice que tiene un espacio de direcciones de  $2^m$  posiciones (direcciones de 0 a  $2^m - 1$ ). En cuanto a las unidades de información direccionables pueden considerarse dos tipos de memoria:

- **Memorias direccionables por palabras**, en las que cada dirección corresponde a una palabra de memoria, de forma que dos direcciones consecutivas corresponden a dos palabras consecutivas de memoria.

<sup>1</sup> Con frecuencia en lugar de dos buses de datos, uno de entrada y otro de salida, sólo se dispone de un único bus bidireccional, tanto para entrada como para salida.



**Figura 5.2.** Esquema de una memoria principal, mostrando sus buses de datos y direcciones, y las posibles señales de control.

- **Memorias direccionables por bytes**, en las que las direcciones corresponden a bytes de memoria. En este caso las direcciones de inicio de cada palabra serían múltiplos del número de bytes que incluye cada palabra, según se pone de manifiesto en el siguiente ejemplo.

### EJEMPLO 5.2

Sea una memoria direccionable por bytes y de longitud de palabra de  $n = 32$  bits, en la que en cada palabra caben 4 bytes, y las direcciones de comienzo de las palabras (Figura 5.3) son 0, 4, 8, 16, etc. Obsérvese que, en binario, todas las direcciones de palabra acabarán con dos ceros, de forma de que si  $m = 32$ , las direcciones de las palabras serían las que se indican en la Tabla 5.1.

Puede concluirse de este ejemplo que de los 32 bits de dirección, los 30 más significativos identifican la palabra, y los dos menos significativos el byte dentro de la palabra.

Cuando las palabras se organizan en la forma anteriormente indicada se tienen **direcciones alineadas**; ahora bien, como en realidad el direccionamiento de la memoria se hace por bytes, una palabra puede comenzar en cualquier posición arbitraria, y si no coincide con un múltiplo del número de bytes por palabra se tienen **direcciones no alineadas**.

Cuando una palabra incluye diversos bytes de un mismo dato o instrucción, según se comentó en la Sección 2.4 (Figura 2.2a) el byte almacenado en la parte más baja de las direcciones de memoria puede albergar los 8 bits menos significativos del dato o instrucción, o los más significativos, según se siga el **criterio del extremo menor** o el **criterio del extremo mayor**, respectivamente.

En la Sección 3.2.3.4 describimos brevemente cómo está constituida internamente una memoria RAM. Es conveniente recordar que, desde un punto de vista tecnológico, hay dos tipos básicos de memoria RAM: estáticas o SRAM y dinámicas o DRAM. Las memorias del primer tipo presentan, frente al segundo, la ventaja de una mayor velocidad (del orden de nanosegundos, *ns*, frente a decenas de nanosegundo), pero con las memorias del segundo tipo se obtienen mayores densidades de integración, menor consumo de energía y resultan más baratas.

Palabra	Dirección en binario (bytes)	Dirección HEX
0	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000
1	0000 0000 0000 0000 0000 0000 0000 0100	0000 0004
2	0000 0000 0000 0000 0000 0000 0000 1000	0000 0008
3	0000 0000 0000 0000 0000 0000 0000 1100	0000 000C
4	0000 0000 0000 0000 0000 0000 0001 0000	0000 0010
5	0000 0000 0000 0000 0000 0000 0001 0100	0000 0014
6	0000 0000 0000 0000 0000 0000 0001 1000	0000 0018
7	0000 0000 0000 0000 0000 0000 0001 1100	0000 001C
8	0000 0000 0000 0000 0000 0000 0010 0000	0000 0020
...	...	...
$2^{32}-4$	1111 1111 1111 1111 1111 1111 1111 0000	FFFF FFF0
$2^{32}-3$	1111 1111 1111 1111 1111 1111 1111 0100	FFFF FFF4
$2^{32}-2$	1111 1111 1111 1111 1111 1111 1111 1000	FFFF FFF8
$2^{32}-1$	1111 1111 1111 1111 1111 1111 1111 1100	FFFF FFFC

**Tabla 5.1.** Direccionamiento por bytes en una memoria con  $n = 32$  bits.

En general, los accesos a memoria pueden realizarse de tres formas:

- **Acceso por palabras:** Esta es la forma normal de acceso, recuérdese que una palabra era el conjunto de bits que se pueden leer o escribir en una operación aislada de acceso a memoria.
- **Acceso por bytes:** La memoria lee todos los bytes que componen la palabra a la que pertenece el byte, pero los no solicitados son ignorados por el procesador. En el caso de escritura de un byte, la memoria dispone de circuitos de control que únicamente hacen escribir el byte implicado en la operación, dejando inalterados los otros bytes de la palabra a la que se accede.
- **Acceso por bloques:** Con una gran frecuencia se solicita la lectura o escritura de posiciones consecutivas de memoria; tal es el caso que se presenta cuando se carga un programa en la memoria principal, o se salva el contenido de un archivo en memoria secundaria. Es posible transferir bloques de información, siendo necesario indicar al sistema de memoria tan sólo la dirección inicial del bloque y su tamaño.

### *Medidas de prestaciones*

Existen diversos parámetros para determinar las prestaciones de la memoria, como son los siguientes:

- **Tiempo de acceso a memoria ( $t_a$ ) o latencia:** Es el tiempo que transcurre desde el instante en que se presenta una dirección a la memoria y el instante en el que el dato queda memorizado o está disponible para ser usado.
- **Tiempo de ciclo de memoria ( $t_c$ ):** El tiempo mínimo que debe transcurrir entre dos accesos sucesivos.
- **Ancho de banda ( $AB$ ):** Número máximo de bytes que se pueden transmitir por segundo entre la memoria y el procesador. Este parámetro no sólo depende del tiempo de acceso a memoria, sino también del número de bytes a los que se puede acceder en paralelo y de la capacidad de transferencia del bus de interconexión entre memoria y procesador.

Es conveniente indicar que cuando se transfiere un bloque de datos se obtiene un rendimiento mayor que si se transfiriesen dichos datos uno a uno. La primera palabra transferida consume un tiempo igual al de una palabra aislada (por ejemplo, 5 ciclos de reloj), pero cada una de las siguientes palabras del bloque puede consumir tan sólo un ciclo de reloj. Una vez que se dispone a la salida de la memoria de un dato leído, se consumirá al menos otro ciclo de reloj para transferir el dato por el bus al procesador, aunque esta operación de transferencia puede solaparse con un acceso posterior a la memoria.

**EJEMPLO 5.3**

Suponiendo un procesador que consume 6 ciclos de reloj para acceder a una palabra, y 3 ciclos para accesos sucesivos, calcular el número de ciclos que se invierten desde que el procesador solicita un bloque de 16 palabras consecutivas hasta que las recibe.

**SOLUCIÓN**

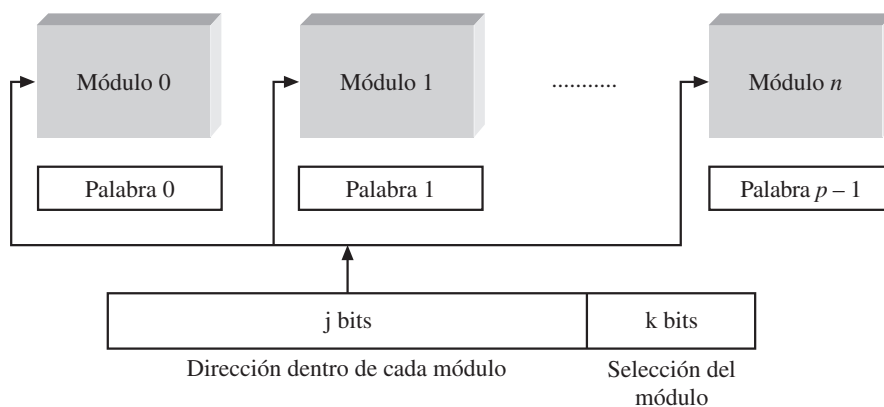
Tenemos que considerar los siguientes tiempos:

- Transmisión de la dirección procesador → memoria: 1 ciclo.
- Acceso a la 1.<sup>a</sup> palabra del bloque: 6 ciclos.
- Acceso a la 2.<sup>a</sup> palabra del bloque: 3 ciclos (en el primero de ellos simultáneamente se transmite la 1.<sup>a</sup> palabra de memoria → procesador).
- Acceso a la tercera palabra del bloque: 3 ciclos (en el primero de ellos simultáneamente se transmite la 2.<sup>a</sup> palabra de memoria → procesador).
- .....
- Acceso a la 16.<sup>a</sup> palabra del bloque: 3 ciclos (en el primero de ellos simultáneamente se transmite la 15.<sup>a</sup> palabra de memoria → procesador).
- Transmisión de la 16.<sup>a</sup> palabra de memoria → procesador: 1 ciclo.

Es decir:

$$N_c = 1 + 6 + 16 \cdot 3 + 1 = 56 \text{ ciclos} \quad [5.3]$$

Existen diversas técnicas para mejorar la velocidad de acceso a memoria; una de ellas se denomina **entrelazado de memoria**. Consiste en disponer varios,  $p$ , módulos de memoria en paralelo. Las conexiones se realizan de forma tal que los distintos módulos guardan posiciones consecutivas de memoria (Figura 5.3a); así el módulo 0 contendrá las posiciones 0,  $p$ ,  $2p$ ,  $3p$ , ...; el segundo módulo 1,  $p + 1$ ,  $2p + 1$ ,  $3p + 1$ , etc.; y el módulo  $p-1$ , las posiciones,  $p-1$ ,  $2 \cdot (p-1)$ , etc. Cuando se desea acceder a una determinada palabra, en realidad se acceden a  $p$  palabras consecutivas. El esquema de direccionamiento se muestra en la Figura 5.3b; si cada módulo almacena  $q = 2^j$  palabras y suponiendo que el número de módulos es  $p = 2^k$ , los primeros  $j$  bits seleccionarán la palabra de dirección  $j$  dentro de cada módulo, y los últimos  $k$  bits el módulo que contiene la palabra solicitada.



**Figura 5.3.** Esquema simplificado de un sistema de memoria entrelazada con cuatro módulos.

### EJEMPLO 5.4

Una memoria dispone de 16 módulos entrelazados, cada uno de ellos de 1 Mpalabra. El esquema de direccionamiento de memoria sería el siguiente:

- Para direccionar dentro de cada módulo, serán necesarios 20 bits, ya que  $2^{20} = 1$  Mega.
- Como hay 16 módulos, serían necesarios 3 bits para identificar cada uno de ellos, ya que  $2^4 = 16$ .

En consecuencia, las direcciones serían de 24 bits. Al leer, por ejemplo, la dirección *A35B78*, se leería en los 16 módulos 16 las palabras de posición *A35B7*, y de todas ellas la palabra solicitada sería la del módulo 8.

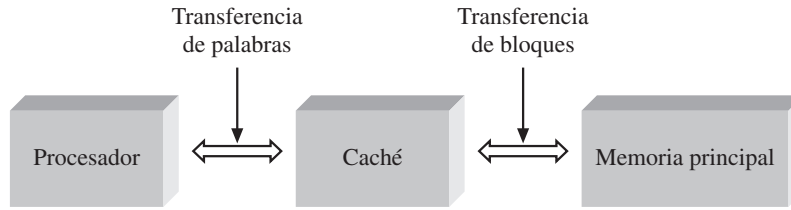
### 5.1.2. MEMORIA CACHÉ

Aunque la memoria principal es muy rápida (tiempos de acceso a una posición de memoria del orden de decenas de nanosegundo) es del orden de 20 veces más lenta que el procesador (unos nanosegundos), por lo que esta última unidad se ve frenada considerablemente cuando tienen que captar o escribir una palabra de memoria. Una forma de paliar este problema es utilizar una **memoria caché** que es un sistema de almacenamiento de tecnología más rápida, intermedia entre la memoria principal y el procesador. En efecto, la memoria caché se suele implementar con circuitos SRAM (Sección 3.2.2.4) que es una tecnología unas 10 veces más rápida que la usada para la memoria principal (DRAM), aunque más cara, de mayor consumo de energía eléctrica y con la que se obtiene una menor miniaturización. La memoria caché es usada por el sistema de memoria para mantener la información más comúnmente usada por el procesador, evitando así los relativamente lentos accesos a la memoria principal.

Tanto la memoria caché como la memoria virtual (que se analizará en la Sección 9.4.5) se fundamentan en el concepto de **localidad de las referencias**, que se basa en los siguientes hechos: cuando se ejecuta un determinado programa o se utiliza un grupo de datos, si se referencia un elemento es muy probable que los elementos cercanos a él tiendan a ser referenciados pronto (principio denominado de **localidad espacial**); y, en segundo lugar, debido a que los bucles son muy frecuentes en programación, si se referencia un elemento, tenderá a ser nuevamente referenciado pronto (principio de **localidad temporal**). Como consecuencia, cuando el procesador requiere una información determinada, son tácticas muy adecuadas:

- a) Recuperar no sólo el dato o instrucción requerida sino también los de direcciones próximas a él (consideración espacial). Ese grupo de direcciones adyacentes se denomina **bloque de datos**.
- b) Almacenar temporalmente el bloque de datos recuperado, en un subsistema con tiempo de acceso lo más próximo posible al del procesador, ya que con gran probabilidad en un plazo de tiempo pequeño volverá a ser solicitado (consideración temporal).

La memoria caché es una memoria que se sitúa entre el procesador y la memoria principal (Figura 5.4) y, simplícticamente, funciona de la manera que se describe a continuación. El procesador genera peticiones de acceso a memoria (lectura o escritura) proporcionando direcciones de posiciones de memoria. El controlador de la caché, a partir de la dirección emitida por el procesador comprueba si su contenido está o no en la caché; si lo está, se produce un **acierto en la caché** y se realiza la lectura o escritura en la caché. Si, por el contrario, no lo está, se produce un **fallo en la caché**, y como consecuencia de ello se carga en la caché un bloque de datos (**línea de caché**), que es un conjunto de palabras (64 bytes, por ejemplo) con direcciones consecutivas que contiene la instrucción o dato solicitado, y, simultáneamente, se pasa el dato o instrucción solicitado al procesador. Una vez que



**Figura 5.4.** Conexión de la memoria caché.

un bloque de datos se lleva a la caché, se deja allí el mayor tiempo posible. Claramente el concepto de memoria caché aprovecha los principios de localidad espacial (se almacenan en ella además de los datos solicitados los próximos) y de localidad temporal (los bloques de datos se mantienen en lo posible en la caché).

Los chips de los procesadores actuales suelen incluir en su interior uno o varios niveles de caché, y se puede utilizar otro nivel externo más capaz, aunque más lento.

Más detalles sobre la memoria caché se dan en la siguiente sección.

### EJEMPLO 5.5

El microprocesador Pentium IV dispone de tres cachés internas (L1I, L1D y L2). La L1I es para almacenar microoperaciones (12 Kmicrooperaciones), la L1D para datos (8 KB) y la L2 para instrucciones y datos (256 o 512 KBytes). El procesador accede directamente a el nivel 1 (L1I y L1D), y el nivel 1 se comunica con el exterior tan sólo a través del nivel 2 (L2).

## 5.2. Jerarquía de memoria

El procesador es la unidad más rápida del computador y debe captar instrucciones e intercambiar datos con la memoria; por lo tanto, es conveniente que las velocidades de funcionamiento de ambas unidades sean del mismo orden de magnitud; sin embargo, esto no es así. Los elementos de memoria más rápidos son los propios registros de uso general del procesador, pero éstos tienen una capacidad muy limitada. El siguiente tipo de memoria, en rapidez, es la memoria caché, pero resulta también escasa y cara. La memoria principal es mucho más capaz, pero también resulta insuficiente para gran cantidad de aplicaciones y además es volátil ya que al desconectar su suministro de energía eléctrica se pierde la información almacenada en ellos. Debido a ello es necesario disponer de otros sistemas de memoria que agrupamos bajo la denominación de **memoria externa**.

Considerando globalmente la forma de almacenamiento de información de un computador, se puede establecer una **jerarquía de memoria** (Figura 5.5), que hay que considerar bajo cuatro puntos de vista:

- a) Tamaño o capacidad,  $s$ , de almacenamiento suficiente.
- b) Tiempo de acceso,  $t$ , lo menor posible.
- c) Ancho de banda,  $b$ , alto.
- d) Coste por bit,  $c$ , reducido.

En el nivel superior se encuentran los registros internos de la unidad de procesamiento, y en el inferior las cintas magnéticas. Puede observarse (según se simboliza en la Figura 5.5), que cuanto más alto es el nivel menor es su capacidad, pero la velocidad es mayor; en concreto, se verifica:

$$s_i < s_{i+1}; t_i < t_{i+1}; b_i > b_{i+1}; c_i > c_{i+1} \quad [5.4]$$

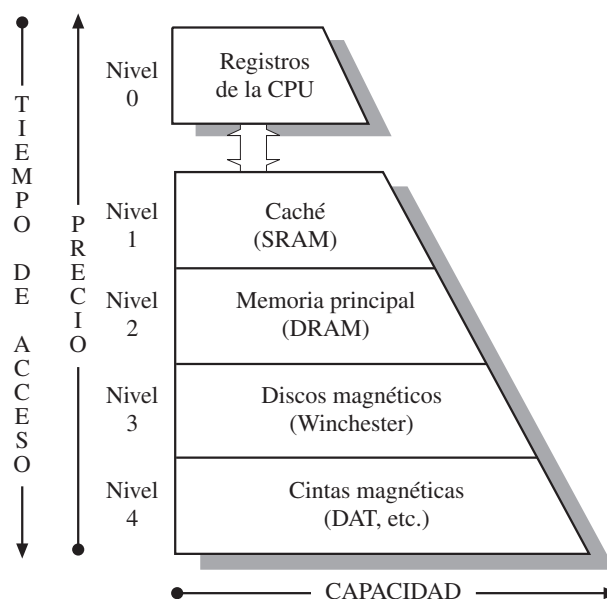


Figura 5.5. Jerarquía de memoria.

El procesador es el elemento principal del computador, ya que desde allí se controla el funcionamiento completo de éste y en él se hace el tratamiento de los datos. Interesa que las instrucciones y los datos con los que en un momento dado va a operar el procesador estén lo más próximos a él; es decir, en el nivel más alto de la jerarquía. En la sección anterior (Sección 5.1.2) vimos cómo la caché trata de cubrir este objetivo. De forma similar actúa la memoria virtual, que será estudiada en la Sección 9.4.5.

En general, cuando se solicita el contenido de una dirección de memoria en un determinado nivel y se encuentra allí se dice que se ha producido un **acierto**, y si no es así, un **fallo**. Por lo general, se satisface la propiedad de **inclusión**, según la cual la información en un determinado nivel se encuentra replicada en niveles inferiores. Cuando se produce un fallo en un determinado nivel, se copia el bloque de datos del nivel inmediatamente inferior, dentro del cual se encuentre la información solicitada: en el caso de la memoria virtual la transferencia se produce entre disco y memoria principal y el bloque de datos se suele denominar **página**. Una vez que una dirección está en los niveles superiores de la jerarquía (registros o caché) el procesador accede directamente a leer o escribir. Existen sistemas de **escritura inmediata** en los que en las operaciones de escritura se copian los datos en todos los niveles de la jerarquía donde se encuentran. Se obtienen mejores prestaciones con los **sistemas de postescritura**, según los cuales inicialmente sólo se modifican los datos en el nivel superior. En este caso, cada bloque mantiene un **bit de modificación** (*bm*) que se hace cero en el momento de cargarse el bloque en un nivel, y se hace uno caso de que el procesador escriba en él; en otras palabras el bit de modificación indica si el contenido del bloque coincide (*bm* = 0) o no (*bm* = 1) con la copia de ese bloque en el nivel inmediatamente inferior.

Según se van ejecutando los programas el nivel correspondiente se irá llenando de información, llegándose a la situación en la que esté completamente lleno. En este caso al tener que introducir un nuevo bloque, un **algoritmo de reemplazo** debe decidir qué bloque de datos debe desalojarse para acoger al nuevo. Caso de que el bit de modificación del bloque a desalojar sea cero, puede sobrescribirse sin más, ya que existe una copia exacta de él en el nivel inmediatamente inferior; si, por el contrario, el bit de modificación es uno, antes de alojar el nuevo bloque hay que actualizar la copia del antiguo existente en el nivel inmediatamente inferior.

Se denomina **tasa de aciertos**,  $\tau_{aciertos,i}$ , de un determinado nivel  $i$  al cociente entre el número de accesos realizados con éxito y el número total de accesos a ese nivel; y **tasa de fallos**,  $\tau_{fallos,i}$ , de un determinado nivel  $i$  al cociente entre el número de accesos realizados sin éxito y el número total de accesos a ese nivel. También se suele medir el volumen de aciertos y fallos en porcentajes (%).

Conociendo el tiempo de acceso de cada sistema de memorización en cada nivel, podemos obtener el tiempo medio de acceso a un nivel  $i$  con la siguiente expresión:

$$t_{a,i} = (\tau_{aciertos,i} \cdot t_i) + (\tau_{fallos,i} \cdot t_{i-1}) \quad [5.5]$$

donde  $t_i$  representa el tiempo de acceso del sistema de almacenamiento del nivel  $i$ .

## 5.3. Memoria externa

En este apartado vamos a presentar una serie de dispositivos, basados en principios magnéticos y ópticos, que son periféricos que actúan como prolongación de la memoria principal. El conjunto de estos dispositivos constituye la **memoria externa** del computador, y trata de solventar el problema de la volatilidad y de la relativa pequeña capacidad de la memoria interna. Los principales soportes que se utilizan como memoria externa (también denominada memoria auxiliar o masiva), y que describiremos en las secciones siguientes, son:

- **Memorias magnéticas:**
  - Disco magnético.
  - Cinta magnética.
- **Memorias ópticas:**
  - CD-ROM (Disco compacto).
  - DVD-ROM (Disco digital versátil).

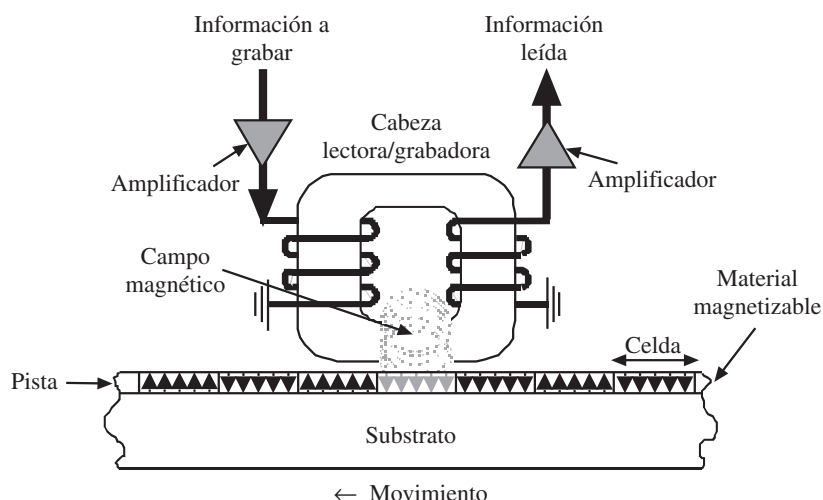
Previamente daremos unas ideas básicas comunes a todos los dispositivos magnéticos, indicando la forma de grabar y de leer la información.

### 5.3.1. ESCRITURA Y LECTURA DE INFORMACIÓN EN FORMA MAGNÉTICA

Los discos y cintas magnéticas contienen soportes de información constituidos por un sustrato, de plástico o aluminio, recubierto por un material magnetizable, tradicionalmente óxido férrico u óxido de cromo (Figura 5.6). La información se graba en unidades elementales o **celdas** que forman líneas o **pistas**. Cada celda puede estar sin magnetizar o estar magnetizada en uno de dos estados o campos magnéticos: norte (N) o sur (S). Los ceros y unos se pueden representar de muy diversas formas: magnetización N o S, cambio o no cambio de magnetización respecto a la celda anterior, etc. La celda se comporta como un elemento de memoria ya que almacena un bit. Para escribir o leer en una celda se utilizan señales eléctricas que actúan en una **cabeza** o **cápsula de lectura/escritura**, como la que esquemáticamente se muestra en la Figura 5.6.

Para escribir en una celda, una vez posicionada la cabeza sobre ella, se hace pasar por el devanado de escritura un pulso de corriente (del orden de 10 a 200 mA), que crea un campo magnético dentro del cual queda inmersa la celda del soporte. Dependiendo del sentido de la corriente (como el que se indica en la figura, o en sentido contrario) así será la polaridad del campo magnético creado y, por tanto, el estado en que queda magnetizada la celda.





**Figura 5.6.** Esquema que muestra el fundamento de la grabación y lectura en un soporte magnético por una cabeza lectora/grabadora.

La información de una celda se lee por medio de otro devanado y un amplificador-sensor. Como es conocido, un flujo magnético variable *induce* sobre una espira o bobina una fuerza electromotriz (tensión). Ésta tendrá una polaridad (+ o –, respecto de tierra) impuesta por el estado de magnetización (S o N) y el sentido de arrollamiento de la bobina. En la lectura se obtienen tensiones del orden de 0,1 a 100 mV. Las bobinas son usualmente de 10 a 1.000 espiras, y sus inductancias aproximadamente están comprendidas entre  $10\ \mu\text{H}$  y 100 mH. La separación de las bobinas con respecto a la superficie de la pista es de 1 a  $600\ \mu\text{m}$ .

En la actualidad la lectura se suele hacer con **cabezas magnetorresistivas**, desarrolladas por IBM, que se basan en el uso de materiales cuya resistencia eléctrica varía con el campo magnético en el que se encuentren inmersos. De esta forma se puede leer la información de la pista del disco (la escritura se sigue haciendo por inducción). También IBM, en 1991, propuso cambiar el recubrimiento de óxido magnetizable del sustrato por una **película delgada** (microscópica) de metal puro que se puede realizar por deposición al vacío u otros métodos utilizados en el proceso de fabricación de circuitos integrados, consiguiéndose mayores densidades de grabación.

La información contenida en un soporte magnético se transfiere desde y hacia el procesador o memoria principal no celda a celda, sino a ráfagas de información, denominadas **bloques o registros físicos**. El tiempo que se tarda en acceder a un registro determinado es variable, dependiendo, entre otras cosas, del lugar previo donde se encuentra la cabeza lectora. Como en el caso de la memoria interna, se denomina **tiempo de acceso (medio)** al tiempo que por término medio se tarda en acceder a cualquier registro físico. Si para acceder a un bloque concreto es necesario que la cabeza lectora vaya recorriendo (o leyendo) uno a uno los bloques que hay desde su posición inicial a la final, se dice que el dispositivo es de **acceso secuencial**. Si, por el contrario, la cabeza lectora puede posicionarse directamente en un registro dado (indicando su posición física) se dice que el dispositivo es de **acceso directo**. La memoria central y los discos magnéticos son unidades de acceso directo, por el contrario las cintas magnéticas son de acceso secuencial. Los dispositivos de acceso directo son mucho más rápidos que los de acceso secuencial.

### 5.3.2. DISCOS MAGNÉTICOS

Los discos magnéticos son sistemas de almacenamiento de información que en la actualidad constituyen el principal soporte utilizado como memoria auxiliar, tanto en los microcomputadores como en

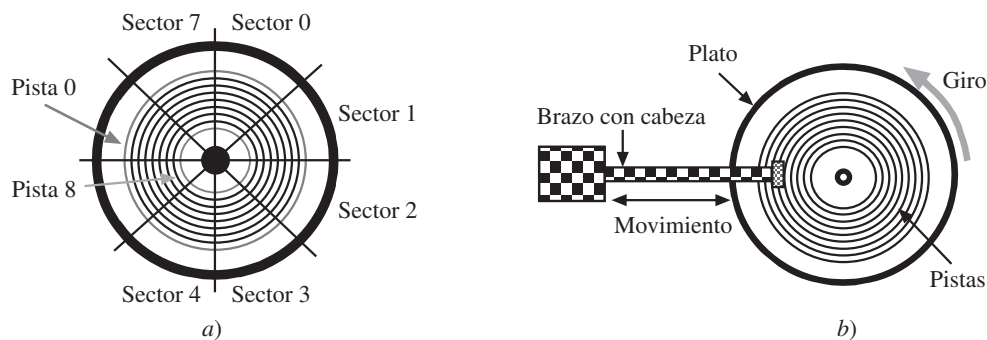
grandes sistemas informáticos. A pesar de que son más costosos que las cintas magnéticas y discos ópticos, tienen la ventaja sobre éstas de que son sistemas de acceso directo, con lo que se consiguen tiempos medios de acceso, del orden de 10 a 100 milisegundos, sustancialmente menores que con los discos ópticos y las cintas magnéticas.

### 5.3.2.1. Principios de funcionamiento

Los distintos tipos de discos magnéticos se fundamentan en la grabación magnética de la información en las superficies de un plato o disco circular recubierto de la capa magnetizable (óxido o película delgada). El plato o disco puede ser de plástico flexible, o puede ser rígido (usualmente de aluminio). En el primer caso tenemos disquetes (discos flexibles) y en el segundo caso discos rígidos o *duros* (*hard disks*).

Tanto en los discos duros como flexibles la información se graba en circunferencias concéntricas. Cada una de las circunferencias concéntricas grabadas constituye una **pista** (Figura 5.7), que se consideran numeradas correlativamente de fuera a dentro, empezando por cero. Asimismo el disco se considera dividido en arcos iguales denominados **sectores**, de forma que cada pista está compuesta por sectores. Los sectores también se consideran numerados en una secuencia única para todo el disco, y la capacidad de información del usuario que suele almacenarse en un sector es de 512 B.

La unidad física de lectura/escritura es el sector, y ésta es la unidad utilizada, por ejemplo, por el sistema de archivos NTFS. No obstante, hay sistemas operativos que utilizan como unidades de transferencia conjuntos de un número determinado de sectores, que denominan **unidades de asignación** (*clusters*). El número de sectores que conforma una unidad de asignación depende del tamaño y tipo de disco, suelen estar comprendidos entre 4 y 64 sectores.



**Figura 5.7.** a) Distribución de sectores y pistas en la superficie de un disco. b) El brazo se desplaza radialmente en busca de la pista a la que ha de acceder.

### EJEMPLO 5.6

Si se tienen 512 B/sector y 4 sectores/cluster, una unidad de asignación se compone de 2 KB, con lo que toda la información transferida debe ser troceada en bloques de 2 KB. Si tenemos un fichero de 270 Bytes, se desperdiciarán  $2 \cdot 1.024 - 270 = 1.778$  Bytes.

Según se observa en la Figura 5.7, los sectores de las pistas más exteriores son de mayor longitud que las interiores; ahora bien, el número de bits grabados en cada sector es siempre el mismo, con lo que la densidad de grabación (bits grabados por pulgada, bpi) será menor en las pistas exteriores que en las interiores. Esto es evidente si se tiene en cuenta que la velocidad de transferencia de información hacia, o desde, la superficie del disco es constante, con lo que, como el tiempo en recorrer un

sector interior es igual al de uno exterior, en ambos casos se grabará la misma cantidad de información.

La lectura y escritura en la superficie del disco, como se vio en la Sección 5.3.1, se hace mediante una cabeza o cápsula. La cabeza, en las unidades de cabeza móvil, está insertada en un extremo de un brazo mecánico móvil, que se desplaza hacia el centro o hacia la parte externa del disco bajo el control de los circuitos electrónicos del periférico (Figura 5.7b). El direccionamiento para leer o grabar un sector del disco se efectúa dando al periférico: el número de unidad, el número de superficie, el número de pista y el número de sector. El brazo sitúa rápidamente la cápsula encima de la pista correspondiente y espera a que el sector en cuestión se posicione (como consecuencia del giro del plato) bajo la cápsula.

En el acceso (lectura o escritura) de un bloque de información de capacidad  $C$ , por ejemplo, hay que considerar tres operaciones, cada una consumiendo su tiempo correspondiente:

1. La cabeza debe posicionarse encima de la pista donde se encuentra el sector inicial del bloque a transferir. La duración de esta operación se denomina **tiempo de búsqueda** ( $T_b$ ), y puede suponerse que es lineal. Depende del tiempo  $T_0$ , que tarda en reaccionar el motor de desplazamiento una vez recibida la orden, del tiempo medio que tarda la cabeza en atravesar una pista,  $t_p$ , y del número de pistas,  $n_p$ , a recorrer hasta posicionarse en la búsqueda. Se verifica, por tanto:

$$T_b = T_0 + n_p \cdot t_p \quad [5.6]$$

2. La cabeza espera encima de la pista hasta que, como consecuencia del giro del plato, el sector a acceder se posicione debajo de ella. El tiempo que dura esta operación se denomina **tiempo de espera** (o **latencia rotacional**),  $T_e$ , y depende de la velocidad de rotación del disco, y de la posición donde se encuentre el sector a acceder. Puede ser que en el momento que llega la cabeza a la pista, el inicio del sector se encuentre justo debajo de aquélla, con lo que el tiempo de espera sería cero; pero también puede ser que acabe de sobrepasarla con lo cual el tiempo sería el que dure una rotación completa; es decir,  $1/\omega_r$  siendo  $\omega_r$  la velocidad de rotación del disco en revoluciones/segundo. Teniendo en cuenta todas las situaciones posibles, por término medio el tiempo de espera será el que se tarda en realizar media rotación; es decir:

$$T_e = \frac{1}{2 \cdot \omega_r} \text{ segundos} \quad [5.7]$$

3. A continuación hay que considerar y estimar el **tiempo de lectura o tiempo de escritura** de los  $C$  bytes que constituyen el bloque. Los circuitos electrónicos y buses deben ser tales que sean capaces de leer al ritmo con que se desplaza el plato bajo la cabeza lectora/grabadora; con lo que, si una pista tiene una capacidad de  $C_p$  bytes, la velocidad de lectura será:

$$v_t = C_p \cdot \omega_r \text{ Bytes/segundo} \quad [5.8]$$

Y el tiempo total de lectura de los  $C$  bytes será:

$$T_t = \frac{C}{v_t} = \frac{C}{C_p \cdot \omega_r} \text{ segundos} \quad [5.9]$$

Entonces, el tiempo conjunto (total de acceso y lectura/escritura) es:

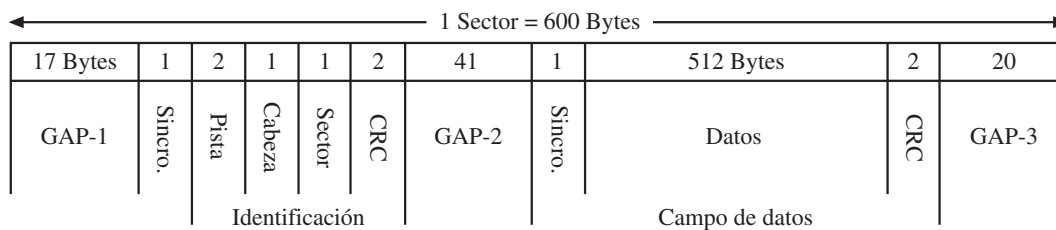
$$T_c = T_b + T_e + T_t \quad [5.10]$$

Se denomina **tiempo de acceso**,  $T_a$ , al tiempo que tarda la unidad en posicionarse al inicio del sector al que se quiere acceder. Del razonamiento efectuado anteriormente se deduce que será la suma del tiempo de búsqueda y del tiempo de espera; es decir:

$$T_a = T_b + T_e \quad [5.11]$$

Podemos concluir que para leer un determinado bloque de información consecutiva hay que sumar el tiempo de acceso al inicio del bloque y el tiempo de transferencia. De lo dicho anteriormente se deduce una cuestión práctica de una gran relevancia: cuanto mayor es el bloque a leer menor será el tiempo efectivo total de lectura o escritura. En otras palabras, se tarda menos en leer un bloque de  $C$  bytes si se puede realizar con un solo acceso en vez de hacerlo con varios accesos.

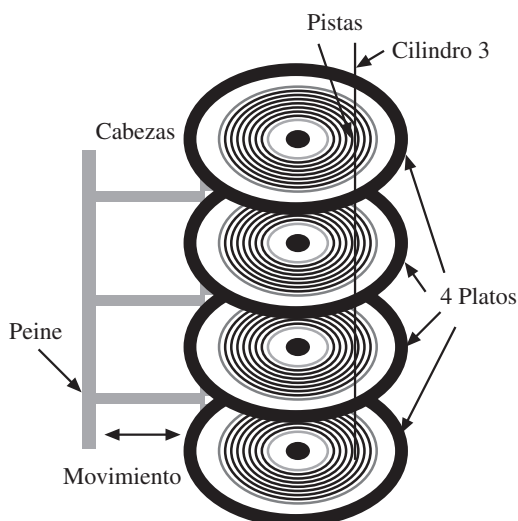
Otra consideración relevante es el formato con el que se almacenan los datos. Como indicamos anteriormente, la unidad básica de transferencia es el sector, y éste se puede grabar como se indica en el ejemplo de la Figura 5.8. Hay zonas donde no se graba nada de información (*Gaps*), un campo de identificación del sector, códigos detectores de error (*CRC*) y los datos del usuario (512 Bytes).



**Figura 5.8.** Ejemplo de información contenida en un sector de un disco (*GAP*, es una zona de separación, sin grabar; y *CRC* son bits de un código detector de errores).

Se pueden considerar cinco tipos de unidades de discos magnéticos:

- **Discos de cabezas fijas.** Tienen una cabeza individual de lectura/escritura por cada pista, consiguiéndose con ello tiempos de accesos relativamente bajos (del orden de milisegundos), ya que éstos vienen fijados únicamente por la velocidad de giro del disco.
- **Paquetes de discos.** Son unidades compuestas por varios platos que giran solidariamente alrededor de un eje común (Figura 5.9). Las cabezas de lectura/escritura son móviles, existiendo una por superficie, y se desplazan simultáneamente a gran velocidad radialmente buscando la pista en que se encuentra el sector que deben leer o escribir. Cada cabeza lee/graba en el sector correspondiente a su superficie. Cada grupo de pistas de igual radio se denomina **cilindro**, existiendo tantos cilindros en el dispositivo como pistas en una superficie. Con frecuencia las superficies externas no se utilizan para grabar; así, una unidad con 6 platos puede utilizar sólo 10 superficies, y una de 11 platos, 20 superficies.
- **Discos cartucho.** Son unidades con un plato y dos superficies de grabación, encerradas dentro de una carcasa, con una abertura lateral por donde se introducen las cabezas.
- **Discos Winchester (IBM, 1973).** Son paquetes de dos o más platos en los que, con objeto de reducir los efectos de la suciedad ambiental, están herméticamente cerrados y son fijos (no intercambiables). Las cabezas van más próximas a la superficie que las de las unidades descritas en los párrafos anteriores, lográndose grandes densidades de grabación. Las unidades de **discos duros** de hoy día son de tecnología Winchester, y pueden tener de 2 a 20 platos, girando a velocidades usualmente entre 3.600 y 10.800 rpm, conteniendo de 500 a más de 100.000 pistas por superficie, 32 a 800 sectores/pista, y con dimensiones de los platos de 1,3 a 14".
- **Disquetes.** Los disquetes son pequeños discos cuyos platos son flexibles, ya que están constituidos por un material plástico, Mylar, recubierto de óxido férrico. La velocidad de rotación de funcionamiento suele ser de 300 a 600 rpm, y son intercambiables. Los primeros disquetes eran de 8" y  $5\frac{1}{4}$ " y los utilizados en la actualidad son los de  $3\frac{1}{2}$ ".



**Figura 5.9.** Esquema simplificado de un paquete de discos.

Hay tres formas de grabación de discos:

- **CAV (*velocidad angular constante*)**. Es la forma tradicional de grabación, que se realiza a una velocidad de escritura y lectura de las pistas (en bits/segundo) constante, de forma que, como la velocidad de rotación también es constante, las pistas más internas se graban con una densidad mayor que las externas. De esta forma el número de bits grabados en cada sector es siempre el mismo, con lo que la densidad de grabación (bits/pulgada o bpi) es menor en las pistas exteriores.
- **ZCAV (o MZR) *velocidad angular constante por zonas* o ZBR (*Zoned-bit recording*)**. En los discos más avanzados, para conseguir mayor capacidad, la velocidad de grabación de las pistas exteriores se hace mayor, de forma que en las pistas más largas la capacidad de datos resulta mayor (se graban más sectores). De esta forma se consigue una densidad de grabación uniforme y que en las pistas externas haya más sectores que en las internas. Esta forma de grabación (que únicamente complica la electrónica del controlador, ya que los elementos electromecánicos no varían) tiene dos consecuencias prácticas relevantes: *a*) se consigue una mayor capacidad de almacenamiento y *b*) como la velocidad de giro sigue manteniéndose constante, a los datos de las pistas más externas se accede 2 o 3 veces más rápidamente que a los de las pistas internas. Con el **sistema ZCAV** las pistas contiguas se agrupan por zonas (8 a 30 zonas/disco). En los accesos a las zonas más exteriores se incrementa la velocidad angular, con lo que el número de sectores por pista se incrementa de las pistas interiores hacia las exteriores. Se obtiene un rendimiento óptimo del disco ubicando en los cilindros más externos la información que más se intercambia con el computador (los archivos de intercambio de la memoria virtual, ver Sección 9.4.5, los ficheros temporales de respaldo de seguridad, etc.).
- **CLV (*velocidad lineal constante*)**. Las pistas más externas contienen más información que las internas, rotando el disco más rápidamente en ellas. Este sistema es el que se utiliza en los discos ópticos (Sección 5.3.4).

Las unidades de disco, como la mayoría de periféricos, disponen de una memoria intermedia (*buffer*) en el controlador correspondiente, a través de la cual se realizan las transferencias con el

procesador o la memoria principal. La velocidad media de transferencia interna de datos (ancho de banda interno) viene dada por el número de bytes transferidos por segundo desde el disco al *buffer*.

El *buffer* permite adaptar los diferentes anchos de banda entre el bus interno y el bus externo. Por ejemplo, una unidad de disco puede tener un ancho de banda interno de 95 MB/s y estar conectada a un bus o interfaz SATA/150 con ancho de banda de 150 MB/s.

Las unidades de disco modernas disponen de la posibilidad de efectuar la escritura en una memoria caché en el controlador de discos. Esta facultad hace que la velocidad efectiva del periférico mejore considerablemente. El tamaño de la memoria caché puede llegar a decenas de MBytes en los discos de mayor capacidad; de esta forma en una caché de 8 MBytes se pueden almacenar hasta 16.384 sectores. El controlador, simultáneamente a ir recibiendo información a través del bus de E/S, va grabando la información de la caché en las superficies del disco. Así, además de obtener mayor rendimiento del procesador, la velocidad efectiva de escritura del disco es mayor ya que la información recibida en sucesivas operaciones se podrá grabar en sectores contiguos sin necesidad de consumir los tiempos de latencia inherentes a cada operación individual de lectura/escritura. También la caché permite aprovechar el principio de localidad espacial efectuando una **lectura anticipada de datos** con la que en la caché no sólo se almacena el sector solicitado, sino además todos los que le siguen (o una pista o cilindro completo, por ejemplo). Cuando se trata de leer un bloque de datos, primero se consulta si están en la caché, y si así es, se captan de ella, sólo en caso contrario se realiza la lectura de las superficies del disco.

### EJEMPLO 5.7

Como ejemplo de unidad de disco duro (paquete Winchester) para un servidor de archivos se puede citar el Seagate Barracuda 7.200.8, que tiene las siguientes características:

- Capacidad: 372 GB.
- Tiempo medio de búsqueda: 8 ms.
- Latencia rotacional media: 4,16 ms.
- Velocidad de giro: 7.200 rpm.
- Velocidad máxima de transferencia externa: 150 MB/s.
- Velocidad de transferencia interna: 95 MB/s.
- Bytes/sector: 512.
- Número de sectores: 781.250.000.
- Caché: 8 o 16 MB.

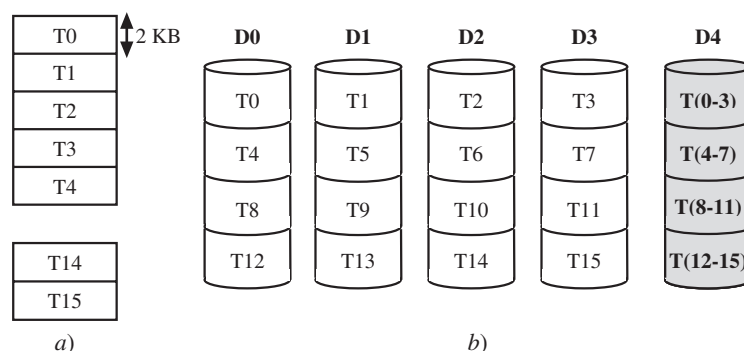
### 5.3.2.2. Unidades RAID

Una **unidad RAID** (*Redundant Array of Independent Disks*) o **agrupación redundante de discos independientes**<sup>2</sup>, es un conjunto de discos que funcionan en paralelo y que son considerados por el sistema operativo como una única unidad. El objetivo de este tipo de unidades es doble: aumentar la velocidad y mejorar la seguridad y fiabilidad de los datos almacenados.

La idea básica consiste en almacenar los datos en varios discos que funcionan en paralelo (Figura 5.10). El sistema operativo considera divididos los datos del archivo a almacenar (o leer) en *tiras* consecutivas, cada una de las cuales corresponde a un número determinado de sectores. Supongamos que cada tira es de 2 KB, y corresponde a 4 sectores; y que la unidad RAID se compone de 5 unida-

---

<sup>2</sup> RAID inicialmente eran las siglas de *Redundant Array of Inexpensive Disks* o agrupación redundante de discos de bajo coste.



**Figura 5.10.** Ejemplo de unidad RAID de nivel 4.

des de discos (D0 a D4). Entonces, si el archivo a almacenar es de 30 KB, se tienen en total 15 tiras (T0, T1, ..., T14). El sistema operativo da la orden a los controladores de almacenar la tira T0 en el disco D0, la T1 en el D1, la T2 en el D2, la T3 en el D3, la T4 en el D0, ..., y la T15 en el D3. Obsérvese que el disco D4 no se utiliza para almacenar los datos originales; en efecto, en el disco D4 se almacenan bits de comprobación. Concretamente se crea una tira de paridad, TP(0-3), en la que cada uno de sus bits corresponde al bit de paridad de los cuatro bits del mismo orden correspondientes a T0, T1, T2 y T3. La segunda tira, TP(4-7), almacenada en el disco D4 corresponderá a los bits de paridad de las tiras T4, T5, T6 y T7; y así sucesivamente.

Con este ingenioso esquema la velocidad se aumenta ya que, al funcionar en paralelo la velocidad de transmisión efectiva aumenta notablemente; así, si la velocidad de transmisión de un disco individual es de 40 MB/s, la memoria puede enviar datos a un conjunto RAID de 4 unidades a 160 MB/s. La fiabilidad del sistema aumenta considerablemente, ya que aparte de las redundancias de comprobación que se incluyen dentro de cada sector, existe un disco con redundancias adicionales. Si se avería una unidad cualquiera, la D2, por ejemplo, el sistema operativo, a partir del resto de los discos (incluyendo el D4), recalcula la información perdida en D2 y la almacena en el disco redundante (D4) (ver Problema P5.15). Una vez hecha esta operación, el disco D4 pasa a sustituir al D2, accediéndose a la información de los discos en el siguiente orden: D0, D1, D4 y D3; de forma que el sistema puede seguir funcionando normalmente. En el momento de tener reparado y listo de nuevo el disco D2, el sistema operativo vuelca toda la información de D4 en D2, y recalcula los bits de paridad para almacenarlos en D4; para prevenir futuras averías.

Existen diversas alternativas para organizar las unidades RAID, que se suelen conocer como niveles. La alternativa anteriormente explicada se conoce como RAID nivel 4.

### 5.3.3. CINTAS MAGNÉTICAS

Las cintas magnéticas se basan en los mismos principios de lectura/grabación de las cintas de los magnetófonos y casetes convencionales. El soporte de grabación consiste en un plástico (poliéster) muy flexible recubierto de un óxido magnetizable (óxido de hierro, óxido de cromo, etc.) de aproximadamente 100  $\mu\text{m}$  de espesor. La cinta se encuentra enrollada, y la lectura y grabación se efectúa haciéndola pasar por una estación de lectura/escritura al transferirla de un carrete en un eje de giro a otro carrete en otro eje. En las primeras décadas del uso de computadores las **cintas magnéticas de tipo carrete** se utilizaban como único sistema de memoria masiva, en la actualidad han sido sustituidas por cintas de tipo casete o cartucho. Las cintas magnéticas constituyen un soporte de información barato y de gran capacidad, pero son muy lentas (acceso secuencial). En la actualidad, la principal misión de las cintas magnéticas es obtener **copias de seguridad** de la información contenida en discos completos, o almacenar información obsoleta (ficheros “históricos”). Existen gran



cantidad de tipos y tecnologías de unidades de cintas magnéticas; los más importantes se enumeran en la Figura 5.11.

- *Cintas clásicas o de carrete* (cintas  $\frac{1}{2}$ " de ancho). Grabación lineal paralela de 7 o 9 pistas.
- *Cartuchos con cintas de  $\frac{1}{4}$ ",  $\frac{1}{2}$ " u 8 mm.*
- *Cartuchos compactos* (cartuchos pequeños):
  - QUIC (*Quarter Inch Cartridge*).
  - DAT (*Digital Audio Tapes*).
  - EXABYTE.
  - SAIT (*Super Advanced Intelligent Tape*).
  - DLT/SDLT (*Digital Linear Tape/Super DLT*).
  - LTO (*Linear Tape Open*).

**Figura 5.11.** Principales tipos y tecnologías de unidades de cintas magnéticas.

La grabación de una cinta se hace en unidades de información denominadas **bloques físicos** o **particiones** que contienen un conjunto de bytes de una longitud preestablecida, de forma similar a los sectores de los discos magnéticos. En el caso de las cintas clásicas de carrete la longitud del bloque podía ser seleccionada arbitrariamente por el usuario dentro de unos límites (usualmente entre 200 y 1.024 Bytes). La cinta iba leyendo o escribiendo bloque a bloque, que cargaba en su memoria intermedia. Cuando acababa de escribir en la cinta el bloque contenido en la memoria intermedia, la cinta se detenía, esperando a que el computador enviase el siguiente bloque (un proceso similar tiene lugar durante la lectura), para volver a repetir el proceso de escritura bloque a bloque. Debido a que la cinta no puede detenerse instantáneamente, entre cada dos bloques consecutivos se desperdicia (no se graba) un determinado espacio (de  $\frac{1}{2}$  a  $\frac{3}{4}$  de pulgada en las cintas de carrete) que se denomina **interbloque** o **IRG** (*Inter-record-gap*). Cada bloque contiene, además de los datos del usuario, secuencias preestablecidas de caracteres y espacios identificadores de los límites del bloque, e información adicional redundante para poder detectar automáticamente posibles errores de grabación.

En las cintas actuales es habitual que se disponga de dos modalidades de funcionamiento: de arranque/parada (o *start/stop*) y de bobinado continuo (*streaming*). Las primeras intercambian bloques de información de longitud similar a las cintas tradicionales, y las segundas intercambian bloques de gran longitud consiguiéndose aprovechar la cinta en aproximadamente un 95 por 100 o más (apenas hay IRG).

La capacidad de una cinta,  $C$ , depende fundamentalmente de su longitud,  $L$ , densidad de grabación,  $d$ , longitud de bloque y formato de grabación. Conforme han ido pasando los años y mejorando la tecnología las densidades de grabación se han incrementado desde unos 200 bpi a 160.000 bpi. Un cálculo aproximado, por exceso, de la capacidad puede efectuarse multiplicando longitud por densidad:

$$C \leq L \cdot d \quad [5.12]$$

Cuanto menores sean los bloques menos datos del usuario podrán almacenarse en la cinta, ya que por bloque se pierden los caracteres de identificación del bloque, los caracteres redundantes de detección de errores y el espacio IRG. No obstante, aplicando la fórmula anterior se puede obtener una idea de la capacidad máxima de una cinta. También conociendo la velocidad lineal de lectura/escritura, que se suele dar en pulgadas/segundo (ips o i/s) se puede determinar fácilmente la velocidad máxima de transferencia de datos entre la cabeza y la memoria intermedia (o computador).

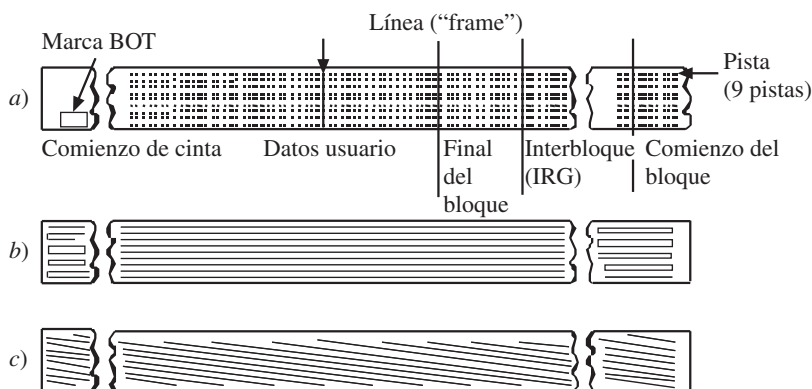
Existen tres técnicas básicas de grabar una cinta magnética (Figura 5.12), y cada tipo de unidad de cinta utiliza uno de ellos:

- **Lineal paralelo.** Se graba la información simultáneamente en varias pistas (Figura 5.12a). El conjunto de bits que se leen simultáneamente se denomina línea de grabación (*frame*), y en las



cintas tradicionales corresponde a un carácter (de 6 u 8 bits) con un bit adicional de paridad, siendo, por tanto, las cabezas de 7 o 9 pistas (es decir, la cabeza contiene 7 o 9 bobinas de lectura/escritura, respectivamente). Actualmente los cartuchos que utilizan esta técnica de grabación suelen tener de 2 a 8 pistas.

- **Lineal serpentina.** Las unidades de cartuchos más utilizadas son de 24 pistas que se graban en forma de *serpentina* y en un instante dado sólo se graba una pista (Figura 5.12b). La cabeza lectora/grabadora puede desplazarse verticalmente en 24 posiciones preestablecidas (en la Figura 5.12b sólo se explicitan 9). La cinta se desplaza horizontalmente, y cuando comienza a grabarse, la cabeza se ubica en la posición superior, continuando en ella hasta que se llega al final de la cinta, en cuyo momento la cabeza desciende hasta la posición inmediatamente inferior. Cuando se llega al otro extremo de la cinta, vuelve a descender la cabeza, y así sucesivamente.
- **Helicoidal.** El tambor de lectura/grabación gira, de forma que hay dos movimientos superpuestos, el de la cinta que se desplaza de un eje de enrollamiento al otro y el de la cabeza. De esta forma se obtienen velocidades relativas de desplazamiento entre tambor y cinta relativamente bajas, consiguiéndose mayor calidad de grabación y duración de la cinta. La posición del tambor y la disposición en él de las bobinas de lectura/escritura hacen que las pistas se graben en “forma de espiga”, formando un ángulo de  $6^\circ$  con el eje longitudinal de la cinta (Figura 5.12c).



**Figura 5.12.** Técnicas de grabación de una cinta: a) todas las pistas en paralelo; b) en serpentina, una única cabeza graba todas las pistas, y c) en espiga, con cabeza móvil de barrido helicoidal.

Las cintas magnéticas se pueden clasificar en los siguientes tipos (Figura 5.11):

- **Cintas clásicas o de carrete.** Son unidades para cintas de  $\frac{1}{2}$ " de ancho, y grabación lineal paralela de 7 o 9 pistas.
- **Cartuchos.** Existen diversos tipos de unidades de cartucho, como:
  - Con cintas de  $\frac{1}{4}$ " o de  $\frac{1}{2}$ " de ancho. Ejemplo de  $\frac{1}{2}$ ", Magstar, con grabado lineal, 128 pistas o más, y capacidades de hasta 20 GB.
  - Con cintas de 8 mm. Ejemplo Magstar MP (*Magstar MultiPurpose*), grabado lineal. Utiliza cartuchos tipo casete de 5 GB, y estas unidades se pueden configurar en forma de conjuntos o bibliotecas donde las cintas son manipuladas por robots.
- **Cartuchos compactos** (cartuchos pequeños):
  - QUIC. El nombre de esta tecnología se debe al tamaño de sus primeras cintas. Básicamente hay dos tipos: SLR (de  $5\frac{1}{4}$ " y Travan (de  $3\frac{1}{2}$ "). Las capacidades varían desde 40 MB a 25 GB.

- DAT. Cintas de 4 mm de grabación de barrido helicoidal, con velocidades relativas entre cinta y tambor de 0,32 i/s. Ejemplo DAT DDS-5 de 40 GB. Grabación digital.
- EXABYTE. Cintas de 8 mm de barrido helicoidal con velocidades relativas entre cinta y tambor 30 i/s. Grabación analógica. Capacidad 10 GB.
- SAIT. Cintas de  $\frac{1}{2}$ ", grabación helicoidal, con capacidades de 500 GB (SAIT-1) y 1 TB (SAIT-2). Grabación helicoidal. En estas cintas la capa magnetizable se forma vaporizando metal con cobalto utilizando técnicas avanzadas, consiguiéndose así muy grandes densidades de grabación (155.000 BPI en SAIT 1), velocidad de transferencia 30 MB/s.
- DLT/SDLT. Cintas de  $\frac{1}{2}$ ", 128 o 208 pistas lineales con capacidades de 40 a 80 GB, y anchos de banda de 3 a 8 MB/s. Con las unidades SDLT se consiguen capacidades de 110 a 300 GB con velocidades de transferencia de 11 a 36 MB/s.
- LTO. Estándar que define componentes tales como cabeza lectora/grabadora, trazado de pistas y tecnología de servomotores. Hay dos formatos (1998): Accelis (25 GB) y Ultrium (100 y 200 GB).

Los parámetros sobre capacidades y anchos de banda dados anteriormente corresponden a **datos nativos** (sin comprimir); con técnicas de compresión se consiguen capacidades y velocidades mayores.

### EJEMPLO 5.8

Las especificaciones de la tecnología SAIT-1 son las siguientes:

- Método de grabación: helicoidal.
- Densidad de grabación: 155.000 bpi.
- Longitud del bloque: fijo o variable.
- Capacidad 500 GB (con compresión ALDC 2,6:1, 1,3 TB).
- Velocidad de transferencia sostenida: 30 MB/s.
- Velocidad de transferencia máxima en ráfagas (síncrona SCSI): 160 MB/s.
- Tiempo medio de carga: 23 s.
- Tiempo de acceso medio: 70 s.
- Velocidad máxima de búsqueda: 394 pulgadas/s.
- Velocidad máxima de rebobinado: 551 pulgadas/s.
- Velocidad rotacional: 5.000 rpm.
- Tamaño memoria intermedia: 75 MB.
- Longitud de la cinta: 600 m de cinta de  $\frac{1}{2}$ ".

### 5.3.4. DISCOS ÓPTICOS

Los discos ópticos son dispositivos para almacenamiento masivo de información, cuya lectura se efectúa por medios ópticos. Existen diversos tipos, siendo los más relevantes:

- **CD** (*Compact Disk*) o **disco compacto**.
- **DVD** (*Digital Versatil Disk*<sup>3</sup>) o **disco digital versátil**.

---

<sup>3</sup> Inicialmente se denominaban *Digital Video Disk*, Disco Digital para Vídeo; al extenderse su uso a otras aplicaciones se ha cambiado el significado de las siglas.

Las características comunes de estos sistemas son:

- Alta capacidad de almacenamiento, típicamente entre 650 MB y 17 GB; es decir, en este último caso equivalente a unos 12.000 disquetes de 1,44 MB.
- El precio por bit es el más bajo de todos los dispositivos de memoria masiva.
- Los soportes de grabación (los discos) son intercambiables (como los disquetes), y son, aproximadamente, unas 10 veces más lentos que los discos duros y el doble de rápidos que los disquetes.
- La degradación o pérdida de información es prácticamente nula, ya que no se producen desgastes por lectura, y no necesitan altos requerimientos en la limpieza de sus superficies externas.

En la mayoría de los discos ópticos la información, a diferencia de las unidades de disco magnético, es grabada en espiral (y no en circunferencias concéntricas), y puede ser leída (dependiendo del tipo de unidad) a velocidad lineal constante (CLV), o a velocidad angular constante (CAV), como las unidades tradicionales de discos magnéticos (microsurcos). Tal como indicamos en la Sección 5.3.2.1, en las unidades CLV la velocidad de rotación depende de la posición radial de la cabeza (en este caso de 600 a 1.800 rpm), consiguiéndose directamente así que la densidad de grabación sea constante. A continuación se describe brevemente cada uno de los tipos de estos dispositivos.

#### 5.3.4.1. Discos compactos (CD)

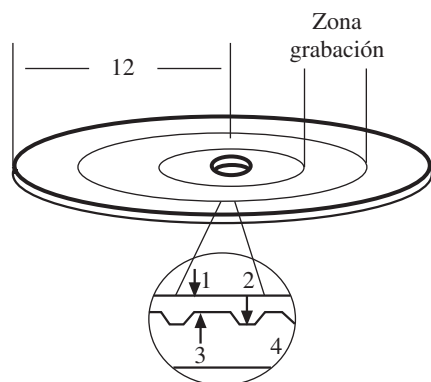
Existen tres tipos de discos compactos: CD-ROM, CD-R y CD-RW, que a continuación se describen brevemente.

##### **CD-ROM**

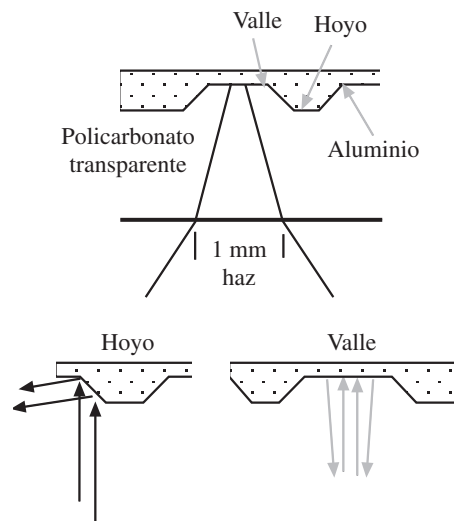
Los **CD-ROM** (*Compact Disc, Read Only Memory*) son dispositivos de sólo lectura, ya que el proceso de grabación resulta muy complejo, siendo de interés comercial sólo cuando se producen tiradas grandes de discos con la misma información. Esto hace que sus aplicaciones principales sean para grabar, por ejemplo, enciclopedias accesibles con computador, grandes manuales de computador, distribución de sistemas operativos, etc.

La información es almacenada en forma de **hoyos** (*pits*) y **valles** (*lands*), grabados mecánicamente sobre un sustrato de aluminio brillante, y es leída midiendo la luz de un haz láser reflejada sobre la superficie de hoyos y valles. En la Figura 5.13 puede verse un esquema simplificado de un CD-ROM. De la parte superior a la inferior se encuentra: la etiqueta, una capa protectora, la capa de aluminio brillante, en cuya superficie inferior se han grabado los hoyos y valles, y la base de plástico transparente (policarbonato). La lectura (según se muestra en la Figura 5.14) se efectúa por abajo. La superficie inferior de aluminio es recorrida, a través de la capa transparente inferior, por un haz de láser, que se refleja perpendicularmente en las superficies planas de hoyos y valles y se desvía a otra dirección en los bordes de los hoyos. Un fotosensor detecta cuando hay presencia o ausencia de luz reflejada perpendicularmente a la superficie, siendo transformada esta información en un valor eléctrico. La lectura, por tanto, se hace sin contacto físico ni desgaste; además la superficie con la información (en aluminio) no se encuentra en contacto directo con el exterior, por lo que la degradación o pérdida de datos es prácticamente nula.

Para grabar un CD-ROM la información binaria de datos se transforma con un código especial denominado **código de canal**, con el que cada byte (carácter ASCII) queda representado por un conjunto de 14 bits o **símbolo**. Los hoyos sobresalen de los valles  $0,12\ \mu\text{m}$  y tienen  $0,6\ \mu\text{m}$  de profundidad (espesor del Aluminio). La longitud a lo largo de pista de los hoyos y valles varía de  $0,9$  a  $3,3\ \mu\text{m}$ . La separación radial entre dos pistas consecutivas es de  $1,6\ \mu\text{m}$ , con lo que se obtiene una densidad de 16.000 pistas/pulgada (t/i), muy superior a la de los discos magnéticos (los disquetes 96 t/i).



**Figura 5.13.** Estructura básica de un disco óptico.



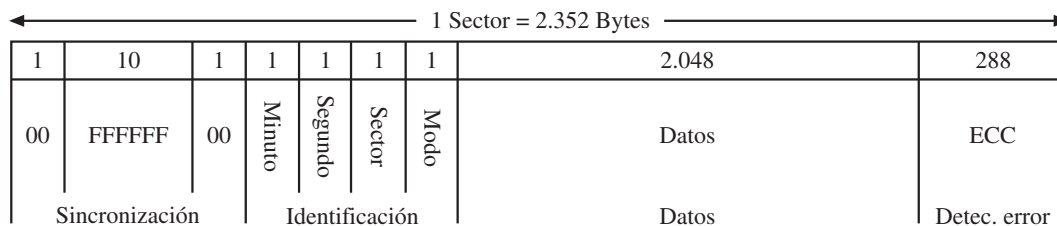
**Figura 5.14.** El haz es desviado al reflejarse en los bordes de los hoyos, y se refleja perpendicularmente en las superficies planas.

### Codificación de la información en CD-ROM

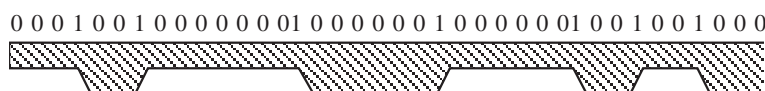
Para analizar cómo se almacena la información en un CD-ROM hay que considerar dos aspectos: 1) el formato lógico y 2) el formato físico de grabación o código de canal. Los formatos lógicos más usados son el ISO 9660 (para computadores con sistemas operativos Windows) y el HFS (*Hierarchical File Systems*, una variante del cual es usado en los Macintosh). El formato físico o código de canal más usual es el **EFM** (modulación de ocho a catorce).

Desde el punto de vista del formato lógico la información se organiza en **bloques (sectores)**. Cada sector contiene 2.352 Bytes (Figura 5.15):

- 12 bytes de sincronización.
- 4 bytes de identificación (ID): minuto, segundo, bloque, modo.
- 2.048 bytes de datos del usuario.
- 288 bytes de detección y corrección de errores:
  - 4 bytes de detección de errores (EDC).
  - 8 bytes todo ceros, 00<sub>H</sub>.
  - 276 bytes de corrección de errores (ECC).



**Figura 5.15.** Estructura lógica de un bloque o sector de un CD-ROM.



**Figura 5.16.** Las superficies de zonas planas y hoyos corresponde a “ceros”, y los bordes a “unos”.

### Otros CD

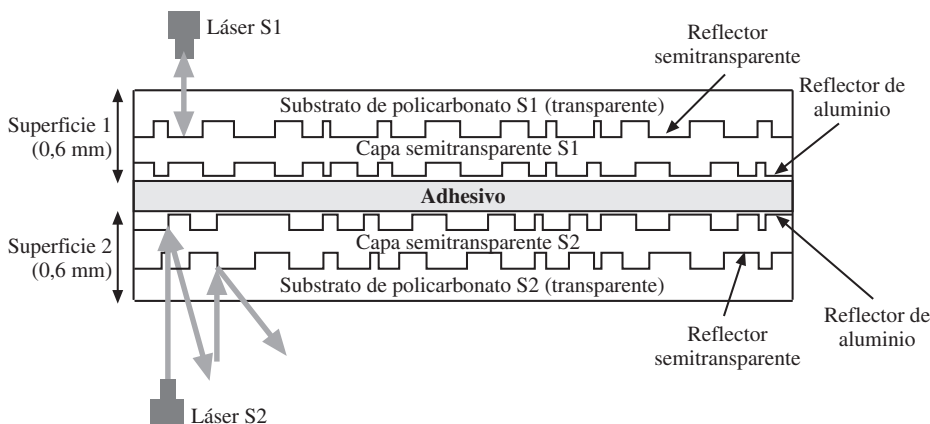
Según se indicó anteriormente hay otros dos tipos de CD:

- **Discos compactos grabables, CD-R (*CD Recordable*)**. Son unidades de discos ópticos similares a los lectores de CD-ROM, pero que contienen un láser de mayor potencia de forma que en la propia unidad se puede efectuar la grabación del disco. Por tanto, con una de estas unidades el propio usuario puede grabar (una sola vez) el disco. La capa reflectante es de oro en lugar de aluminio, y está recubierta de una capa de tinte (similar a las utilizadas en fotografía). Inicialmente, la capa de tinte es transparente, pasando a través de ella el láser, y pudiéndose reflejar sin problema en la capa de oro. Durante la grabación se hace incidir el haz láser con una mayor potencia (8 a 16 mW) que el de lectura (0,5 mW) en determinados puntos, quemándose el tinte y eliminándose su transparencia. Los puntos negros son interpretados como transiciones pozo-valle, o viceversa.
- **Discos compactos regrabables, CD-RW (*CD-ReWritable*)**. Son similares a los CD-R, pero la capa de tinte está formada por una aleación de plata, indio, antimonio y telurio, que posee dos estados estables: cristalino y amorfo, con dos índices de reflexión distintos (alto y bajo, respectivamente). El láser actúa con tres potencias posibles:
  - **Alta**: funde la aleación, convirtiéndola de estado cristalino a amorfo, representando una transición pozo-valle o valle-pozo.
  - **Media**: funde la aleación, haciéndola pasar al estado cristalino de alta reflexión.
  - **Baja**: no altera el estado de la aleación, y se utiliza para leer.

### 5.3.4.2. Disco digital versátil (DVD)

El fundamento físico de los DVD es el mismo que los CD. Existen también versiones DVD-R, DVD-RW (similares a las de los CD), con las siguientes mejoras:

- Los pozos son más pequeños (0,4 mm en lugar de 0,8 mm).
- La espiral es más pequeña (0,74 mm en lugar de 1,6 mm).
- La longitud de onda del láser menor (0,65 mm en lugar de 0,78 mm).
- Se pueden superponer dos capas, la primera de ellas semitransparente. El láser se enfoca a una capa u otra, dependiendo de la capa donde se desea que se refleje (se desee leer) (Figura 5.17).
- Se pueden grabar (y leer) ambas superficies.



**Figura 5.17.** Esquema simplificado de la estructura de un DVD.

Las tres primeras características hacen que se pueda grabar con una mayor densidad, y por tanto en la misma superficie se puede almacenar mayor información. También la utilización de distintas capas hace posible un incremento adicional de capacidad, existiendo unidades con la posibilidad de grabar hasta cuatro superficies (Figura 5.17).

Las unidades de DVD-ROM disponen de una caché usualmente de 256 o 512 KB, donde almacenan 128 o 256 sectores, respectivamente.

En la Tabla 5.2 se incluyen algunas características de discos ópticos. En los primeros CD la velocidad de transferencia era de 150 KB/s, y esta velocidad pasó a denominarse **factor de velocidad**. La velocidad 32X, quiere decir  $150 \cdot 32 = 4,69$  MB/s. En los DVD el factor de velocidad es de 1,35 MB/s.

Característica	CD-ROM	DVD
Capacidad	650 MB	4,7 GB
Factor de transferencia (x)	150 KB/s	1,35 MB/s
Velocidad de transferencia	12x a 52x	1x a 24x
Tiempo medio de acceso	80 a 150 ms	80 a 220 ms

**Tabla 5.2.** Parámetros usuales de distintos medios ópticos.

## 5.4. Conclusión

Este capítulo se ha dedicado a estudiar la memoria de un computador. En primer lugar (Sección 5.1) se han analizado las dos formas básicas en que se organizan las memorias semiconductoras (circuitos integrados) para integrar la memoria interna: memoria principal y memoria caché. En segundo lugar (Sección 5.2) se ha presentado el concepto de jerarquía de memoria, que trata de dar una visión general del sistema de memoria de un computador, justificar los distintos tipos de memoria y la interconexión de un nivel con sus adyacentes. Por último (Sección 5.3) se considera la memoria externa, que puede estar constituida por dispositivos de distinta naturaleza como son discos magnéticos, cintas magnéticas y discos ópticos.

### Test



**T5.1.** La longitud de palabra de una memoria:

- a) Coincide con el ancho del bus de direcciones.
- b) Coincide con el ancho del bus de datos.
- c) Es independiente del ancho de los buses.
- d) Necesariamente coincide con el ancho de los datos con los que opera la ALU.

**T5.2.** Suponiendo que una memoria se direcciona por palabras de 32 bits, si su bus de direcciones tiene 20 hilos, puede direccionar una memoria de hasta:

- a) 1 MByte.
- b) 4 MBytes.
- c) 5 MBytes.
- d) 2 MBytes.

**T5.3.** Suponiendo que el ancho de banda de un bus es de 133 MB/s y está constituido por 32 bits (líneas), la velocidad de transferencia en cada línea del bus es:

- a) 4,15625 Mbits/s.
- b) 33,25 Mbits/s.
- c) 532 Mbits/s.
- d) Ninguna de las contestaciones anteriores es correcta.

**T5.4.** Suponiendo que cada una de las líneas de un bus de 128 bits transmite a una velocidad de 33,3125 Mbits/s, el ancho de banda de dicho bus es:

- a) 3,84 MBytes/s.
- b) 2,08 MBytes/s.

- c) 533 MBytes/s.
- d) Ninguna de las contestaciones anteriores es correcta.

**T5.5.** Si el ancho de los buses de direcciones y de datos de un computador son, respectivamente, de 24 y 32 bits, entonces su capacidad máxima de memoria podrá ser de:

- a) 4 MBytes.
- b) 2 MBytes.
- c) 8 MBytes.
- d) 64 MBytes.

**T5.6.** Los dispositivos que se indican, en general, se ordenan de menor a mayor tiempo de acceso de la siguiente manera:

- a) Procesador, memoria principal, disco óptico, disco duro, cinta magnética.
- b) Procesador, memoria principal, disco duro, disco óptico, cinta magnética.
- c) Procesador, memoria principal, disco duro, cinta magnética, disco óptico.
- d) Procesador, disco duro, memoria principal, disco óptico, cinta magnética.

**T5.7.** Los registros de un procesador tienen un tiempo de acceso del orden de:

- a) Nanosegundos.
- b) Decenas a centenas de nanosegundos.
- c) Milisegundos.
- d) Décimas de segundos.

**T5.8.** La memoria caché de un computador actual tiene un tiempo de acceso del orden de:

- a) Nanosegundos.
- b) Decenas de nanosegundos.
- c) Milisegundos.
- d) Décimas de segundos.

**T5.9.** La memoria principal de un computador actual tiene un tiempo de acceso del orden de:

- a) Nanosegundos.
- b) Decenas a centenas de nanosegundos.
- c) Milisegundos.
- d) Décimas de segundos.

**T5.10.** Una unidad de disco duro actual tiene un tiempo de acceso del orden de:

- a) Nanosegundos.
- b) Microsegundos.
- c) Milisegundos.
- d) Décimas de segundos.

**T5.11.** Una unidad de disco óptico actual tiene un tiempo de acceso del orden de:

- a) Microsegundo.
- b) Milisegundos.
- c) Decenas de milisegundos.
- d) Segundos.

**T5.12.** Una cinta magnética actual tiene un tiempo de acceso del orden de:

- a) Microsegundos.
- b) Milisegundos.
- c) Décimas de segundo.
- d) Segundos.

**T5.13.** La capacidad de la memoria principal de un computador actual es del orden de:

- a) Cientos de Bytes.
- b) KBytes a MBytes.
- c) MBytes a GBytes.
- d) GBytes en adelante.

**T5.14.** La capacidad de la memoria caché de un computador actual es del orden de:

- a) Cientos de Bytes.
- b) KBytes a MBytes.
- c) MBytes a GBytes.
- d) GBytes en adelante.

**T5.15.** La capacidad de una unidad de disco óptico (CD) de un computador actual es del orden de:

- a) Cientos de Bytes.
- b) KBytes a MBytes.
- c) MBytes a GBytes.
- d) GBytes en adelante.

**T5.16.** La capacidad de una unidad de disquete de un computador actual es del orden de:

- a) Cientos de Bytes.
- b) KBytes.
- c) MBytes.
- d) GBytes.

**T5.17.** La memoria caché es una memoria cuyo tiempo de acceso está comprendido entre la de:

- a) Los registros del procesador y la memoria principal con objeto de disminuir la gran diferencia de rangos de velocidades entre estos elementos.
- b) La memoria principal y los discos duros con objeto de disminuir la gran diferencia de rangos de velocidades entre estos elementos.
- c) Los discos duros y discos ópticos con objeto de disminuir la gran diferencia de rangos de velocidades entre estos elementos.
- d) Los registros del procesador y los discos duros con objeto de disminuir la gran diferencia de rangos de velocidades entre estos elementos.

**T5.18.** Una cápsula magnetorresistiva de un disco magnético:

- a) Hace la escritura por inducción electromagnética y la lectura utilizando el efecto magnetorresistivo.
- b) Hace tanto la lectura como la escritura utilizando el efecto magnetorresistivo.
- c) Hace tanto la lectura como la escritura por medio de inducción electromagnética.
- d) Hace la lectura por inducción electromagnética y la escritura utilizando el efecto magnetorresistivo.



**T5.19.** Una celda de un disco magnético:

- a) Representa el uno con magnetización Norte y el cero con magnetización Sur, en toda la superficie de la celda.
- b) Representa el uno con magnetización Norte y el cero con magnetización Sur, en la zona central de la superficie ocupada por la celda.
- c) Representa el uno con magnetización Norte y el cero con magnetización Sur, en la zona inicial de la celda, dentro de la pista.
- d) Representa los ceros y unos dependiendo del código de grabación que se utilice.

**T5.20.** La unidad básica de almacenamiento y transferencia en un disco magnético (unidad física de lectura/escritura), en general es:

- a) La pista.
- b) El sector.
- c) El cilindro.
- d) 1 KByte.

**T5.21.** En un disco cuya velocidad de rotación es de 7.200 rpm, el tiempo medio de espera (latencia rotacional) es:

- a) 8,33 milisegundos.
- b) 0 segundos.
- c) 4,17 milisegundos.
- d) 69 microsegundos.

**T5.22.** En un disco de cabezas fijas, cuya velocidad de rotación es de 7.200 rpm, el tiempo medio de búsqueda de pista es:

- a) 8,33 milisegundos.
- b) 0 segundos.
- c) 4,17 milisegundos.
- d) Se dan datos insuficientes para conocerlo.

**T5.23.** La capacidad máxima de una unidad de disco de tipo CAV que dispone de 20 superficies, 3.000 cilindros y 64 sectores por pista se puede estimar en:

- a) 1,8 GBytes.
- b) 937,5 MBytes.
- c) 1,8 MBytes.
- d) 3,67 MBytes.

**T5.24.** Con el sistema de grabación de discos CAV (velocidad angular constante):

- a) Las pistas externas contienen más información que las internas.
- b) Todas las pistas se graban con igual densidad.
- c) Todos los sectores abarcan el mismo ángulo, y el número de bits grabados en cada uno de ellos es constante.
- d) Las pistas próximas se agrupan en zonas, y dentro de cada zona la densidad de grabación es la misma.

**T5.25.** Con el sistema de grabación de discos ZCAV (velocidad angular constante por zonas):

- a) Las pistas, conforme van siendo más externas, son grabadas con mayor densidad.
- b) Todas las pistas se graban con igual densidad.

- c) Todos los sectores abarcan el mismo ángulo, y el número de bits grabados en cada uno de ellos es constante.
- d) Las pistas próximas se agrupan, y dentro de cada grupo la densidad de grabación es la misma.

**T5.26.** Con el sistema de grabación de discos CLV (velocidad lineal constante):

- a) En todas las pistas se graba la misma capacidad de información.
- b) Todas las pistas se graban con igual densidad, incrementándose su capacidad cuantas más externas son.
- c) Todos los sectores abarcan el mismo ángulo, y el número de bits grabados en cada uno de ellos es constante.
- d) Las pistas próximas se agrupan en zonas, y dentro de cada zona la densidad de grabación es la misma.

**T5.27.** En un disco de cabezas fijas:

- a) El tiempo de búsqueda de la pista es cero.
- b) El tiempo de espera al sector (tiempo de latencia) es cero.
- c) El tiempo de transferencia de la información es nulo.
- d) El tiempo de lectura/escritura de un sector es cero.

**T5.28.** Suponiendo que  $C_{SF}$  es la capacidad de un disco sin formato y  $C_F$  es la capacidad del mismo disco una vez que se le ha dado formato, se verifica lo siguiente:

- a)  $C_{SF} < C_F$ .
- b)  $C_{SF} = C_F$ .
- c)  $C_{SF} > C_F$ .
- d) Depende del sistema operativo.

**T5.29.** La capacidad de un disco duro (Winchester) actual es del orden de:

- a) KBytes.
- b) MBytes.
- c) GBytes.
- d) TBytes.

**T5.30.** La velocidad de rotación de un disco duro (Winchester) actual es del orden de:

- a) Decenas de rpm (revoluciones/minuto).
- b) Centenas de rpm.
- c) Miles de rpm.
- d) Millones de rpm.

**T5.31.** Las cintas magnéticas en la actualidad se utilizan en informática para:

- a) Grabar imágenes de vídeo para sistemas multimedia.
- b) Grabar información de audio en equipos multimedia.
- c) Hacer copias de seguridad y guardar archivos históricos.
- d) Grabar información para utilizarla en forma de acceso directo.

**T5.32.** La diferencia fundamental entre un CD y un DVD es:

- a) Hay versiones de CD que se pueden regrabar, pero de DVD no.
- b) El DVD es un CD perfeccionado en el que se puede almacenar mucha más información que en este último.



- c) El DVD sólo se utiliza para almacenar información de vídeo y audio (películas, por ejemplo), mientras que en un CD se puede almacenar cualquier tipo de información digital.
- d) Un DVD es similar a un CD, pero la información se graba en pistas concéntricas y no en espiral (como ocurre en el CD).

**T5.33.** La capacidad de almacenamiento de un DVD es del orden de:

- a) Decenas de MBytes.  
 b) Centenas de MBytes.  
 c) GBytes y decenas de GBytes.  
 d) Cientos de GBytes.



## Problemas resueltos

### CONCEPTOS BÁSICOS Y MEDIDAS DE PRESTACIONES

- P5.1.** Una memoria tiene una capacidad máxima de 64 Mpalabras de 32 bits y con acceso por bytes. ¿Con cuántos bits se formarán las direcciones de memoria?

#### SOLUCIÓN

Como el acceso se realiza por bytes, tenemos que determinar la capacidad, en bytes, de la memoria del computador:

$$C = 64 \text{ Mpalabras} \cdot 4 \frac{\text{Bytes}}{\text{palabra}} = 256 \text{ MB} = 2^{28} \text{ Bytes}$$

Luego el número de bits que forman las direcciones es 28.

- P5.2.** El bus de direcciones de una memoria con acceso por bytes contiene 32 líneas, y sus palabras son de 64 bits:
- a) ¿Cuál es su capacidad máxima en palabras?
- b) ¿Cuál es la dirección que debe darse a la memoria para acceder a la palabra cuya dirección es 13B754C7.
- c) El byte de dirección CDA7 325A, ¿a qué palabra corresponde?

#### SOLUCIÓN

- a) Como el acceso se realiza por bytes, la capacidad máxima de la memoria en Bytes es:

$$C = 2^{32} \text{ Bytes}$$

Por otra parte, cada palabra contiene  $64/8 = 8$  Bytes. Luego la capacidad máxima en palabras es:

$$C = \frac{2^{32} \text{ Bytes}}{8 \frac{\text{Bytes}}{\text{palabra}}} = 2^{29} \text{ palabras} = 512 \text{ Mpalabras}$$

- b) Suponemos las direcciones alineadas. La dirección de la palabra contiene 29 bits, la dirección 23B754C7 en binario es:

$$1 \ 0011 \ 1011 \ 0111 \ 0101 \ 0100 \ 1100 \ 0111$$

Las direcciones de los bytes ocupan 32 bits, y las correspondientes a palabras alineadas son múltiplos de 16; es decir, acaban con tres ceros, con lo que la dirección de memoria de inicio de la palabra indicada será:

$$1 \ 0011 \ 1011 \ 0111 \ 0101 \ 0100 \ 1100 \ 0111 \ 000;$$

o, en hexadecimal:

$$1001 \ 1101 \ 1011 \ 1010 \ 1010 \ 0110 \ 0011 \ 1000 \rightarrow 9\text{DBA} \ \text{A638}$$

La palabra, por tanto, ocupará los bytes de dirección 9DBA A638 a 9DBA A63F.

- c) El byte de dirección:

CDA7 325A  $\rightarrow$  1100 1101 1010 0111 0011 0010 0101 1010

corresponde al tercer Byte (1.º: 000, 2.º: 001 y 3.º: 010) de la palabra:

1100 1101 1010 0111 0011 0010 0101 1  $\rightarrow$  1 1001 1011 0100 1110 0110 0100 1011  $\rightarrow$   
 $\rightarrow$  19B4 E64B

- P5.3.** Estimar el ancho de banda que podría obtenerse con un sistema de memoria entrelazado con 4 módulos, sabiendo que cada módulo está organizado en palabras de 32 bits, y que el tiempo de ciclo de las memorias utilizadas es de 60 ns.

#### SOLUCIÓN

La cadencia de bytes por segundo a la que se podría acceder por cada módulo sería la siguiente:

$$\text{Frecuencia de accesos: } \frac{1}{60 \cdot 10^{-9}} = 17 \cdot 10^6 \text{ accesos/s}$$

Ahora bien, como hay cuatro módulos a los que se accede en paralelo, y en cada uno de ellos se obtienen 4 Bytes (32 bits), la frecuencia global de bytes a los que se accede será:

$$\text{Frecuencia de acceso: } 17 \cdot 10^6 \cdot 4 \cdot 4 = 272 \cdot 10^6 \text{ Bytes/s}$$

Por otra parte, el ancho de banda dependerá del bus de interconexión de la memoria con el procesador o con las unidades con las que intercambie datos. Así, en un instante dado, se generan  $4 \cdot 32 = 128$  bits; con lo que si el bus fuese de 128 líneas y pudiese actuar a una frecuencia igual o superior a 17 MHz, el ancho de banda sería de 259 MB/s.

- P5.4.** Suponga un sistema de memoria que consume 8 ciclos de reloj para acceder a una palabra, y 4 ciclos para accesos sucesivos y con frecuencia de reloj de 750 MHz. Si el procesador requiriese un bloque de 16 palabras, estimar el tiempo que se invertiría desde que el procesador solicita el bloque hasta que concluye su recepción, en los siguientes supuestos:

- a) El acceso no se hace en forma de bloque, sino palabra a palabra.  
 b) El acceso se hace por bloques.  
 c) Se utiliza una memoria entrelazada de cuatro módulos.

#### SOLUCIÓN

- a) Para cada palabra se necesitarían los siguientes ciclos:

1 ciclo de envío de dirección +  
 8 ciclos de acceso a la palabra +  
 1 ciclo de envío al procesador de la palabra = 10 ciclos

Como hay que transmitir 16 palabras, se invertirán:  $10 \cdot 16 = 160$  ciclos.

Pero, teniendo en cuenta el período de cada ciclo, se verificará:

$$t = \frac{160}{750 \cdot 10^6} = 0,213 \cdot 10^{-6} = 213 \text{ ns}$$

En realidad se obtendría un tiempo mejor, ya que las operaciones de envío de direcciones o datos a través de los buses y las de acceso a memoria se pueden solapar. De esta forma, cuando se está accediendo a la segunda palabra se puede enviar al procesador la primera, y así sucesivamente; de esta forma se ahorrarían 15 ciclos y el tiempo total sería 193 ns.

- b) Si el acceso se realiza por bloques, se necesitarán los siguientes ciclos:

1 ciclo de envío de dirección  
 8 ciclos de acceso de la 1.ª palabra  
 4 ciclos de acceso de la 2.ª palabra y envío de la 1.ª  
 .....  
 4 ciclos de acceso a la 16.ª palabra, y envío de la 15.ª  
 1 ciclo de envío de la palabra 16.ª

Es decir, el número de ciclos es:  $10 + 15 \cdot 4 = 70$ , con lo que el tiempo de transmisión del bloque sería:

$$t = \frac{160}{750 \cdot 10^6} = 0,093 \cdot 10^{-6} = 93 \text{ ns}$$

c) Utilizando memoria entrelazada con cuatro módulos, se necesitarían los siguientes ciclos:

- 1 ciclo de envío de dirección.
- 8 ciclos de acceso, se accede de la 1.<sup>a</sup> a la 4.<sup>a</sup> palabra (una por módulo).
- 4 ciclos, se accede de la 5.<sup>a</sup> a la 8.<sup>a</sup> palabra, y se envían de la 1.<sup>a</sup> a la 4.<sup>a</sup>.
- 4 ciclos, se accede de la 9.<sup>a</sup> a la 12.<sup>a</sup> palabra, y se envían de la 5.<sup>a</sup> a la 8.<sup>a</sup>.
- 4 ciclos, se accede de la 13.<sup>a</sup> a la 16.<sup>a</sup>, y se envían de la 9.<sup>a</sup> a la 12.<sup>a</sup>.
- 4 ciclos para enviar de la 13.<sup>a</sup> a la 16.<sup>a</sup>.

En total el número de ciclos es 25, con lo que el tiempo de acceso resulta ser:

$$t = \frac{25}{750 \cdot 10^6} = 0,033 \cdot 10^{-6} = 33 \text{ ns}$$

Se observa cómo con los distintos métodos se obtienen distintas prestaciones.

**P5.5.** Se desea diseñar un sistema de memoria que tenga un ancho de banda de  $400 \cdot 10^6$  Bytes/s. Si se dispone de módulos de palabras de 16 bits con tiempos de ciclo de 15 ns, ¿cuántos módulos habría que utilizar de forma entrelazada para conseguir el objetivo?

**SOLUCIÓN**

El ancho de banda obtenible por cada módulo individual es:

$$AB_m = \frac{2 \frac{\text{Bytes}}{\text{módulo}}}{15 \cdot 10^{-9} \text{ s}} = 133 \cdot 10^6 \frac{\text{Bytes}}{\text{s}}$$

Con  $n$  módulos de memoria entrelazada se conseguiría un ancho de banda  $n$  veces mayor, con lo que:

$$n = \frac{AB_{\text{total}}}{AB_m} = \frac{400 \cdot 10^6}{133 \cdot 10^6} = 4 \text{ módulos}$$

## JERARQUÍA DE MEMORIA

**P5.6.** Suponga un sistema que no dispone de memoria virtual, y con tan sólo dos niveles de jerarquía (caché y memoria principal). Si los tiempos de acceso de los dispositivos utilizados son 5 y 50 ns, respectivamente, y el porcentaje de aciertos en caché del 95 por 100, ¿qué porcentaje de mejora se ha obtenido en el tiempo de ciclo al haber introducido la caché?

**SOLUCIÓN**

Si no hubiese caché, el tiempo medio de acceso sería el de la memoria principal: 50 ns, ya que el procesador efectuaría todos los accesos a ella.

Con memoria caché el tiempo medio de acceso sería, de acuerdo con la expresión (5.11):

$$t_{\text{acceso}} = (\tau_{f,\text{caché}} \cdot t_{\text{caché}}) + (\tau_{f,\text{caché}} \cdot t_{\text{mp}}) = (0,95 \cdot 5) + (0,05 \cdot 50) = 7,25 \text{ ns}$$

Es decir, el tiempo de acceso del sistema de caché mejora con respecto al sistema sin ella en un:

$$\text{mejora} = \frac{50 - 7,25}{50} \cdot 100 = 85,5 \%$$

**P5.7.** El sistema de memoria de un computador contiene una caché con tiempo de acceso de 4 ns, una memoria principal con tiempo de acceso de 80 ns, y una unidad de disco donde se gestiona la memoria virtual con tiempo de acceso de 12 ms. Después de analizar el comportamiento de la memoria se concluye que

los porcentajes de aciertos en la caché es del 80 por 100 y en la memoria principal del 99,5 por 100. Obtener los tiempos medios de acceso de caché y de memoria principal.

#### SOLUCIÓN

Como para obtener los tiempos de acceso de un determinado nivel es necesario conocer el tiempo de acceso del nivel anterior, debemos empezar por el nivel inferior:

- Tiempo de acceso del disco: 12 ms, y su porcentaje de aciertos es del 100 por 100, ya que el programa está completamente en el disco, y nunca se producirá algún fallo.
- Tiempo de acceso medio de memoria principal:

$$t_{am,mp} = (\tau_{a,mp} \cdot t_{mp}) + (\tau_{f,mp} \cdot t_{disco}) = (0,995 \cdot 80) + (0,005 \cdot 12 \cdot 10^6) = 60.080 \text{ ns} \approx 60 \mu s$$

- Tiempo de acceso medio de caché:

$$t_{am,caché} = (\tau_{a,caché} \cdot t_{caché}) + (\tau_{f,caché} \cdot t_{am,mp}) = (0,8 \cdot 4) + (0,2 \cdot 60.080) = 12.019 \text{ ns} \approx 12 \mu s$$

**P5.8.** Se dispone de un sistema de memoria con una jerarquía de tres niveles: caché, memoria principal y memoria virtual. Se observa que de 1.500.000 referencias a memoria que hace el procesador, en 1.300.000 casos la dirección se encuentra en la caché y en 150.000 casos en la memoria principal. Obtener las tasas de acierto de los distintos niveles.

#### SOLUCIÓN

- Tasa de aciertos de acceso a la caché:

$$\tau_{a,caché} = \frac{n.^o \text{ de accesos con éxito}}{n.^o \text{ de accesos solicitados}} = \frac{1.300.000}{1.500.000} = 0,87$$

- Tasa de fallos de caché:

$$\tau_{a,caché} = 1 - 0,87 = 0,13$$

- Tasa de aciertos de memoria principal. Téngase en cuenta que las solicitudes de acceso que llegan a la memoria principal son las que no ha podido atender el nivel inmediato superior; es decir:  $1.500.000 - 1.300.000 = 200.000$ ; con lo que:

$$\tau_{a,mp} = \frac{n.^o \text{ de accesos con éxito}}{n.^o \text{ de accesos solicitados}} = \frac{150.000}{200.000} = 0,75$$

- Tasa de fallos de memoria principal:

$$\tau_{f,mp} = 1 - 0,75 = 0,25$$

- Tasa de aciertos de disco:

$$\tau_{a,disco} = 1$$

- Tasa de fallos de disco:

$$\tau_{f,disco} = 0$$

### DISCOS MAGNÉTICOS

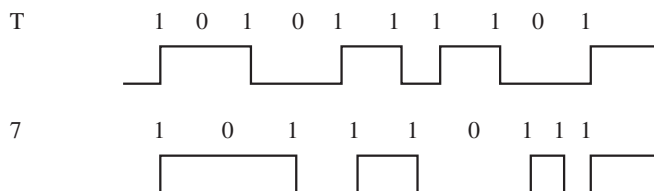
**P5.9.** Una unidad de disco recibe la información del procesador en código ASCII (paridad impar), y utiliza como sistema de grabación magnética el MNRZI. Este sistema trata de lograr que nunca se graben más de dos ceros seguidos, consiguiéndose así mayores densidades de grabación, y un uno se representa por cambio de magnetización. Si la transformación de código es la que se indica en la Tabla 5.3, ¿cómo sería la información a grabar de los caracteres T7?

Carácter	ASCII (impar)	MNRZI
T	01010100	1010111101
7	00110111	1001110111

Grupo de 4 bits	5 bits grabados
0000	11001
0001	11011
0010	10010
0011	10011
0100	11101
0101	10101
0110	10110
0111	10111
1000	11010
1001	01001
1010	01010
1011	01011
1100	11110
1101	01101
1110	01110
1111	01111

**Tabla 5.3.** Transformación de código para grabación MNRZI.

La señal magnética en código NRZ1 sería:



**P5.10.** Una unidad de paquetes de discos tiene 12 platos que giran a 7.200 rpm, con 24 cabezas de lectura/grabación, una por superficie. Si, una vez formateado el disco, cada superficie contiene 24.247 pistas con 793 sectores, por término medio, cada uno de 512 Bytes, obtener:

- El número de pistas por cilindro.
- El número de medio de bytes por pista.
- La capacidad total de la unidad.

**SOLUCIÓN**

- El número de pistas por cilindro es 24, ya que coincide con el número de cabezas.
- El número medio de bytes por pista es:

$$C_{pista} = 793 \text{ sectores/pista} \times 512 \text{ Bytes/sector} = 406.016 \text{ Bytes/pista}$$

- Capacidad total del disco con formato será:

$$C_f = 24.247 \text{ pistas/sup} \times 24 \text{ sup/unidad} \times 406.016 \text{ Bytes/pista} = 236272078800 \text{ bytes/unidad} = 220 \text{ GBytes}$$

**P5.11.** En la unidad de disco del ejercicio anterior, indicar la capacidad real del disco sin formatear, suponiendo que el formato que se le dio es el de la Figura 5.8. ¿Qué porcentaje de la unidad se desperdicia al dar el formato?

**SOLUCIÓN**

De la Figura 5.8 se deduce que cada sector realmente ocupa 600 bytes, de los que se aprovechan 512 para datos del usuario; es decir, la tasa y porcentaje de utilización del disco son, respectivamente:

$$\tau = \frac{512}{600} = 0,853 \quad ; \quad Porc = \tau \cdot 100 \%$$

Con lo que la capacidad del disco virgen (sin formato) será:

$$C_v = \frac{C_f}{\tau} = \frac{220}{0,853} = 257,81 \text{ GB}$$

- P5.12.** ¿Cuál debería ser el ancho de banda interno (entre la unidad física y la caché) de la unidad de los ejercicios anteriores suponiendo que en una revolución del disco se tuviese que leer una cuarta parte de los sectores de un cilindro?

#### SOLUCIÓN

Para obtener el ancho de banda debemos considerar la situación más extrema, que es cuando se transmite un cilindro completo. Las 24 cabezas lectoras leen y escriben a la vez y un cilindro es la información leída/escrita en una revolución.

Capacidad de un cilindro:

$$C_{cilindro} = \frac{C_{unidad}}{N_{cilindros}} = \frac{220 \text{ GB}}{24.247} = 9,29 \text{ MB}$$

Tiempo en leer un cilindro o tiempo en dar una revolución:

$$t_{cilindro} = \frac{60 \text{ s/m}}{7.200 \text{ c/m}} = 8,33 \cdot 10^{-3} \text{ s}$$

Con lo que el ancho de banda será:

$$AB = \frac{C_{cilindro}/4}{t_{cilindro}} = \frac{2,32 \text{ MB}}{8,33 \cdot 10^{-3} \text{ s}} = 286,7 \text{ MB/s}$$

- P5.13.** Suponiendo que la unidad de disco del Problema P5.10 tiene un tiempo medio de búsqueda de la pista es  $T_b = 7,4 \text{ ms}$ , y que, una vez que se accede a un sector se tardan  $T_{les} = 0,1 \mu\text{s}$  en leer y comprobar el campo identificación del sector, obtener el tiempo medio de acceso al campo de datos del sector.

#### SOLUCIÓN

El tiempo de espera o latencia rotacional será:

$$t_e = \frac{60 \text{ s/m}}{2 \cdot 7.200 \text{ c/m}} = 4,16 \text{ ms}$$

Con lo que el tiempo de acceso al sector es:

$$T_a = T_e + T_b = 7,4 + 4,16 = 11,56 \text{ ms}$$

a este tiempo habría que añadir el tiempo de lectura y comprobación del campo identificador del sector, con lo que el tiempo de acceso al campo de datos del sector resulta ser:

$$T_{a,datos} = T_a + T_i = 11,56 + 100 = 111,56 \text{ ms}$$

- P5.14.** Calcular la capacidad máxima de almacenamiento que puede soportar un controlador de disco IDE sabiendo que utiliza 10 bits para determinar el cilindro, 4 bits para el número de cabeza y cada pista se divide en 63 sectores. ¿Cuál sería esta capacidad caso de utilizar una ROM-BIOS<sup>4</sup> mejorada que permite definir el número de cabeza con 8 bits, en lugar de con 4? (Suponer que en un sector se incluyen 512 Bytes de datos del usuario.)

#### SOLUCIÓN

El direccionamiento de un disco se hace físicamente por sectores, y para especificar un sector es necesario proporcionar:

$$\text{Cilindro/pista/cabeza/sector}$$

Con 10 bits podemos direccionar 1.024 cilindros.

Con 4 bits podemos direccionar 16 cabezas (superficies).

<sup>4</sup> Una ROM-BIOS es una ROM que contienen los PC compatibles con programas y parámetros de control de los periféricos estándar de entrada salida.

Con lo que el número máximo de sectores será:

$$N_{\text{sectores}} = 1.024 \times 16 \times 63 = 1.032.192 \text{ sectores}$$

Como cada sector tiene 512 Bytes, la capacidad total será:

$$C_T = N_{\text{sectores}} \cdot C_{\text{sector}} = 1.032.192 \cdot 512 = 500 \text{ MBytes}$$

Si se dedicasen 8 bytes en lugar de 4 para identificar las cabezas, podríamos tener 256 cabezas en lugar de 16, la capacidad se multiplicaría por  $256/16 = 16$ ; es decir:

$$C_T = 500 = 8.256 \text{ MBytes} = 8 \text{ GBytes}$$

## DISCOS RAID

**P5.15.** En una unidad de discos RAID nivel 4, formada por 6 unidades de disco (D0 a D5) y que utiliza como redundancia un bit de paridad (criterio par), en un momento dado se produce una avería de la unidad D3. Suponiendo que la información de los 16 primeros bits de los discos es la que se indica en la Tabla 5.4:

- a) Obtener los cambios que haría el sistema operativo para que el sistema siguiese funcionando correctamente, y los valores de los 16 primeros bits de los discos.

D0	D1	D2	D3	D4	D5
0000 1010 1100 0101 ..... .....	1001 1100 1010 1011 ..... .....	0110 1101 1010 1101 ..... .....	<i>averiado</i>	0011 1111 1010 1001 ..... .....	1111 0010 1000 0111 ..... .....

**Tabla 5.4.** Situación de las unidades de disco en el momento de la avería.

Cinco horas después la unidad se repara, y las unidades de disco, en el momento de volver a poder utilizar el disco D3, contienen la información que se indica en la Tabla 5.5.

D0	D1	D2	D3	D4	D5
1100 1010 0101 1001 ..... .....	1000 1011 0000 0001 ..... .....	0011 0100 0010 1101 ..... .....	<i>reparado</i>	0110 1100 0100 0001 ..... .....	0110 1000 0101 1101 ..... .....

**Tabla 5.5.** Situación de las unidades de disco en el momento de reparar la avería.

- b) Obtener los nuevos cambios y los 16 bits iniciales de los discos que obtendría automáticamente el sistema.

## SOLUCIÓN

- a) Con la información de los discos D0, D1, D2, D4 y D5 se puede reconstruir la información que había en D3. Esta información se grabaría en D5, y el sistema operativo utilizaría D5 como si fuese D3; es decir, en el bus de salida proporcionaría la información de D0, D1, D2, D5, D4.

La información que había en D3 en el momento de producirse la avería, es teniendo en cuenta que el número de unos era par; en otras palabras, obtenemos los siguientes bits:

D0 →	0000 1010	1100 0101
D1 →	1001 1100	1010 1011
D2 →	0110 1101	1010 1101
D4 →	0011 1111	1010 1001
D5 →	1000 0111	1000 0111
Información que había en D3 →	0100 0011	1110 1101

Con lo que la información que se graba en D5 es la que se indica en la siguiente tabla:

DO	D1	D2	D3	D4	D5 con funcionamiento de D3
0000 1010 1100 0101 ..... .....	1001 1100 1010 1011 ..... .....	0110 1101 1010 1101 ..... .....	<i>averiado</i>	0011 1111 1010 1001 ..... .....	0100 0011 1110 1101 ..... .....

- b) Una vez reparado D3 la información de D5 se pasaría a D3, y en D5 se volverían a calcular nuevos bits de paridad.

## CINTAS MAGNÉTICAS

**P5.16.** Una unidad de cinta opera a 800 bpi, con 9 pistas. Los bloques tienen el siguiente formato:

- $N$  Bytes de datos.
- Zona en blanco equivalente a tres líneas, seguida de un carácter de comprobación de redundancias cíclicas (carácter CRCC).
- Zona en blanco de final de bloque. Ocupa el equivalente a 3 líneas. Esta zona también se denomina “blanco EOR”. Es seguida de un carácter LPCC (carácter de comprobación de paridad longitudinal).
- Interbloque (IRG), con  $L_{IRG} = 0,6''$ .

Siendo la longitud del bloque de datos  $N = 512$  Bytes, ¿cuántos bloques y bytes de datos se podrán almacenar en una cinta de 2.400 pies?

### SOLUCIÓN

La estructura de los bloques es la siguiente:

Datos	Blanco	CRCC	EOR	LPCC	IRG
512 B	3 B	1 B	3 B	1 B	0,6''

La densidad es igual a:

$$800 \text{ bpi} = 314,9 \text{ b/cm.} \quad (1\text{i} = 2,54 \text{ cm})$$

- L. bloque =  $(512 + 3 + 1 + 3 + 1)\text{B}/314,9 \text{ b/cm} + 0,6'' = 3,174 \text{ cm}$ .
- N.º bloques = L.cinta/L.bloque.  
N.º bloques =  $2.400 \times 30,48 \text{ cm}/3,174 \text{ cm/bl.} = 23.047 \text{ bloques}$ .
- Bytes datos = N.º bloques  $\times$  N.º B/bloque.  
Bytes datos =  $23.047 \text{ bl.} \times 512 \text{ B/bl.} = 11.800.196 \text{ B} = 11,25 \text{ MB}$ .

**P5.17.** La separación de archivos en la unidad de cinta descrita en el ejercicio anterior se realiza según la siguiente estructura:

- Zona en blanco de final de fichero de 3'' (EOF GAP).
- Marca de Fin de Fichero (EOF): 1 Byte.
- EOR GAP (blanco de final de registro): tres líneas en blanco.
- Carácter LPCC (para comprobar paridad longitudinal).
- IRG, con  $L_{IRG} = 0,6''$ .

¿Cuántos ficheros de 64 KBytes cabrán en una cinta de 2.400 pies grabada en la unidad indicada?

### SOLUCIÓN

La estructura es la siguiente:

N Bloques	EOF GAP	EOF	EOR GAP	LPCC	IRG
N	3''	1 B	3 B	1 B	0,6''

- N.º bloques (64 KB) =  $64 \times 1.024 \text{ B}/512 \text{ B/bloque} = 128 \text{ bloques}$ .
- $L_{TOTAL} = L.\text{bloques} + L.\text{Fin Fich.}$



$$L_{\text{Fin Fich.}} = (1 + 3 + 1)B/314,9 \text{ b/cm} + (3 + 0,6)'' = 9,159 \text{ cm.}$$

$$L_{\text{TOTAL}} = 128 \times 3,174 \text{ cm} + 9,159 \text{ cm} = 415,431 \text{ cm} = 13,62 \text{ pies.}$$

$$3. \quad N.^{\circ} \text{ ficheros (64 KB)} = 2.400 \text{ pies}/13,62 \text{ pies} = 176 \text{ ficheros.}$$

**P5.18.** Una unidad de cinta utiliza el sistema de grabación PE, y opera a 1.600 bpi, con 9 pistas y  $v_t = 75$  ips. Los bloques tienen la siguiente estructura:

- 41 Bytes delimitadores del comienzo del bloque (40 todos 0, y 1 todo unos).
- $N$  Bytes de datos (especificados por la longitud del bloque de datos).
- 41 Bytes delimitadores del final del bloque (1 todo 1, 40 todos 0) Siendo la longitud del interbloque  $L_{\text{IRG}} = 0,6''$  y  $N = 1.024$ , ¿cuántos bloques y bytes de datos se podrán almacenar en una cinta de 2.400 pies?

### SOLUCIÓN

En primer lugar, pasamos la densidad a b/cm:

$$1.600 \text{ bpi} = 1.600 \text{ b}/2,54 \text{ cm} = 629,92 \text{ b/cm.}$$

La estructura de los bloques es la siguiente:

Inicio	Datos	Final	IRG
41 B	1.024 B	41 B	0,6''

a) Número de bloques:

$$a1) \quad L. \text{ bloque} = [(41 + 1.024 + 41)B/629,92 \text{ b/cm}] + 0,6''.$$

$$L. \text{ bloque} = 3,28 \text{ cm} (1'' = 2,54 \text{ cm})$$

$$a2) \quad N.^{\circ} \text{ bloques} = (L. \text{ cinta}/L. \text{ bloque}) = (2.400 \text{ pies}/3,28 \text{ cm}).$$

$$N.^{\circ} \text{ bloques} = 22.302 (1 \text{ pie} = 30,48 \text{ cm})$$

b) Bytes de datos:

$$\text{Bytes de datos} = 22.302 \text{ bloques} \times 1.024 \text{ B/bloque.}$$

$$\text{Bytes de datos} = 22.837.697 \text{ Bytes} = 21,8 \text{ MBytes.}$$

**P5.19.** La unidad del ejercicio anterior separa los ficheros con marcas “fin de fichero” también denominadas marcas EOF (*End of File*), y utilizando el siguiente formato:

3.º	41 Bytes delimitadores del final del último bloque del fichero $M$ .
1.º	IRG, con $L_{\text{IRG}} = 0,6''$ .
2.º	Zona de 3'' en blanco, de fin de fichero (análoga a un IRG).
3.º	40 Bytes con marcas EOF.
4.º	IRG, con $L_{\text{IRG}} = 0,6''$ .
1.º	41 Bytes delimitadores del comienzo del primer bloque del fichero $M + 1$ .

Calcular la longitud de cinta que ocuparía un fichero de  $C = 64$  KBytes de capacidad.

### SOLUCIÓN

La estructura es la siguiente:

Inicio	Datos	Final	IRG	blanco	EOF	IRG
←			→	←		→
41B	1.024B	41B	0,6''	3''	40B	0,6''
Último bloque				Fin fichero		

1.  $L. \text{ Bloque} = L. \text{ Bloque anterior} = 3,28 \text{ cm.}$
2.  $L. \text{ Fin Fich.} = (40 \text{ B}/629,92 \text{ cm}) + (3'' + 0,6'').$   
 $L. \text{ Fin Fich.} = 9,2 \text{ cm.}$

3. L. Fichero = L. Bloques (64 KB) + L. Fin Fich.  
 L. Bloques =  $(64 \times 1024 \text{ B}/1.024 \text{ B/bloque}) = 64 \text{ bloques}$ .  
 L. Fichero =  $(64 \text{ bloques} \times 3,28 \text{ cm/bloque}) + 9,2 \text{ cm}$ .  
 L. Fichero = 219,12 cm.

**P5.20.** Estimar la capacidad máxima y velocidad de transferencia de datos de una cinta de 2.400 pies grabada a 1.600 b/i, y que es leída por una unidad que funciona a una velocidad lineal de 75 i/s.

#### SOLUCIÓN

La capacidad será:

$$C = 2.400 \text{ pies} \cdot 1.600 \frac{\text{bits}}{\text{pulgada}} = (2.400 \cdot 12) \text{ pulgadas} \cdot 1.600 \frac{\text{bits}}{\text{pulgada}} \approx 44 \text{ Mbits}$$

Es decir, puede almacenar en total unas 44 Megalíneas o MB.

Como la velocidad lineal es 75i/s en un segundo se leen:

$$1.600 \frac{\text{bits}}{\text{pulgada}} \cdot 75 \frac{\text{pulgada}}{\text{s}} = 120.000 \text{ bits longitudinales (líneas)}$$

Como en cada línea se almacena un byte, la velocidad de transferencia será:

$$v_{\text{trans}} = \frac{120.000}{1.024} \approx 117 \text{ KB}$$

### DISPOSITIVOS ÓPTICOS

**P5.21.** Calcular el número de bits de canal que se leen por segundo y la velocidad de transferencia media de datos de usuario, en una unidad de CD-ROM estándar, sabiendo que la velocidad de lectura es de 75 sectores por segundo. ¿Qué capacidad de datos cabría en un CD-ROM de 74 minutos?

#### SOLUCIÓN

Primero se puede calcular el número de bits de canal que contiene cada trama. Una trama está compuesta por:

$$(24 \text{ Bytes/trama} \times 17 \text{ bits de canal/Byte}) + (27 \text{ bits patrón de sincronización}) + (17 \text{ bits/símbolo} \times 1 \text{ símbolo de control}) + (17 \text{ bits/símbolo} \times 8 \text{ símbolos de corrección de errores}) = 588 \text{ bits/trama}$$

Un sector contiene:

$$588 \text{ bits/trama} \times 98 \text{ tramas/sector} = 57.624 \text{ bits de canal}$$

Es decir, se leen:

$$57.624 \times 75 = 4.321.800 \text{ bits/s}$$

Como en cada sector hay 2 KB datos de usuario, la velocidad de transferencia será:

$$2 \text{ KB/sect} \times 75 \text{ sect/seg} = 150 \text{ KB/s}$$

En 74 minutos se pueden leer:

$$74 \times 60 \times 75 = 333.000 \text{ sectores}$$

Es decir:

$$333.000 \times 2 \text{ KB} = 666.000 \text{ KBytes}$$

con lo que la capacidad máxima de la unidad es de 650 MBytes.

**P5.22.** Una unidad CD-ROM estándar tiene una velocidad de lectura de 75 sectores/segundo, y una capacidad para almacenar 74 minutos de tiempo. Calcular la longitud total del canal (pistas) de un CD-ROM. (Recuérdese que 1 sector contiene 2.352 Bytes, de los cuales 2 KB son de datos del usuario, y que la densidad de grabación de las pistas es de 0,6 mm/bit de canal.)

## SOLUCIÓN

El número total de sectores del CD serán:

$$C_{CD} = 75 \text{ sectores/s} \times 74 \text{ minutos} \times 60 \text{ s/minuto} = 333.000 \text{ sectores}$$

Con lo que la capacidad en bytes será:

$$C_{CD} = 2.352 \text{ Bytes/sector} \times 333.000 \text{ sectores} = 7,85 \times 10^8 \text{ Bytes}$$

Teniendo en cuenta que cada byte (8 bits) se almacena en un código de canal de 14 bits (EFM), la capacidad en bits de canal será:

$$C_{CD} = 7,85 \times 10^8 \text{ Bytes} \times 14 \text{ bits/byte} = 1,09 \times 10^{10} \text{ bits de canal}$$

Teniendo en cuenta la densidad, la longitud total del canal será:

$$L = 1,09 \times 10^{10} \text{ bits de canal} \times 0,6 \mu\text{m/bit de canal} = 6,6 \text{ km}$$

Por tanto, la longitud total del canal de un CD-ROM estándar es de 6,6 km.



## Problemas propuestos

## CONCEPTOS BÁSICOS Y MEDIDAS DE PRESTACIONES

**P5.23.** Estimar el ancho de banda de un sistema de memoria que tiene un tiempo de ciclo 30 ns, y en el que se accede y transfieren palabras de 32 bits.

**P5.24.** El bus de direcciones de una memoria con acceso por bytes contiene 32 líneas, y sus palabras son de 64 bits:

- a) ¿Cuál es su capacidad en megapalabras?
- b) ¿Cuál es la dirección que debe darse a la memoria para acceder a la palabra cuya dirección es AB7 54CF.
- c) El byte de dirección CDA7 325A, ¿a qué palabra corresponde?

**P5.25.** En una memoria con longitud de palabra de 64 bits se desea almacenar en la palabra de posición C37 43F2 el texto "CODE". Indicar las direcciones de memoria donde se almacena cada uno de los cuatro caracteres sabiendo que el acceso se realiza por bytes y que se utiliza el criterio del extremo menor.

**P5.26.** Suponga un sistema de memoria que dispone de una caché que utiliza bloques de datos de 16 palabras, y una memoria principal (DRAM) que consume 6 ciclos de reloj para acceder a una palabra y 2 ciclos para accesos consecutivos. Estimar el tiempo que tarda en transferirse un bloque de datos suponiendo que la frecuencia del reloj es de 1 GHz.

**P5.27.** Para el sistema de memoria anterior, estimar el tiempo que tarda en transferirse el mismo bloque suponiendo que la memoria es entrelazada con 8 módulos.

**P5.28.** Se dispone de un sistema de memoria entrelazado, cuyas direcciones se interpretan de la siguiente manera: los 28 primeros bits representan la dirección dentro de cada módulo, y los 4 bits menos significativos el módulo. Por otra parte cada

módulo está organizado en palabras de 32 bits y acceso por bytes. Determinar:

- a) La capacidad total de la memoria en bytes.
- b) El módulo y posición dentro de él donde se encuentra el byte de dirección A354 BC76.
- c) La palabra de la que forma parte el byte referenciado anteriormente.

## JERARQUÍA DE MEMORIA

**P5.29.** El sistema de memoria de un computador contiene una caché con tiempo de acceso de 6 ns, una memoria principal con tiempo de acceso de 100 ns y una unidad de disco donde se gestiona la memoria virtual con tiempo de acceso de 12 ms. Después de analizar el comportamiento de la memoria se concluye que los porcentajes de aciertos en la memoria principal es del 99 por 100 y el tiempo medio de acceso a la caché de 15 microsegundos. Estimar el porcentaje de aciertos de la caché.

**P5.30.** Un sistema de memoria dispone de caché (SRAM) con un tiempo de acceso de 4 ns, memoria principal, con tiempo de acceso de 50 ns, y disco duro con tiempo de acceso de 7 ms. Realizar una gráfica mostrando la variación del tiempo medio de acceso a la caché con el porcentaje de aciertos en los dos niveles superiores, en los siguientes supuestos:

- a) Porcentaje de aciertos en caché y memoria principal del 85 por 100.
- b) Porcentaje de aciertos en caché y memoria principal del 90 por 100.
- c) Porcentaje de aciertos en caché y memoria principal del 95 por 100.
- d) Porcentaje de aciertos en caché y memoria principal del 99 por 100.

**P5.31.** Suponga un sistema que no dispone de memoria virtual, y con tan sólo dos niveles de jerarquía. Si los tiempos de acceso son 5 y 50 ns, respectivamente:

- a) ¿Qué tasa de aciertos debería haber para conseguir un tiempo medio de acceso de 8 ns?
- b) Sin incluir más niveles en la jerarquía de memoria, y sin modificar los algoritmos de reemplazo, ¿cómo se podría mejorar la tasa de aciertos?

### DISCOS MAGNÉTICOS

**P5.32.** Un disco duro de 4 GB gira a una velocidad angular constante de 7.200 rpm y está dividido en 1.024 pistas y 256 sectores/pista. La unidad de disco está conectada a la memoria principal mediante un bus de 16 hilos. Determinar:

- a) Tamaño, en bytes, del sector.
- b) El ancho de banda del bus para que el sistema pueda transferir una pista completa en cada giro del disco.
- c) Ancho de banda mínimo de cada línea que compone el bus.
- d) Tiempo necesario para transferir 512 KB desde el disco a la memoria principal.

**P5.33.** Hacer una estimación del tiempo medio que se tarda en leer o escribir un sector de 512 Bytes en un disco magnéti-

co, sabiendo que el tiempo medio de búsqueda de la pista es de 20 ms, la velocidad de rotación es de 3.600 rpm, la velocidad de transferencia es de 1 MB/s y el tiempo invertido por el controlador (comprobación de errores, etc.) es de 2 ms. (*Sugerencia:* Suponer que los tiempos anteriores no se solapan y el tiempo a calcular es el transcurrido desde que el disco recibe la orden de leer hasta que la información llega al DMA o CPU.)

**P5.34.** Calcular la capacidad máxima de almacenamiento que puede soportar un controlador de disco ATA sabiendo que utiliza 10 bits para determinar el cilindro, 4 bits para el número de cabeza y cada pista se divide en 63 sectores (suponer que en un sector se incluyen 512 Bytes de datos del usuario).

**P5.35.** Calcular la capacidad máxima de almacenamiento que puede soportar un controlador de disco E-IDE sabiendo que utiliza 16 bits para determinar el cilindro, 8 bits para el número de cabeza y cada superficie se divide en 63 sectores (suponer que en un sector se incluyen 512 Bytes de datos del usuario).

**P5.36.** Obtener la capacidad de almacenamiento de una unidad de disco E-IDE de 3 ½ pulgadas que contiene 2 platos, con todas sus superficies grabables (*Nota:* Utilice los datos que necesite del problema anterior.)



# Periféricos de E/S

En este capítulo se estudian los dispositivos (periféricos de entrada y salida) a través de los cuales el computador intercambia información con el mundo exterior. Como es bien conocido existen muy diversas formas de dar información a un computador, como puede ser a través del lenguaje oral (reconocedores del habla), seleccionando iconos u opciones de un menú (ratón, pantallas sensibles al tacto y lápices), o por medio del lenguaje escrito (teclado). Por otra parte, existen dispositivos de salida inversos a los de entrada, que transforman la información binaria interna del computador en textos o imágenes o sonidos producidos por medio de pantallas, impresoras o altavoces. El presente capítulo pretende dar una visión general acerca de ellos.

## 6.1. Definición y tipos

Se denominan **periféricos** a las unidades o dispositivos a través de los cuales el procesador se comunica con el mundo exterior, y a los sistemas que almacenan o archivan información, sirviendo de **memoria auxiliar** de la memoria interna. Es decir, un periférico es cualquier dispositivo del computador que no sea el procesador o la memoria principal.

El computador es una máquina que no tendría sentido si no se comunicase con el exterior, es decir, si careciese de periféricos. Debe disponer de:

- **Unidades de entrada**, a través de las cuales poder introducir los programas que queramos que se ejecuten, y los datos correspondientes. Estas unidades transforman la información externa según un código de E/S (ASCII, por ejemplo). Así el procesador/memoria recibe dicha información adecuadamente preparada (en binario).
- **Unidades de salida**, con las que el computador nos da los resultados de los programas. Éstas efectúan el proceso inverso: la información binaria que llega del procesador se transforma de acuerdo con el código de E/S en caracteres escritos o en formas inteligibles por los usuarios.
- **Memoria externa**, que sirve de complemento a la memoria interna y para almacenar tanto programas como datos de forma permanente. La segunda parte del Capítulo 5 se dedicó a este tipo de periféricos, por lo que no los vamos a considerar en el presente.

Cada periférico está constituido por dos partes claramente diferenciadas en cuanto a su misión y funcionamiento: una parte mecánica y otra electrónica.

- La **parte mecánica** está formada básicamente por dispositivos electromecánicos (conmutadores manuales, relés, motores, electroimanes, servomecanismos, etc.), controlados por los elementos electrónicos.
- La **parte electrónica o controlador del periférico** se encarga de interpretar las órdenes que le llegan del procesador para la recepción o transmisión de datos, dependiendo de que se trate de un periférico de entrada o salida, respectivamente, y de generar las señales de control para activar los elementos electromecánicos del periférico que producen o captan los datos en el soporte de información correspondiente (pantalla, impresora, disco magnético, etc.). Por lo general contiene una **memoria temporal o buffer** que almacena los datos que se intercambian entre el periférico y las unidades centrales del computador. También muchos periféricos contienen **convertidores A/D**, que transforman señales analógicas en digitales, o **convertidores D/A**, que realizan la conversión contraria: señales digitales en analógicas (ver Sección 6.9).

En las Tablas 6.1 a 6.3 se enumeran distintos tipos de periféricos. En las secciones siguientes se describen los más relevantes.

<ul style="list-style-type: none"> <li>• Lectora de tarjetas perforadas<sup>1</sup>.</li> <li>• Lectora de cinta de papel perforada<sup>1</sup>.</li> <li>• Teclado.</li> <li>• Detectores ópticos:               <ul style="list-style-type: none"> <li>— De marcas.</li> <li>— De barras impresas.</li> <li>— De caracteres impresos.</li> <li>— De caracteres manuscritos.</li> <li>— Escáner de imágenes.</li> <li>— Cámara de fotos digital.</li> <li>— Cámara de vídeo.</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>• Tarjeta de edición de vídeo.</li> <li>• Lectora de banda magnética.</li> <li>• Detector de caracteres magnetizables.</li> <li>• Sensores-convertidores analógico/digital (Interfaz industrial de entrada).</li> <li>• Unidad de reconocimiento de la voz.</li> <li>• Lápices óptico, electrostático y de presión.</li> <li>• Pantalla sensible al tacto.</li> <li>• Palanca de control para juegos (<i>joystick</i>).</li> <li>• Digitalizador o tableta gráfica.</li> <li>• Ratón.</li> </ul>
--	---

<sup>1</sup> Dispositivo obsoleto.

**Tabla 6.1.** Ejemplos de unidades de entrada.

<ul style="list-style-type: none"> <li>• Unidad perforadora de tarjetas<sup>1</sup>.</li> <li>• Unidad perforadora de cinta de papel<sup>1</sup>.</li> <li>• Monitores de visualización:               <ul style="list-style-type: none"> <li>— Pantallas de rayos catódicos (CRT).</li> <li>— Pantallas planas.</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>• Impresora.</li> <li>• Conversor digital/analógico-efector (Interfaz industrial de salida).</li> <li>• Sintetizador de voz.</li> <li>• Visualizadores (<i>displays</i>).</li> <li>• Registrador gráfico (<i>plotter</i>).</li> </ul>
---	--

<sup>1</sup> Dispositivo obsoleto.

**Tabla 6.2.** Ejemplos de unidades de salida.

<ul style="list-style-type: none"> <li>• <b>Dispositivos magnéticos:</b> <ul style="list-style-type: none"> <li>— Tambor magnético<sup>1</sup>.</li> <li>— Disco magnético.                   <ul style="list-style-type: none"> <li>■ Discos de cabezas fijas.</li> <li>■ Paquetes de discos.</li> <li>■ Discos cartucho.</li> <li>■ Discos Winchester.</li> <li>■ Disquetes.</li> </ul> </li> <li>— Cinta magnética.                   <ul style="list-style-type: none"> <li>■ Carrete<sup>1</sup>.</li> <li>■ Cartucho.</li> <li>■ ExaByte.</li> <li>■ DAT, etc.</li> </ul> </li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>• <b>Dispositivos ópticos:</b> <ul style="list-style-type: none"> <li>— CD (Disco compacto).                   <ul style="list-style-type: none"> <li>■ CD-ROM (sólo lectura).</li> <li>■ CD-R (grabable).</li> <li>■ CD-RW (regrabable).</li> </ul> </li> <li>— DVD (Disco digital versátil).                   <ul style="list-style-type: none"> <li>■ DVD-ROM (sólo lectura).</li> <li>■ DVD-R (grabables).</li> <li>■ DVD-RW (regrabables).</li> <li>■ DVD-RAM.</li> </ul> </li> </ul> </li> <li>• <b>Dispositivos magneto-ópticos:</b> <ul style="list-style-type: none"> <li>— Disco magneto-óptico (MO).</li> </ul> </li> </ul>
---	--

<sup>1</sup> Dispositivo obsoleto.

**Tabla 6.3.** Ejemplos de unidades de memoria externa (estudiados en el Capítulo 5).

## 6.2. Teclados

En un teclado, al pulsar una tecla se cierra un conmutador que hay en el interior del teclado, esto hace que unos circuitos del **controlador del teclado** generen el código correspondiente al carácter seleccionado (ASCII, por ejemplo), almacenándolo en el *buffer* del teclado. El controlador del teclado envía una petición de interrupción al procesador para que, cuando sea aceptada, el programa **gestor del teclado** lea del *buffer* el código tecleado llevándolo a un registro del procesador. Normalmente el programa gestor del teclado hace un *eco* del carácter pulsado visualizándolo en el monitor.

Existen dos tipos de teclados: mecánicos y de membrana.

- En un **teclado mecánico** cada tecla es un pulsador que al ser oprimido provoca un contacto en una terminación metálica del circuito impreso del propio teclado, cerrando así un circuito específico del carácter. Al cesar la opresión de la tecla, un muelle la levanta llevándola a su posición inicial. El controlador del teclado identifica el carácter seleccionado y genera e introduce en el *buffer* el código correspondiente.
- En los **teclados de membrana** se utilizan tres capas, que de arriba a abajo son las siguientes:
  - Capa que contiene los dibujos de los caracteres y en la parte inferior unas placas conductoras.
  - Capa intermedia de goma con agujeros justo debajo de cada una de las teclas.
  - Capa conductora con pequeñas protuberancias coincidiendo con los agujeros de la capa de goma.

Al pulsar una tecla se oprime la goma entrando en contacto la plaquita conductora con la protuberancia de la capa inferior, cerrándose así un circuito eléctrico, cuya corriente y posición es detectada por el controlador del teclado. Estos teclados son más económicos que los mecánicos, pero su calidad de funcionamiento y tiempo de vida son menores, al deformarse la goma con el uso.

## 6.3. Entradas manuales directas

Como alternativa a los teclados existen diversos dispositivos que se adaptan a la mano del usuario y permiten seleccionar directamente opciones de menú, iconos o dibujar objetos o escribir directamente en el periférico. Este es el caso del ratón, las pantallas sensibles al tacto, los lápices y la palanca de control para juegos, dispositivos que a continuación se describen brevemente.

### Pantallas y paneles sensibles a la presión

Son pantallas o paneles que pueden detectar las coordenadas ( $x$ ,  $y$ ) de la zona de la propia pantalla donde se aplica una presión (por ejemplo, con un dedo).

La pantalla o panel puede contener embebida una red rectangular o matriz de hilos conductores muy finos, paralelos a los ejes  $x$  e  $y$  de la imagen. Al presionar se hace que entren en contacto determinados hilos: uno  $x$  con otro  $y$ . El contacto eléctrico producido puede detectarse fácilmente con circuitos electrónicos, identificándose así la posición del punto presionado. Otras pantallas de menor resolución funcionan mediante retículas de células fotoeléctricas. Las de mayor resolución utilizan una gran cantidad de diminutas fibras ópticas integradas en su interior. En definitiva, estos dispositivos se fundamentan en la localización de un punto de una superficie donde se ha producido un cambio en sus propiedades eléctricas u ópticas.

Las pantallas sensibles a la presión constituyen un sistema muy sencillo e intuitivo para dar entradas o elegir opciones sin utilizar el teclado. Puede ser útil para cajas registradoras, terminales pun-



to de venta<sup>1</sup>, gasolineras, etc. Este tipo de pantallas se utiliza en PDA y en sitios donde el equipamiento informático debe ser lo más compacto y robusto posible, para evitar averías y deterioro por las condiciones ambientales. Los paneles o almohadillas táctiles (*touchpad*) se utilizan en equipos portátiles, y en agendas personales (PDA) combinadas con pequeñas pantallas de cristal líquido, para operaciones de entrada y salida.

### Lápices óptico, electrostático y de presión

Un **lápiz óptico** físicamente tiene la forma de una pluma o lápiz, de uno de cuyos extremos sale un cable con un conector para unirlo a un monitor de pantalla o a una tableta especial. En el otro extremo hay una abertura o ventana por la que puede pasar la radiación luminosa de la pantalla. Esta radiación es enfocada internamente o llevada por una fibra óptica a la superficie de un fotodetector. La señal eléctrica generada en él, después de ser amplificada, se transmite a los circuitos de control del monitor. El lápiz contiene un pulsador, transmitiéndose información hacia el monitor sólo en el caso de estar presionado. Si desde el programa en ejecución se activa el lápiz óptico, al colocar la ventana de éste frente a un punto  $(x_0, y_0)$  de la pantalla, en el instante  $t_0$ , al incidir el haz de electrones que barre la pantalla en el punto señalado se genera un pulso eléctrico. A partir del valor  $t_0$  los circuitos de control pueden obtener las coordenadas  $(x_0, y_0)$  donde estaba el haz de electrones en  $t = t_0$ , o lo que es lo mismo, las coordenadas del lugar donde apuntaba el lápiz. El programa en curso podrá ejecutar un bloque de instrucciones o subrutina en función de las coordenadas del punto marcado por el usuario (el programa obtiene  $x_0, y_0$ ).

Además de los lápices ópticos existen otros de los siguientes tipos:

- **Lápices electrostáticos:** generan descargas electrostáticas, y son detectadas de forma similar a la indicada, pero la superficie sensible de la pantalla, en lugar de contener detectores de luz, contiene detectores electrostáticos.
- **Lápices de presión:** se utilizan en conjunción con paneles sensibles a la presión, similares a los vistos en la sección anterior.

Los lápices se utilizan frecuentemente para dar entradas o seleccionar opciones de menús en los computadores móviles (PDA, por ejemplo), estando la interfaz contenida en el controlador de vídeo.

### Palanca de control para juegos

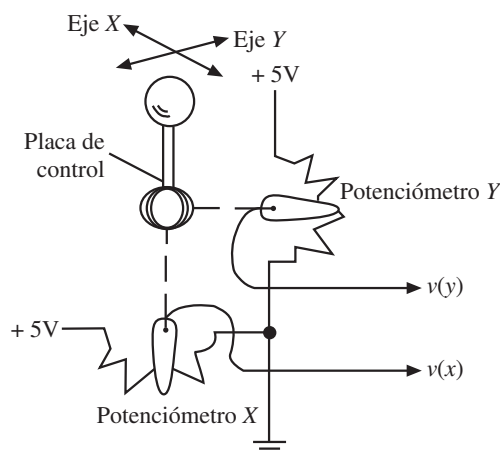
Una **palanca de control para juegos** (*joystick*) está constituida por una cajita de la que sale una palanca corta o mando móvil. El usuario puede actuar sobre el extremo de la palanca exterior a la caja, y a cada posición de ella le corresponde en la pantalla un punto de coordenadas  $(x, y)$ . La cajita o varilla dispone de uno o varios pulsadores que deben ser presionados para interactuar en tiempo real entre el programa y el usuario.

Con las instrucciones adecuadas el programa puede captar el estado de los pulsadores (presionado o no) y las coordenadas  $(x, y)$  correspondientes a la posición de la varilla. De esta manera puede lograrse que el cursor, o una figura o motivo, se desplacen en la pantalla siguiendo las posiciones correspondientes al movimiento de la palanca. Esta función es muy útil en videojuegos y aplicaciones gráficas.

La cajita contiene en su interior dos potenciómetros circulares y perpendiculares, situados en la superficie interior de un casquete esférico, cuyos ejes móviles están unidos solidariamente a la palanca de control. En la Figura 6.1 se esquematiza el circuito; con él se obtienen en todo momento dos

---

<sup>1</sup> Un **terminal punto (o puesto) de ventas** o **terminal POS** (*Point-Of-Sale*) es una caja registradora de un centro comercial que en realidad es un terminal interactivo de una red local, y que dispone de lector de barras impresas, lector de tarjetas magnéticas, etc., de forma que el control de existencias, de ventas y de caja se hace de forma centralizada.



**Figura 6.1.** Esquema eléctrico del interior de una palanca de control.

tensiones,  $v(x)$  y  $v(y)$ , proporcionales a los valores de  $x$  e  $y$ , respectivamente. Estos valores analógicos pueden ser transformados en digitales por un conversor A/D. Otros modelos contienen diversos microinterruptores distribuidos perpendicularmente en cuatro direcciones, que se cierran en función de la posición de la palanca.

### Ratones

Un **ratón** es un dispositivo de entrada que sirve para introducir información gráfica o seleccionar coordenadas  $(x, y)$  de una pantalla. Los usos más habituales del ratón se encuentran en la selección de iconos u opciones de menús, selección y arrastre de ventanas y otros objetos, y dibujo manual de líneas u otros elementos geométricos. Al desplazar el ratón manualmente sobre una alfombrilla o superficie plana, se reproducen los movimientos en el cursor de la pantalla. Por lo general, cada vez que se desplaza el ratón una distancia mínima (por ejemplo, 0,01") se envía (usualmente en forma serie) una secuencia compuesta de 3 Bytes, indicando cada uno de ellos: desplazamiento X, desplazamiento Y y estado de los pulsadores. Los ratones detectan, por tanto, movimientos relativos. Puede considerarse como uno de los dispositivos de entrada más populares. Existen tres tipos básicos de ratones:

- **Ratón mecánico.** Está internamente constituido por una bola que puede girar libremente, y se acciona haciéndola rodar sobre una superficie. La bola es solidaria con dos rodamientos o sensores perpendiculares entre sí, cuyos desplazamientos se detectan eléctricamente en forma similar a los de una palanca de control para juegos (Figura 6.1). Actualmente está en desuso.
- **Ratón opto-mecánico.** En este caso la bola hace girar dos ruedas dentadas perpendiculares. Cada rueda dentada se encuentra entre un diodo emisor de luz y un fotodetector, de forma tal que el desplazamiento (ángulo de giro) de la rueda se puede detectar con los impulsos de luz que recibe el detector (cuando en el camino óptico se interponen los dientes de la rueda). Los valores recibidos de los dos fotodetectores son llevados a sendos contadores binarios (X e Y), que se incrementan o decrementan conforme sea el sentido, horizontal o vertical, del movimiento de la bola del ratón. Esta información es transmitida por unos cables flexibles ("cola del ratón") al computador, y el programa gestor del ratón puede determinar la distancia, dirección y sentido del desplazamiento, desde que se inició el último movimiento.
- **Ratón óptico.** Contiene un emisor de luz y un detector de la luz reflejada. El ratón se hace mover sobre una base que tiene dibujada una retícula de dimensiones prefijadas. La detección de los movimientos relativos se efectúa midiendo la intensidad de luz reflejada que variará al pasar el fotodetector por encima de las líneas de la retícula. Otros ratones más modernos no ne-

cesitan almohadilla con retícula pudiéndose utilizar sobre cualquier superficie; disponen de una pequeña retina artificial que capta unas 1.500 imágenes por segundo, comparando los cambios que se producen en imágenes sucesivas se determinan con precisión los desplazamientos realizados. Los ratones ópticos, por tanto, no tienen elementos móviles, y además tienen mayor precisión que los mecánicos.

Con los computadores portátiles o en situaciones en las que no hay espacio suficiente para desplazar el ratón, suelen utilizarse **ratones estacionarios** (*trackball*). Son como ratones mecánicos pero que se utilizan con la bola hacia arriba, de forma que ésta se desplaza con el dedo pulgar, y no haciéndola rodar por una superficie. De hecho hay ratones que pueden utilizarse en las dos formas indicadas.

### EJEMPLO 6.1

Un ratón<sup>2</sup> óptico inalámbrico puede tener las siguientes características:

- Base USB o PS/2 que conecta por radio frecuencia (RF) con el ratón en un radio de acción de 2 metros.
- Resolución del sensor óptico: 800 dpi.
- Velocidad máxima de captura: 4,7 Megapíxeles/s.
- Funcionamiento correcto hasta una aceleración de 10 g.
- Funcionamiento correcto hasta una velocidad de 1 m/s.

## 6.4. Detectores ópticos

Hay ciertos documentos o productos que se utilizan en la vida ordinaria en gran cantidad y que pueden ser controlados por computador, introduciendo con rapidez y sin error sus características sin necesidad de teclear el código o información que los identifica. Esto es así porque en su superficie (o envoltorio) llevan impresos caracteres, barras o marcas predefinidas, que pueden ser detectados por sistemas especiales (dispositivos de captura directa de datos) normalmente de tipo optoelectrónico o magnético. Como ejemplos de los productos y documentos citados anteriormente están los talones o cheques bancarios, artículos comerciales, quinielas, impresos para corrección exámenes tipo test de elección múltiple, etc.

En la mayoría de estos sistemas existe un conjunto de caracteres o formas (**patrones**) predefinidos. En el producto, los datos de identificación se imprimen con los patrones establecidos. Los lectores ópticos suelen contener una fuente de luz que ilumina intensamente el dato a leer, un sistema óptico de ampliación de la imagen y los elementos necesarios para “barrer” el carácter o descomponer la imagen en las celdillas o zonas de identificación de patrones. Con dispositivos optoelectrónicos se detecta por reflexión el atributo (nivel de gris o niveles de tres colores básicos) de cada zona de identificación. A cada píxel, carácter o forma, por tanto, se le hace corresponder una secuencia ordenada de ceros y unos. El dispositivo de entrada compara esta secuencia con la de los patrones (que tiene grabadas internamente), identificando la forma en cuestión. En el supuesto de no identificarse el dato, se procede a una nueva lectura y se da una señal de error para que el operario de la unidad de lectura conozca esta eventualidad. Otros dispositivos ópticos son los **escáneres**, que se limitan a captar y digitalizar imágenes grabadas en papel, representándolas internamente tal y como se explicó en la Sección 2.3, sin realizar ningún reconocimiento de formas, cosa que se puede realizar posteriormen-

<sup>2</sup> Logitech® MX™700.

te con programas especiales. Los dispositivos de captura directa de datos más comúnmente utilizados son los que se indican a continuación.

### Detectores de marcas

Los **lectores ópticos de marcas** u **OMR** (*Optical Mark Reader*) son sistemas que aceptan información escrita a mano y la transforman en datos binarios inteligibles por el computador. El usuario se limita a marcar con su lápiz o pluma ciertas áreas o posiciones preestablecidas del documento, que representan posibles opciones de determinados eventos o preguntas. Estos documentos pueden ser leídos posteriormente, a gran velocidad, por el computador con un lector óptico de marcas. Éste detecta las zonas marcadas.

### Detectores de barras impresas

En el momento de fabricar un producto se imprime en su envoltorio una etiqueta con información sobre el mismo según un código formado por un conjunto de barras separadas por zonas en blanco. La forma de codificar cada dígito decimal consiste en variar el grosor relativo de las barras negras y blancas adyacentes. Existen varios códigos normalizados, siendo uno de los más utilizados en España el **EAN** (*European Article Numbering*). Con estas marcas o etiquetas se pueden, por ejemplo, controlar fácilmente por computador las existencias y ventas de una determinada empresa. El lector óptico suele formar parte de terminales punto de ventas.

### EJEMPLO 6.2

Según el código EAN-13 cada producto se marca con 13 dígitos, en el orden y con el significado que se da a continuación (Figura 6.2):

- 2 dígitos: código del estado donde se fabricó el producto (España: 84).
- 5 dígitos: código de la empresa fabricante.
- 5 dígitos: código del producto.
- 1 dígito: de verificación o autocorprobación de error.



**Figura 6.2.** Código de barras de productos alimenticios fabricados en España.

El grosor de las barras negras y espacios blancos es simple, doble o triple de un cierto valor (0,33 mm) denominado módulo. Para cada dígito se reservan 7 módulos (es decir, 2,31 mm) donde debe haber dos barras y dos espacios (cada espacio es un conjunto de módulos consecutivos en blanco). Adicionalmente, se incluyen tres separadores: dos laterales que delimitan el principio y fin del código y se componen de sólo tres módulos (negro, blanco y negro), y uno central que separa los códigos de producto y de fabricante y se compone de 5 módulos (blanco, negro, blanco, negro, blanco). Se suelen utilizar simultáneamente tres juegos (A, B y C) de códigos. El que para un dígito concreto se utilice uno u otro juego depende de su posición dentro del código global, o del primer dígito. En la Tabla 6.4 se representa el código EAN (cada bit representa un módulo), siendo el 1 un módulo negro y el cero un módulo blanco. Las barras de los extremos son especiales y distintas, de forma que el lector puede reorientar automáticamente la etiqueta para lograr una lectura correcta.

	Juego A	Juego B	Juego C
0	0001101	0100111	1110010
1	0011001	0110011	1100110
2	0010011	0011011	1101100
3	0111101	0100001	1000010
4	0100011	0011101	1011100
5	0110001	0111001	1001110
6	0101111	0000101	1000100
7	0111011	0010001	1000100
8	0110111	0001001	1001000
9	0001011	0010111	1110100

**Tabla 6.4.** Códigos de barra EAN (cada bit representa un módulo de barra de 2,31 mm, uno: color negro, y cero: blanco).

### Escáneres de imágenes

Un escáner de imágenes es un sistema para digitalización de documentos, fotografías, etc. Contiene tres elementos funcionales básicos: un detector (con la electrónica asociada), una fuente de luz y lentes de barrido. La fuente de luz ilumina el objeto y las lentes forman la imagen del objeto en el detector. El documento debe ser iluminado con suficiente luz de forma que la luz reflejada y enfocada por las lentes sea suficiente para permitir al detector operar con una adecuada relación señal/ruido. A veces la fuente de luz es un haz generado por un diodo láser que barre los distintos puntos de la retícula de la imagen. En la actualidad se suelen utilizar detectores de **dispositivos de carga acoplada (CCD, *Cargue Coupled Device*)** que están constituidos por diminutos condensadores que se cargan eléctricamente al incidir sobre ellos la luz resultando la carga proporcional a la luz incidente. Los escáneres de color suelen disponer de tres fuentes de luz, correspondientes a los tres colores básicos (rojo, verde y azul). Con la superposición de las tres imágenes que se forman iluminando con cada una de las fuentes se puede almacenar y reconstruir la imagen en color (ver Sección 3.5). Otra opción es tener una única fuente de luz blanca y disponer de tres filtros para captar las intensidades de los tres colores básicos.

Las imágenes digitalizadas generan un mapa de bits, pero puede utilizarse un programa de **reconocimiento óptico de caracteres (OCR, *Optical Character Recognition*)** para detectar e identificar los caracteres (alfabéticos y numéricos) impresos, mecanografiados o manuscritos. De esta forma, una imagen en mapa de bits se puede convertir en texto (un archivo ASCII, por ejemplo) posteriormente editable e imprimible por el usuario.

### EJEMPLO 6.3

Un escáner<sup>3</sup> puede tener las siguientes características:

- Resolución óptica: 2.400 dpi (seleccionable de 25 a 9.600 dpi).
- Interfaz: USB.
- Fuente de luz: LED (diodo emisor de luz).
- Profundidad de color: 48 bits.
- Escala de grises: 16 bits.
- Tamaño del documento: A4.
- Velocidad de vista preliminar (antes de escanear): 9 s.
- Consumo eléctrico: 2,5 W en funcionamiento; 1,5 W en espera.

<sup>3</sup> CanoScan LiDE 80.

## 6.5. Detectores de datos magnetizados

Otros dispositivos de captura directa de datos se basan en la medida de campos magnéticos, como los detectores de caracteres magnetizables y los detectores de bandas magnéticas, que se describen brevemente a continuación.

### Detectores de caracteres magnetizables

Los caracteres magnetizables se utilizan, por ejemplo, en los talones y cheques bancarios, y en las etiquetas de los medicamentos. En estos documentos se imprimen, de acuerdo con unos patrones preestablecidos, los caracteres que identifican el cheque o talón. La tinta utilizada es magnetizable (contiene óxido de hierro) y además es legible directamente por el hombre. Una lectora de caracteres magnéticos contiene una unidad de magnetización por la que pasan los documentos antes de llegar a la estación de lectura. El elemento de lectura contiene una microbobina que va barriendo el carácter e induciendo en ella un potencial eléctrico proporcional a la cantidad de tinta impresa bajo su radio de acción.

### Lectores de bandas magnéticas

Las tarjetas de compra y de crédito, así como otras tarjetas de identificación, suelen llevar una banda de un material plástico recubierto de un material magnetizable, tal como óxido de hierro u óxido de cromo. En esta banda se graban los datos de identificación del usuario de la tarjeta, que pueden ser leídos por dispositivos especiales, tales como los que se encuentran en los cajeros bancarios o lectores de tarjetas de créditos.

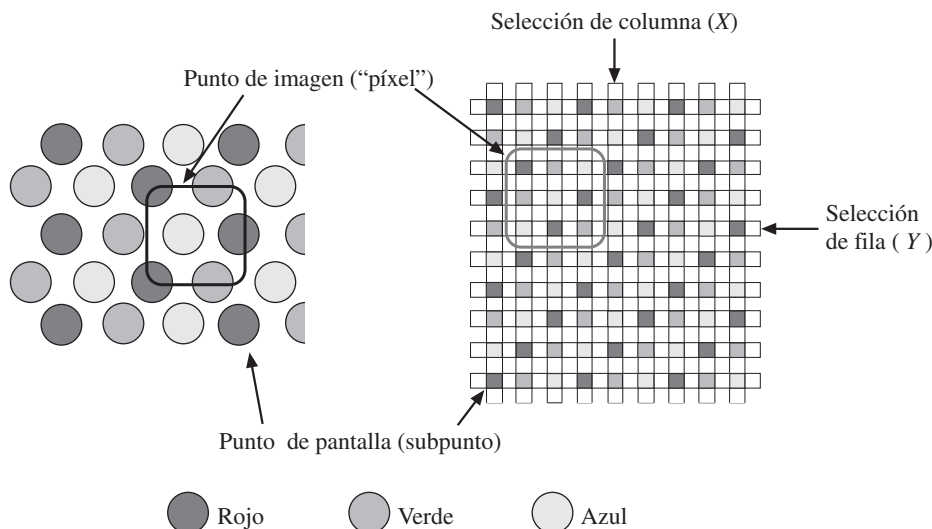
## 6.6. Monitores de visualización

Los monitores de visualización constituyen hoy día el sistema más cómodo y usual de captar las salidas de un computador.

Según analizamos en el Capítulo 2, una imagen en un computador se representa mediante un conjunto discreto de puntos. De igual manera, la imagen de una pantalla no es continua, formándose en la retina del usuario por la yuxtaposición de multitud de **píxeles** (Figura 6.3). Por otra parte, la imagen se forma físicamente en la pantalla del monitor por la activación selectiva de multitud de elementos denominados **puntos de pantalla** o **subpuntos** (*dot pitch*), que en las pantallas en las que la iluminación se produce por excitación de pigmentos de fósforo se denominan **luminóforos**.

Un píxel se iluminará más cuanto mayor sea la activación del elemento correspondiente. El color de cada punto de imagen se obtiene con la mezcla de tres colores básicos: rojo (R), verde (G) y azul (B). Para ello, en la pantalla se encuentran distribuidos alternativamente en todas las direcciones puntos de pantalla emisores de los tres colores (Figura 6.3). Un punto de imagen debe emitir toda una gama de colores, con lo que debe estar formado por al menos un punto de pantalla de cada uno de los colores básicos, activables por separado. Programando de forma individual la intensidad de cada color básico, el ojo humano (que realiza una *integración espacial*) detectará la mezcla obtenida en cada píxel con una gran gama de sensaciones de colores distintos.

Cuando la pantalla se utiliza para visualizar textos, se considera dividida en **celdas**, en cada una de las cuales puede ir un carácter. La celda está constituida por una matriz regular de píxeles. Así, por ejemplo, en una pantalla una celda puede contener  $9 \times 14$  (alto  $\times$  ancho) píxeles formándose un carácter con  $9 \times 7$  puntos. Los principales parámetros que caracterizan a un monitor de visualización son:



**Figura 6.3.** En un monitor de color un punto de imagen (“píxel”) está formado por al menos tres puntos de pantalla.

**Tamaño de la pantalla.** Se da (como en las televisiones convencionales) en función del tamaño de la diagonal principal, en pulgadas. Los tamaños usuales son de 15 o 17 pulgadas (con un área activa de  $210 \times 152$  mm, por ejemplo).

**Relación de aspecto.** Es la relación que existe entre el ancho y el alto de la pantalla. Usualmente la relación de aspecto es  $4/3$ ; aunque se está introduciendo la **relación panorámica**,  $16/9$  que corresponde al cine.

**Ángulo de visión.** En muchas pantallas (principalmente en las planas) la calidad de visión depende del ángulo (con respecto a la perpendicular a la superficie de la pantalla) con que se observe la misma. El ángulo de visión es inferior en las pantallas planas que en las pantallas curvas (CRT).

**Resolución gráfica.** Es el número píxeles en pantalla. Este es uno de los parámetros más importantes para determinar la calidad de visualización de imágenes. En la Tabla 6.5 se dan ejemplos de distintos tipos de resolución.

Tipo de tarjeta	Resolución	N.º colores
MDA	$720 \times 350$	b/n
HGC	$720 \times 350$	b/n
CGA <sup>1</sup>	$640 \times 200$	b/n
EGA	$640 \times 350$	16
VGA <sup>2</sup>	$640 \times 480$	16
SVGA	$800 \times 600$	256
XGA	$1.024 \times 768$	256
Macintosh	$1.152 \times 870$	—
SXGA	$1.280 \times 1.024$	$2^{24}$
UXGA	$1.600 \times 1.200$	$2^{24}$
QXGA	$2.048 \times 1.536$	$2^{24}$

<sup>1</sup> Se puede utilizar también con 4 colores, bajando la resolución a  $320 \times 200$ .

<sup>2</sup> Se puede utilizar también con 256 colores, bajando la resolución a  $320 \times 200$ .

**Tabla 6.5.** Ejemplos de resoluciones.



**Gama de colores.** Es el número de colores o tonos que puede tomar cada punto de la pantalla.

**Resolución óptica (densidad y tamaño de puntos de imagen).** Como acabamos de ver, la resolución gráfica viene determinada por el número de píxeles en filas y columnas, por lo que no depende del tamaño de la pantalla. Como consecuencia de lo anterior, para una resolución gráfica dada, cuanto menor es la pantalla, más próximos aparecerán los puntos dando la imagen mayor sensación de continuidad en sus trazos. Por el contrario, cuanto mayor es la pantalla menor es la calidad de la imagen. Por ello un parámetro de interés es la **resolución óptica** de la pantalla, que es la densidad lineal de puntos de imagen y se suele dar en puntos/pulgada (d/i, o dpi, *dots per inch*). Hay que tener en cuenta que el ojo humano es capaz de resolver 380 d/i en una imagen vista a una distancia de 45 cm. Un texto impreso de una revista alcanza una resolución de 1.200 d/i. Uno de los objetivos tecnológicos de las pantallas grandes es conseguir densidades de 300 d/i.

La distancia entre los centros de dos puntos adyacentes del mismo color se denomina **paso del punto** (*dot-pich*), y es la inversa de la densidad de puntos de pantalla. Para obtener buena calidad de imagen el paso del punto conviene que sea de 0,26 mm o menor. Los tamaños de los puntos son menores que el paso, ya que existe una separación entre ellos (Figura 6.3).

Un monitor de visualización está constituido por dos elementos básicos: una **pantalla de vídeo** y un **controlador de vídeo** (o **controlador gráfico**).

La superficie externa de la pantalla (por donde el usuario observa la imagen) puede ser curva o plana. Las pantallas tradicionales contienen un **Tubo de Rayos Catódicos** (CRT o *Catode Ray Tube*), similar al de los receptores de televisión, y son de tipo curvo. Debido a su principio físico de funcionamiento, las pantallas planas tienen menos fondo, siendo por tanto mucho menos voluminosas que los CRT, y son mucho menos pesadas, por lo que se utilizan preferentemente para computadores portátiles.

En la Tabla 6.6 se muestra una clasificación de los tipos más importantes de pantallas. Algunos de ellos están en proceso de investigación o desarrollo, y aún no han sido comercializados. La siguientes secciones las dedicaremos a pantallas CRT (Sección 6.6.1), pantallas planas (Sección 6.6.2) y controladores de vídeo (Sección 6.6.3).

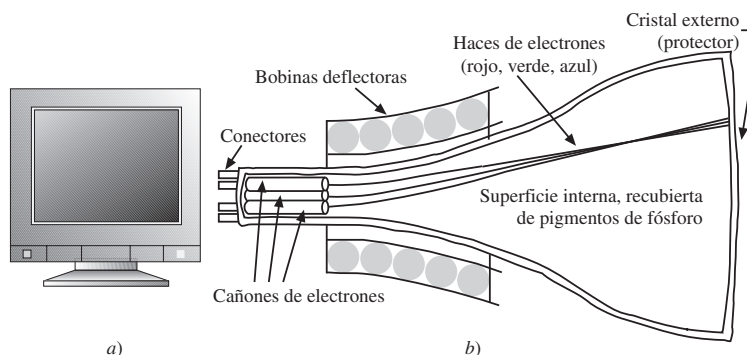
Superficie externa	Fundamento de visualización	Acrónimo	Tipo
Curvas	Tubos de rayos catódicos	CRT ( <i>Catode Ray Tube</i> )	Emisivas
Planas (FDP)	Cristal líquido (LCD)	TN-LCD ( <i>Twisted nematic field effect</i> ), 20 filas. STN-LCD ( <i>Supertwist nematic</i> ), 480 filas. PM-LCD ( <i>Passive-Matrix</i> ). AM-LCD ( <i>Active-Matrix</i> ). F-LCD ( <i>Ferroelectric</i> ). PA-LC ( <i>Plasma-addressed liquid crystal</i> ).	Pasivas
		TFT-LCD ( <i>Thin Film Transistor</i> ).	Emisivas
	Plasma	AC-PDP ( <i>AC plasma display panel</i> ).	
	Electroluminescente	AC-TFEL ( <i>AC thin-film</i> ).	
	Efecto de campo	FED ( <i>Field-emissions displays</i> ).	

**Tabla 6.6.** Distintas tecnologías utilizadas para pantallas de computadores.

### 6.6.1. PANTALLAS DE TUBO DE RAYOS CATÓDICOS (CRT)

La imagen de una pantalla de tubo de rayos catódicos se forma al incidir un haz de electrones (*spot*) sobre la superficie interna de la pantalla, que está recubierta de un material fosforescente (Figura 6.4).





**Figura 6.4.** a) Aspecto de una pantalla de vídeo. b) Esquema simplificado de una pantalla CRT.

Dependiendo del tipo de este material se tienen distintas persistencias de imagen y colores. En las pantallas de color se utilizan tres tipos de fósforos (fósforo rojo, fósforo verde y fósforo azul), que se distribuyen en forma puntual (**luminóforos**) y alternativa a lo largo de las distintas direcciones de la pantalla. Las pantallas CRT hacen que el haz de electrones barra la superficie interna de visualización de la pantalla, de izquierda a derecha y de arriba a abajo y, dependiendo de la intensidad con que inciden los electrones en la pantalla, así de brillante resulta ser cada punto de la imagen. El tamaño del punto producido en la pantalla determinará el número de subpuntos de la pantalla que suele estar comprendido entre 700 y 2.500 a lo largo de cada uno de los ejes.

Para lograr que el haz efectúe el barrido a lo largo de la superficie de visualización, se sitúan externamente al tubo unas bobinas magnéticas (**bobinas deflectoras**) que crean dos campos magnéticos perpendiculares, en direcciones *X* e *Y*, que desvían el haz de electrones en dichas direcciones. Aplicando unas tensiones adecuadas (en forma de “diente de sierra”) a las bobinas deflectoras se consigue que el haz de electrones recorra la superficie de izquierda a derecha y de arriba a abajo, recorriendo de forma ordenada todos los puntos de pantalla. Simultáneamente a efectuar el barrido, la intensidad del haz de electrones se va modulando con una señal analógica procedente del controlador y que corresponde a la intensidad deseada para cada punto de pantalla.

### 6.6.2. PANTALLAS PLANAS

Las pantallas planas (**FPD**, *Flat-Panel Displays*) son ligeras, delgadas y tienen un bajo consumo de potencia. Estas características las hacen preferibles a los CRT para los computadores portátiles, por ejemplo.

Generalmente las pantallas planas constan de dos cristales planos unidos a presión, entre los que se ubican los elementos activos. Utilizan un esquema matricial para seleccionar los elementos de imagen utilizando dos juegos de electrodos conductores transparentes en forma de bandas/tiras. Uno de los juegos se sitúa en sentido horizontal (electrodos horizontales o de filas) y el otro en sentido vertical (electrodos verticales o de columna), y entre ellos se colocan los elementos físicos emisores o reguladores del paso de luz. Los puntos/zonas donde se cruzan los electrodos perpendiculares definen los **puntos de pantalla**, direccionables eléctricamente por fila y columna.

**Pantallas de cristal líquido o LCD (*Liquid Crystal Display*).** Utilizan un cristal líquido; es decir, una sustancia oleaginosa que contiene moléculas (cianobifenil) en forma de pequeñas varillas o barras, que se sitúa entre los juegos de electrodos *X* e *Y*. En estado normal el cristal líquido es transparente; pero si en una zona de él se aplica un campo eléctrico se vuelve opaco; concretamente las moléculas reaccionan ante los campos eléctricos, reorientándose a lo largo de las líneas del campo; así pueden transmitir o bloquear punto a punto el paso de la luz para formar la imagen. Cada punto de imagen es, por tanto, activado sólo durante una fracción del tiempo total de formación del cuadro de

imagen, parpadeando a la frecuencia de cuadro (**matrices de visualización pasivas**). El problema de estas pantallas es que no se refrescan con una adecuada frecuencia, y si, por ejemplo movemos el ratón, el cursor que le sigue en la pantalla desaparece o va dejando una estela. El problema se resuelve con pantallas de doble barrido (*dual scan*) con las que se duplican los circuitos de barrido, y la mejor solución se obtiene utilizando un transistor de película delgada (TFT) por cada punto de imagen, ya que de esta forma se mantiene fija la tensión correspondiente a cada punto (**matrices de visualización activas**) hasta que se refresque (**pantallas TFT**). Éstas son las de mejor calidad, pero más caras.

**Pantallas de plasma.** El funcionamiento de estas pantallas se basa en el principio de que aplicando una tensión alta en un ambiente con un gas a baja presión se emite luz. Estas pantallas están formadas por un tubo con dos electrodos, entre los cuales se encuentra un gas inerte, como argón o neón. La descarga eléctrica provoca la luminosidad de cada subpunto, con un contraste muy elevado y un amplio ángulo de visión. Son emisivas, no existe parpadeo, no hay distorsiones, su espesor es reducido pero tienen una vida limitada.

**Pantallas electroluminiscentes.** Se basan en que cuando pasa una corriente de baja intensidad a través de unos cristales de sulfuro de zinc (u otro compuesto de características similares), brillan. El resultado es una señal de elevada resolución, con la posibilidad de disponer de paneles de grandes dimensiones y un elevado contraste.

**Pantallas de efecto de campo (FED).** En estos monitores se combina el fósforo con la estructura de celdas de las pantallas LCD. Se utilizan mini-tubos para cada píxel y permite conseguir un grosor similar al de las pantallas LCD. Con estas pantallas se obtiene una velocidad de respuesta mayor que con las pantallas TFT y una reproducción de color similar a los monitores CRT, pero el coste y la dificultad de fabricación (480.000 tubos de vacío pequeños por pantalla) y la necesidad de un blindaje de la pantalla hace su viabilidad dudosa.

#### EJEMPLO 6.4

Una pantalla plana<sup>4</sup> puede tener las siguientes características:

- Tecnología: TFT.
- Tamaño: 30".
- Resolución:  $1.280 \times 1.024$ .
- Brillo: 550 candelas/m<sup>2</sup>.
- Contraste: 600:1.
- Ángulo de visión: horizontal, 170°; vertical, 176°.
- Tiempo de respuesta: 12 ms.
- Número de colores: 16,8 millones.
- Frecuencia horizontal: 31-66 Hz.
- Frecuencia vertical: 56-85 Hz.
- Tipo de señal de vídeo: RGB analógico/digital.

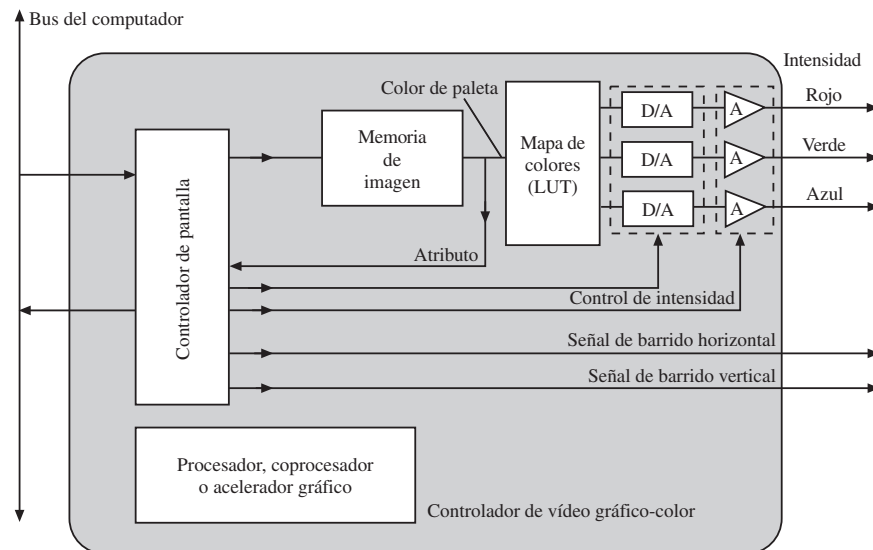
### 6.6.3. CONTROLADOR DE VÍDEO

La mayor parte de los monitores de vídeo no activan los puntos de pantalla de forma continua, sino que lo hacen de forma periódica y durante un corto intervalo de tiempo. La actualización periódica de la información de los puntos de pantalla se denomina **refresco**, e implica un recorrido o **barrido de**

<sup>4</sup> LCD-TFT LG L3020T.

**rastreo** electrónico de los puntos de pantalla, usualmente de izquierda a derecha y de arriba a abajo (tomando como referencia la imagen que percibe el usuario). La **frecuencia de refresco** suele ser de  $F_r = 50, 60, 70$  o  $90$  Hz (período  $T_r = 1/F_r$ ). Un observador no nota el *parpadeo* a esta frecuencia pero al cabo del tiempo sufre una fatiga que será menor cuanto mayor sea la frecuencia de refresco. La imagen que se visualiza en un período  $T$  se denomina **cuadro**; esto es, decir que la frecuencia de barrido es  $50$  Hz, equivale a decir que hay que generar un cuadro cada  $T = 1/50 = 20$  ms.

El controlador de la pantalla contiene una **memoria de vídeo** (también denominada **memoria o buffer de cuadro**) que almacena una reproducción digital de la imagen de pantalla a ser leída iterativamente para que la imagen se mantenga (sea *refrescada*) en la pantalla. Cada palabra de la memoria de cuadro corresponde, por tanto, biunívocamente a un píxel de la pantalla. En la Figura 6.5 se muestra un esquema simplificado de un controlador de vídeo de un monitor gráfico. Cada posición de memoria de cuadro almacena la información de cada píxel, consistente en los niveles de los tres colores básicos. La capacidad de la memoria de cuadro crece con el número de puntos de imagen (resolución) y con el número de bits por punto de imagen (paleta de colores o escala de grises, y atributos).



**Figura 6.5.** Esquema simplificado de un controlador gráfico de vídeo.

Con frecuencia en lugar de almacenar en cada posición de la memoria de vídeo los valores absolutos de los tres colores básicos ( $3 \times 8 = 24$  bits, por ejemplo) se almacena el código de una mezcla elegida de una paleta de colores (Secciones 2.3 y 2.6). En la Figura 6.5 se incluye una memoria que incluye el **mapa de colores** (LUT, *look-up table*) con ayuda del cual se transforma el código de la mezcla en los niveles de los colores básicos que la definen. Los valores binarios obtenidos a la salida de la LUT se transforman a analógicos con ayuda de conversores analógico/digital (Sección 6.9), actuando estas señales directamente como señal moduladora de la intensidad del píxel.

A veces la memoria de vídeo se almacena en una porción de la memoria principal, pero esto supone una gran carga para el bus que interconecta la memoria con el controlador, además de una gran pérdida de la capacidad de la memoria principal (Sección 7.2).

### EJEMPLO 6.5

El ancho de banda entre una memoria y los circuitos de control de la pantalla necesario para refrescar la pantalla 30 veces por segundo con resolución XVGA será:

$$\text{Capacidad de la memoria: } C_M = 1.024 \cdot 768 \cdot 4 \text{ Bytes/pixel} = 3 \text{ MB}$$

Donde hemos tenido en cuenta que la resolución XVGA es de  $1.024 \times 768$  píxeles, y que por cada píxel se almacenan 32 bits (8 bits por cada color básico y otro byte de reserva).

Como hay que transmitir la imagen 30 veces por segundo, el ancho de banda será:

$$AB = C_M \cdot 30 \text{ Hz} = 3 \text{ MB} \cdot 30 \text{ Hz} = 90 \text{ MB/s}$$

Una gran parte de las aplicaciones gráficas (CAD, videojuegos, etc.) necesitan grandes capacidades de cálculo para poder, por ejemplo, a partir de una representación bidimensional (2D), generar imágenes tridimensionales (3D), con perspectiva y con la textura y sombra adecuadas de los distintos objetos. El problema se agrava si deben generarse imágenes en movimiento; así, por ejemplo, para dar la apariencia de un movimiento suave en la pantalla hay que recalcular entre 30 y 40 veces por segundo los píxeles de la imagen. Para descargar al procesador central de estas tareas se han desarrollado **aceleradores gráficos** y **procesadores gráficos**, orientados a estos tipos de aplicaciones. Los términos de acelerador o procesador se utilizan según que el circuito correspondiente realice tan sólo funciones delegadas por el procesador o funciones más complejas programables directamente por el usuario, respectivamente.

### EJEMPLO 6.6

Un procesador gráfico<sup>5</sup> comercial puede tener las siguientes características:

- Arquitectura superescalar de 16 canales de procesamiento (Sección 7.4).
- Bus: PCI Express (> 4 GB/s) (Tabla 7.2).
- Color: 32 bits.
- Memoria: 2 módulos RAMDAC a 400 MHz. Una memoria RAMDAC (*Random Access Memory Digital-to-Analog Converter*) es una memoria SRAM cuyas entradas son binarias y tiene en la salida tres conversores D/A que proporcionan las señales analógicas RGB para actuar directamente con la pantalla (moduladores de los haces de electrones en una CRT).
- Refresco: hasta 85 Hz para  $2.048 \times 1.536$  píxeles.
- Permite conectar pantallas planas grandes con una resolución máxima de  $1.900 \times 1.200$  píxeles.
- Codificador integrado para televisión de alta definición, con resolución máxima de:  $1.920 \times 1.200$  píxeles.

## 6.7. Impresoras

Las impresoras son periféricos que escriben la información de salida (caracteres o puntos que forman una imagen) sobre papel. Su comportamiento inicialmente era muy similar a las máquinas de escribir, pero hoy día son mucho más sofisticadas, pareciéndose algunas en su funcionamiento a máquinas fotocopadoras conectadas en línea con el computador.

La parte mecánica de una impresora, además de encargarse de accionar los elementos que hacen que se imprima el carácter correspondiente, debe dedicarse a la alimentación y arrastre del papel.

El procedimiento de impresión de imágenes en color es similar al de las pantallas, es decir, un color concreto se genera por mezcla de colores; ahora bien, en el caso del papel, el color que percibe el

<sup>5</sup> Nvidia Geforce 6200.

ojo humano no es el generado por una fuente de luz (filtrado o no) sino el que *refleja* el papel, esto es, el no absorbido por la tinta impresa, por lo que habrá que utilizar colores sustractivos. Otra cuestión a tener en cuenta es que, en general, no se dispone de colorantes que produzcan tres colores básicos con calidad, por lo que se suelen utilizar cuatro: uno para el negro (denominado color K de *blacK*) y los otros tres: azul (o C, de Cyan), amarillo (o Y, de Yelow) y rojo (o M, de Magenta). Este conjunto de colorantes se suele denominar **CYMK**.

Existen diversos criterios para clasificar las impresoras, según se especifica en la Tabla 6.7. A continuación describiremos los tipos de impresoras más importantes.

- |  |
|--|
| <ul style="list-style-type: none"> <li>• <b>Calidad de impresión:</b> <ul style="list-style-type: none"> <li>— Normal (matriciales, térmicas).</li> <li>— Semicalidad (matriciales, inyección de tinta).</li> <li>— Calidad (sublimación, inyección de tinta, láser).</li> </ul> </li> <li>• <b>Fundamento del sistema de impresión:</b> <ul style="list-style-type: none"> <li>— Impacto (matriciales).</li> <li>— No impacto (inyección de tinta, láser).</li> </ul> </li> <li>• <b>Forma de imprimir los caracteres:</b> <ul style="list-style-type: none"> <li>— Impresoras de caracteres (matriciales, inyección de tinta).</li> <li>— Impresoras de líneas (térmicas).</li> <li>— Impresoras de páginas (láser e inyección de tinta).</li> </ul> </li> </ul> |
|--|

**Tabla 6.7.** Clasificación de impresoras según diversas características.

### Impresoras matriciales o de agujas

También denominadas de *matriz de puntos*, se utilizan cuando se requiere imprimir con copias (ya que son de impacto); de este tipo suelen ser las pequeñas impresoras (cajas registradoras, comprobantes de pago con tarjetas de crédito, cajeros de entidades bancarias, etc.). Los caracteres se forman por medio de una matriz de puntos (de  $7 \times 5$ ,  $7 \times 9$ ,  $8 \times 9$ ,  $9 \times 9$ ,  $9 \times 11$ ,  $18 \times 8$  o  $24 \times 24$ , etc.). La cabeza de impresión contiene un conjunto de agujas regularmente dispuestas, y cada una de ellas por acción de un electroimán puede ser disparada golpeando la cinta entintada, transfiriéndose así al papel los puntos correspondientes a las agujas disparadas.

### Impresoras térmicas

Se caracterizan porque cada cabezal o sistema de impresión tiene asociado una matriz de pequeñas protuberancias dentro de cada una de las cuales hay una resistencia eléctrica de caldeo. La temperatura de cada una de las protuberancias determina la intensidad de la impresión del punto donde esté posicionada. Podemos incluir en este grupo los siguientes tipos:

- **Impresoras de papel térmico.** Son similares a las impresoras de agujas. Se utiliza un papel especial termosensible que se ennegrece al aplicar calor (unos  $200^\circ$ ). El calor se transfiere desde el cabezal por la matriz de resistencias. Al pasar una corriente eléctrica por las resistencias se calientan, formándose los puntos en el papel.
- **Impresoras de cera.** En este caso los puntos de escritura se forman con cera. Al igual que las primitivas cintas entintadas de color para máquinas a escribir, se dispone de una cinta o banda de gran anchura, formada por tres sub-bandas recubiertas con cera de los colores CYMK. El sistema hace que sucesivamente pasen por encima de la superficie a imprimir las cuatro sub-bandas, y, en cada una de esas situaciones, se aplica un nivel de corriente eléctrica a cada resistencia del punto de imagen con un valor en consonancia con la intensidad con la que deba aplicarse el color básico correspondiente.

- **Impresoras de transferencia térmica con sublimación.** El sistema es análogo a las impresoras de cera, pero más sencillo ya que en lugar de tener que disponer de la banda encerada, el medio de impresión está constituido por paneles o tabletas CYMK sólidas que van posicionándose en la zona de impresión. En este caso los diminutos y numerosos calefactores hacen que se vaporicen (se *sublimen*) los colorantes sólidos y se fijen, por contacto, en el papel. Como en las impresoras de papel térmico, es necesario un papel especial, pero por el contrario se obtienen imágenes de color de una extraordinaria calidad (artes gráficas e impresión fotográfica) ya que se controla de forma eficiente la cantidad de colorante trasferida en cada punto de imagen (250 niveles) y se obtienen mezclas de los colores básicos muy eficientes por realizarse en fase gaseosa.

### Impresoras de inyección de tinta

El fundamento físico es similar al de las pantallas de CRT de vídeo. En lugar de utilizar un haz de electrones se emite un chorro de gotas de tinta ionizadas, que en su recorrido es desviado por unos electrodos que se encuentran a un potencial fijo (del orden de 3 KV). El carácter se forma con la tinta que incide en el papel. La desviación de las gotas, y por tanto la forma del carácter, se regula variando la carga inicial de la gota dada en un electrodo de carga (el potencial de éste varía de 0 a unos 200 V). Cuando no se debe escribir, las gotas de tinta se desvían hacia un depósito de retorno. Para lograr una buena nitidez de la imagen y de los colores que la componen es necesario que el tamaño del punto provocado por cada gota sea lo menor posible, consiguiéndose en algunas impresoras gotas de tan sólo 1 picolitro de volumen. Este tipo de impresora dispone de uno, caso de impresoras blanco y negro, o un conjunto CYMK de tinteros (impresoras de color) acoplados a uno o varios goteros, a través de los cuales se dispara (inyecta) la tinta hacia el papel en forma de diminutas gotitas. Las gotas se suelen obtener por uno de los siguientes dos procedimientos:

- **Burbuja térmica (*Bubble-Jet*).** Se calienta instantáneamente la tinta, se vaporiza en forma de burbuja, hasta que explota, arrastrando la tinta del gotero hacia el exterior.
- **Pulverización piezoeléctrica.** El pulverizador contiene un cristal piezoeléctrico que se dilata y contrae al aplicar un campo eléctrico. Al contraerse el gotero se expulsa una gota, y al dilatarse absorbe el fluido del tintero.

### Impresoras láser

Las impresoras láser tienen una gran importancia por su elevada velocidad, calidad de impresión, bajo precio y utilizar, la mayoría de ellas, papel normal.

La página a imprimir se transfiere al papel por contacto, desde un tambor que contiene una imagen impregnada en **tóner** (polvo de carbón). El tambor está recubierto de un material fotoconductor (usualmente selenio), que a oscuras mantiene la carga eléctrica, y con iluminación se descarga. El tambor, en su giro, inicialmente pasa por delante de una estación que carga negativamente y de forma uniforme las generatrices del tambor que pasan delante de ella. A continuación la generatriz pasa por una zona donde es barrida por un haz láser cuya intensidad es modulada de acuerdo con el nivel de gris o del color a imprimir; los puntos sobre los que incide el haz de láser con mayor intensidad se descargan completamente, correspondiendo a las zonas o puntos que aparecerán no impresos (en blanco) en la página. Posteriormente, la generatriz pasa por la estación del tóner, que al estar cargado positivamente es fuertemente atraído por los puntos de la generatriz del tambor cargados negativamente. Inmediatamente después, electrostáticamente, por presión y por calor se transfiere el tóner del tambor al papel, quedando éste escrito. Por último, una cuchilla, del último elemento por el que pasa el cilindro, limpia éste de los restos de tóner que por error no se hayan adherido al papel.

**EJEMPLO 6.7**

Una impresora<sup>6</sup> de inyección de tinta puede tener las siguientes características:

- Tecnología de impresión: inyección térmica de tinta.
- Conexión al computador: inalámbrica (estándar 802,11 b/g), USB, Ethernet.
- Resolución máxima: b/n:  $1.200 \times 1.200$  dpi; color:  $4.800 \times 1.200$  dpi.
- Velocidad de impresión, calidad borrador: b/n: 30 ppm; color: 20 ppm.
- Velocidad de impresión, calidad óptima: b/n: 2,1 ppm; color: 2,1 ppm.
- Memoria intermedia: 32 MB.

## 6.8. Periféricos para aplicaciones multimedia

Las aplicaciones multimedia son programas que utilizan conjuntamente sonidos, imágenes estáticas e imágenes de vídeo, producidas o no por computador. Estas aplicaciones tienen las siguientes peculiaridades:

- Suelen requerir archivos de gran capacidad, por lo que utilizan las técnicas de compresión de datos vistas en la Sección 2.6.
- Necesitan gran potencia de cálculo, de forma que los repertorios de instrucciones de los modernos microprocesadores suelen incluir conjuntos de instrucciones específicas para mejorar las prestaciones en aplicaciones de imágenes, audio, vídeo 3D y reconocimiento del habla. Este es el caso, por ejemplo, de los conjuntos de instrucciones máquina SSE y MMX de los microprocesadores Pentium.
- El bus (o buses) de interconexión entre memoria y controlador de pantalla debe de ser de gran velocidad, sobre todo si se quieren visualizar imágenes en movimiento.
- Debe disponerse de periféricos adecuados. Puede considerarse que los periféricos específicos de multimedia son el sistema de audio, y sistema de vídeo incluyendo una pantalla de alta resolución.

### Sistemas de audio

Las señales de audio, u ondas sonoras, se transmiten en forma de cambios rápidos de la presión del aire. El oído humano es capaz de detectar ondas de sonido de frecuencias comprendidas entre 20 y 20.000 Hz. Los sensores de señales de audio se denominan **micrófonos** y los actuadores **altavoces**. Los primeros transforman las ondas sonoras en señales analógicas, que posteriormente deben digitalizarse con un conversor A/D (Sección 6.9) para ser procesadas por un computador. Los altavoces transforman señales analógicas en señales de audio (generan ondas de presión).

Las funciones de audio relacionadas con los computadores son:

- **Síntesis de sonido**, que se realiza con **sintetizadores de sonido**, la mayoría de los cuales se basan en la idea de que cualquier sonido se puede formar con la superposición de señales sinusoidales, o, en general, señales periódicas.
- **Reproducción de sonido**. El estándar **MIDI** (*Musical Instrument Digital Interface*) permite la interconexión de distintos instrumentos musicales y accesorios electrónicos juntos (sintetizadores, secuenciadores, máquinas rítmicas y diversos PC). Incluye la definición de un puerto serie específico y un protocolo de transferencia de *órdenes* a través del puerto.

<sup>6</sup> HP Deskjet serie 6840 (C9031B).



- **Síntesis del habla.** Las unidades sintetizadoras de la voz son dispositivos que dan los resultados de un programa emitiendo sonidos (fonemas o palabras) emulando el habla humana.
- **Reconocimiento del habla.** Con estos sistemas se pretende una comunicación directa del hombre con el computador, sin necesidad de transcribir la información a través de un teclado u otros soportes intermedios de información.

Para captar, procesar y producir sonidos con cierta calidad es necesario disponer de un sistema de sonido que incluya los siguientes elementos:

- **Tarjeta de sonido,** que contiene conversores A/D (entrada) y D/A (para salida), sintetizador de sonidos e interfase MIDI. Permiten convertir, crear, almacenar y mezclar sonidos.
- **Altavoces externos,** con bafles, amplificadores y alimentación autónoma.
- **Software específico,** incluye programas de control.

### Sistemas de vídeo

Los sistemas de vídeo pueden procesar imágenes por medio de programas y tarjetas de vídeo. Entre las operaciones a realizar se encuentran retoque o transformación de imágenes, superposición y cambio del orden de imágenes, etc. En cuanto a periféricos de entrada se pueden considerar las máquinas de fotos digitales y las cámaras de vídeo digitales.

Una **máquina de fotos digital** se puede utilizar para introducir imágenes (fotos) estáticas directamente, es decir, sin necesidad de revelar un carrete y utilizar un escáner para digitalizar la imagen. Una máquina de fotos digital enfoca, a través de una óptica adecuada, una imagen estática sobre un filtro de colores rojo, verde y azul, la transforma en señales eléctricas por medio de sensores CCD, la digitaliza con un conversor A/D y la graba en una memoria PROM (generalmente de tipo *Flash*) o disquete, no requiriendo un carrete fotográfico. Debido a que para cada punto de imagen hay que grabar el color asociado, estas imágenes necesitan una gran capacidad de almacenamiento que se reduce notablemente aplicando técnicas de compresión de datos.

Por otra parte, existen pequeñas **cámaras digitales de vídeo** (*WebCam*) que se pueden acoplar a un computador. Estas cámaras tienen una óptica y sistema de sensores (CDC) similar a las cámaras convencionales, pero son más sencillas ya que únicamente tienen que generar imágenes en formato de mapa de bits y transmitir las al computador; es decir, no incluyen los mecanismos de grabación y tracción para almacenar la imagen en un dispositivo de cinta magnética.

Otra posibilidad de introducir en un computador señales de vídeo es desde una cámara de vídeo convencional y una tarjeta de vídeo. Una **tarjeta de edición de vídeo** permiten captar señales de radio y/o vídeo, demodularlas y almacenarlas en forma digital, y/u oírlas o verlas, directa o posteriormente, a través del sistema de audio o la pantalla del computador. Básicamente constan de una entrada de antena, un circuito sintonizador y un conversor A/D.

## 6.9. Entradas/salidas de señales analógicas

Una gran parte de los computadores de uso específico, y más concretamente de los sistemas embebidos, deben operar con señales del entorno que son de tipo analógico (Capítulo 3). También, con frecuencia, es necesario que el computador genere resultados en forma analógica, por ejemplo, para actuar sobre una electroválvula que controla el paso de flujo de un gas o un líquido a través de una conducción, o para desplazar el brazo de un robot.

Un **sensor** o **detector** es un dispositivo que transforma señales físicas de una determinada naturaleza (temperatura, presión, posición, humedad, etc.) en señales eléctricas. Un **efector** o **actuador**



realiza la transformación inversa: convierte una señal eléctrica en otra de distinta naturaleza. Las señales eléctricas involucradas en estos procesos de transformación se denominan **señales analógicas** ya que varían de forma *análoga* a como la hace la señal física original, pudiendo tomar cualquier valor comprendido dentro de un determinado rango.

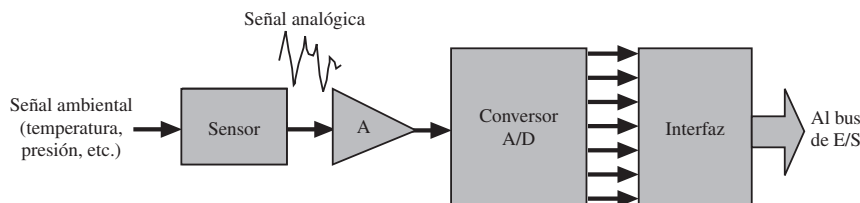
A continuación se dan unas ideas generales sobre sistemas de entrada y sistemas de salida para señales analógicas; es decir, sistemas que captan o proporcionan datos de tipo analógico.

### Sistemas de adquisición de datos analógicos

La mayor parte de las variables físicas de la naturaleza (temperatura, intensidad luminosa, posición, sonido, etc.) son señales o funciones que varían continuamente con el tiempo. Existen sensores específicos para cada tipo de magnitud; así hay detectores de:

- Temperatura (termistores, termopares, etc.).
- Presión.
- Intensidad de luz (fotodetector, etc.).
- Humedad.
- Humo.
- Caudal de líquido.
- Sonido (un micrófono, por ejemplo).
- Nivel de agua.
- Posición lineal (potenciómetros lineales, por ejemplo).
- Posición angular (por ejemplo, potenciómetros circulares, sincrogeneradores, etc.).
- Sensores de señales fisiológicas (que se utilizan en electroencefalografía, o electrocardiología, por ejemplo), etc.

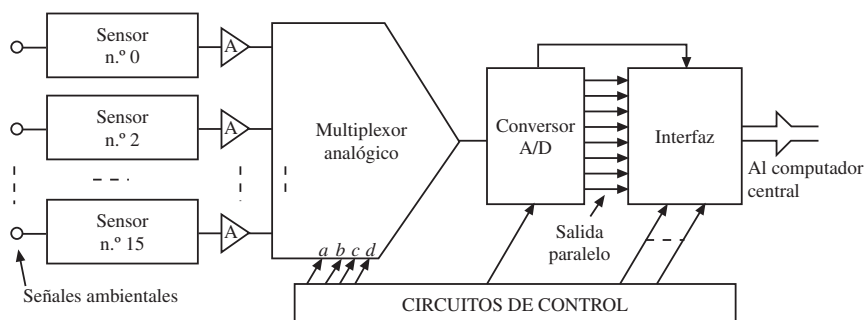
Una vez convertida la señal original en señal eléctrica, se amplifica o atenúa adecuadamente (bloque A de la Figura 6.6). Es necesario, a continuación, darle una forma apta para ser tratada por el computador o, en otras palabras, transformarla en datos numéricos binarios según el código que utilice el computador. Esto se hace con unos circuitos electrónicos específicos denominados **convertidores analógico/digitales** o, abreviadamente, convertidores A/D. El convertidor A/D capta muestras de la señal analógica de entrada y las mide o digitaliza, dando a su salida un conjunto de bits o número binario que representa el valor de la amplitud de la muestra captada (Figura 6.6).



**Figura 6.6.** Diagrama de bloques de un sistema de entrada analógica a una computadora.

Dependiendo de la naturaleza de la señal original y de la aplicación del sistema, la velocidad de captación de muestras (velocidad de muestreo) debe ser mayor o menor. Por ejemplo, en el caso del sistema de un sistema de control industrial se podrían captar muestras del pH de un determinado líquido en un depósito cada 15 minutos. El sistema trata de detectar cuándo el pH sobrepasa un determinado nivel, para, en ese caso, hacer las correcciones oportunas. Entre la toma de una muestra y la siguiente, el convertidor A/D y el resto del sistema podrían tomar muestras de otro tipo: por ejemplo,

temperaturas y pH de otros depósitos. El mismo conversor podría convertir a binario las distintas muestras analógicas. Para lograr este objetivo se dispone de un circuito denominado **multiplexor analógico** que selecciona uno de entre varios canales de entrada (Figura 6.7). En la figura se selecciona como entrada del conversor uno de los 16 canales de entrada; la selección la realizan las señales de control binarias *a*, *b*, *c*, *d*. Así, *abcd* = 0000 selecciona el canal 0, *abcd* = 0001 el canal 1, ..., y *abcd* = 1111 el canal 15.



**Figura 6.7.** Diagrama de bloques de un sistema de adquisición de datos de 16 canales.

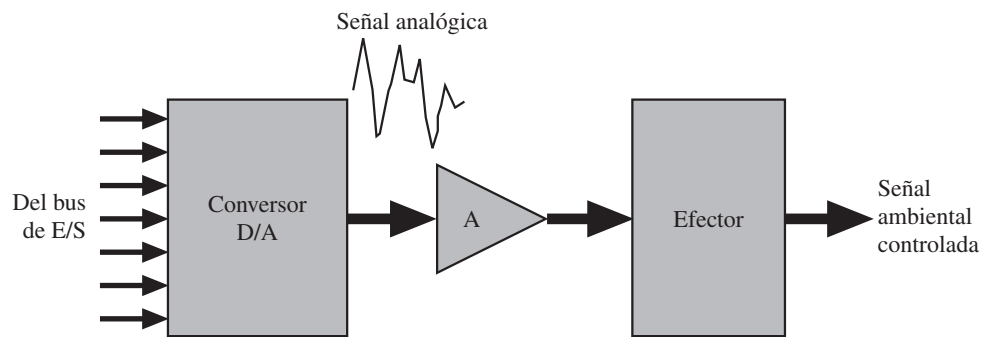
La Figura 6.7 representa un esquema completo de una unidad de entradas analógicas denominada también **sistema de adquisición de datos analógicos**, en este caso de 16 canales de entrada. Después de los sensores se incluyen unos amplificadores o adaptadores (A) para acoplar las señales analógicas a los rangos de amplitud que requiere el multiplexor analógico. Las señales de control pueden ser generadas por circuitos especializados incluidos en el periférico (en el sistema de adquisición de datos) o en el propio procesador del sistema, controlándose la generación de las mismas por programa (por software). La conexión con el computador se efectúa como un periférico normal, a través de un bus de E/S.

### Sistemas de salida analógica

A veces las salidas de un computador deben ser analógicas para actuar sobre un sistema o dispositivo controlable por una señal eléctrica analógica. La señal de salida actúa sobre un **efector** generándose así una señal no eléctrica: el cierre de un contacto eléctrico (relé) de gran potencia (para encender unas lámparas o un horno, por ejemplo), o provocar un movimiento determinado excitando un motor (torreta de una máquina herramienta, por ejemplo), abrir una válvula de una conducción de pintura de un brazo robotizado, etc. Los dispositivos físicos más utilizados, controlables por señales eléctricas, son:

- Contactos electromecánicos o relés. En estos dispositivos una pequeña corriente eléctrica en un electroimán provoca, por atracción magnética, el cierre de un interruptor que soporta una gran intensidad (resistencia de un horno eléctrico, etc.). Existen dispositivos electrónicos de semiconductor (tiristores) que realizan la misma función que los relés.
- Electroválvulas: abren o cierran una conducción de fluido.
- Servomotores: provocan el giro de un eje.
- Altavoz: origina la emisión de un sonido.

La conversión eléctrica de un dato binario en una señal analógica se efectúa en un circuito denominado **convertor digital/analógico**, o, abreviadamente, **convertor D/A**. La Figura 6.8 muestra un esquema simplificado de una unidad de salida analógica.



**Figura 6.8.** Diagrama de bloques para una salida analógica.

Comercialmente se dispone de sistemas de adquisición de datos completos (multiplexor, conversores A/D, interfaz y circuitos de control) en una tarjeta de circuito impreso, incluyendo también sistemas de salida analógica. También los **procesadores digitales de señales (DSP, Digital Signal Processor)** y algunos microcontroladores disponen internamente de conversores A/D y D/A.

### EJEMPLO 6.8

Un sistema de adquisición de datos<sup>7</sup> puede tener las siguientes características:

- Interfaz USB para conexión con el computador (PC y equipos portátiles o móviles).
- 8 canales analógicos de entrada, cada uno de ellos de 12 bits. Admitiendo 4 de ellos muestreo simultáneo.
- 2 conversores D/A con salidas analógicas (rango de 12 bits y velocidad de 500 KS/s, S simboliza muestra, *Sample*).
- 8 entradas digitales independientes y 8 salidas digitales independientes.
- Rango de la señal de entrada programable.
- Memoria FIFO de 4 KS.
- Máxima frecuencia de muestreo: 500 KS/segundo.
- 3 contadores de 16 bits.

## 6.10. Conclusiones

En este capítulo se han presentado los principales dispositivos de entrada y salida de los computadores tradicionales. Así, después de presentar unos conceptos generales sobre periféricos de entrada/salida, se han descrito tipos muy diversos de ellos, como monitores, impresoras, teclados, detectores ópticos, dispositivos de captura directa de datos, interfaces industriales, periféricos para aplicaciones multimedia y dispositivos para entradas manuales directas (ratones, palanca de control, etc.).

<sup>7</sup> USBDAQ-9100MS.

## Test



**T6.1.** Los periféricos están constituidos por:

- a) Unidades de entrada y de salida.
- b) Unidades de entrada, de salida y de memoria (principal y auxiliar).
- c) Unidades de entrada, de salida y de memoria auxiliar.
- d) Unidades de memoria auxiliar (discos magnéticos, discos ópticos, cintas magnéticas, etc.).

**T6.2.** Para realizar un reconocimiento óptico de caracteres (OCR) se necesita disponer de:

- a) Un digitalizador y un programa de reconocimiento.
- b) Un lápiz óptico, electrostático o de presión y un programa de reconocimiento.
- c) Un escáner de imágenes y un programa de reconocimiento.
- d) Un ratón y un programa de reconocimiento.

**T6.3.** Los paneles sensibles a la presión se utilizan frecuentemente en:

- a) Agendas electrónicas (PDA).
- b) PC de sobremesas.
- c) Interfases industriales (con conversores A/D y D/A).
- d) Equipos de videojuegos.

**T6.4.** Los tipos de ratón más duraderos (menos frágiles) son:

- a) Los mecánicos.
- b) Los opto-mecánicos.
- c) Los ópticos.
- d) Los estacionarios.

**T6.5.** Un CCD es un dispositivo:

- a) Para detectar marcas magnéticas.
- b) Para detectar señales luminosas.
- c) Para multiplexar señales analógicas.
- d) Para mezclar y procesar señales de audio.

**T6.6.** Un *píxel* es:

- a) La superficie de visualización más pequeña que percibe el ojo.
- b) La superficie de visualización más pequeña que se corresponde a un color básico individual.
- c) Una celda individual donde se puede visualizar un carácter u otro objeto.
- d) El punto que produce el impacto del haz de electrones en la pantalla.

**T6.7.** La relación de aspecto usual de una pantalla es de:

- a) 1/1.
- b) 2/1.
- c) 4/1.
- d) 4/3.

**T6.8.** El tamaño de una pantalla se da en pulgadas, y corresponde a:

- a) El ancho de la imagen que se visualiza en la pantalla.
- b) El alto de la imagen que se visualiza en la pantalla.
- c) La diagonal de la imagen que se visualiza en la pantalla.
- d) El ancho por el alto de la imagen que se visualiza en la pantalla.

**T6.9.** La altura de la imagen de una pantalla convencional es aproximadamente de 26 cm. ¿Cuál es el tamaño del monitor?

- a) 15".
- b) 17".
- c) 19".
- d) No se puede deducir el resultado con los datos que se dan.

**T6.10.** La resolución óptica ("continuidad") de una imagen depende:

- a) Sólo de la resolución gráfica del monitor.
- b) De la resolución y tamaño de la pantalla (densidad de elementos de imagen).
- c) Sólo del tamaño de la pantalla.
- d) De la relación de aspecto.

**T6.11.** Los siguientes adaptadores de vídeo en orden creciente de resolución son:

- a) CGA, EGA, SVGA, SXGA, VGA, UXGA.
- b) CGA, EGA, VGA, SVGA, SXGA, UXGA.
- c) CGA, EGA, SVGA, VGA, SXGA, UXGA.
- d) CGA, EGA, VGA, SVGA, UXGA, SXGA.

**T6.12.** La memoria de vídeo de un procesador gráfico de  $1.600 \times 1.280$  píxeles que utiliza una paleta de 1.024 colores debe ser del orden de:

- a) 20 MBytes.
- b) 250 MBytes.
- c) 2,5 MBytes.
- d) 19,5 MBytes.

**T6.13.** La capacidad de la memoria de vídeo para una imagen con resolución SVGA ( $800 \times 600$ ; 256 colores) es:

- a) 3,7 MBytes.
- b) 59 KBytes.
- c) 469 KBytes.
- d) 117 MBytes.

**T6.14.** Suponiendo una pantalla de 90 Hz de frecuencia de refresco, y que utiliza resolución X VGA ( $1.024 \times 768$ ; 256 colores), el ancho de banda mínimo que debe tener el bus de salida de la memoria de imagen hacia la pantalla es:

- a) 8,5 MBytes/s.
- b) 16,9 MBytes.
- c) 67,5 MBytes/s.
- d) 24 MBytes/s.

**T6.15.** La mejor calidad de impresión (color) se puede obtener con impresoras de tipo:

- a) Láser.
- b) Inyección de tinta sólida.
- c) Térmicas con sublimación.
- d) De papel térmico.

**T6.16.** La mayor velocidad de impresión se puede obtener con impresoras de tipo:

- a) Láser.
- b) Inyección de tinta sólida.
- c) Térmicas con sublimación.
- d) De papel térmico.

**T6.17.** Una hoja en tamaño DIN-A4 mide  $21,7 \times 29,7$  cm. ¿Qué capacidad de memoria (sin comprimir) se necesitará para guardar la información correspondiente a una hoja DIN-A4 en una impresora láser que tiene una resolución de 1.800 dpi, y cada punto se representa con uno entre 16 niveles de gris?

- a) 155 MBytes.
- b) 618 MBytes.
- c) 223 KBytes.
- d) 1,2 GBytes.

**T6.18.** Un periférico multimedia es un dispositivo de E/S:

- a) Adaptado a trabajar en distintos medios (húmedos, secos, altas temperaturas, etc.).
- b) Que permite simultáneamente la entrada o salida de texto, sonido e imágenes (estáticas o de vídeo).
- c) Un periférico para entrada o salida de sonidos o imágenes (estáticas o de vídeo), con buena calidad.
- d) Utilizado habitualmente por distintos *medios* de comunicación (radio, televisión o prensa).

**T6.19.** Una interfaz MIDI sirve para:

- a) Interconectar distintos periféricos multimedia.
- b) Interconectar distintos equipos musicales y accesorios electrónicos (con un PC, por ejemplo).
- c) Sintetizar voz con gran calidad.
- d) Clasificar palabras en un sistema de reconocimiento automático de la voz.

**T6.20.** Suponiendo que se tiene una cámara de vídeo que utiliza una paleta de 256 colores, resolución VGA ( $320 \times 200$ ) y que transmite 30 imágenes por segundo, el ancho de banda mínimo (sin compresión de datos) que debe tener el bus de conexión con las unidades centrales del computador es:

- a) 15 MBytes/s.
- b) 1,9 MBytes/s.
- c) 235 KBytes/s.
- d) 2,8 KBytes.

**T6.21.** Si se desea que un computador capte periódicamente la temperatura ambiente de un determinado recinto, es necesario utilizar como dispositivo de entrada:

- a) Un sensor de temperatura y un conversor A/D.
- b) Basta con un efector de temperatura.
- c) Un efector de temperatura y un conversor D/A.
- d) Basta con un conversor A/D.

**T6.22.** Un conversor A/D tarda en convertir una muestra  $10 \mu s$ . ¿Cuál es la frecuencia máxima a la que se puede muestrear con dicho conversor?

- a) 10 MHz.
- b) 100 KHz.
- c) 100 Hz.
- d) 100 MHz.

**T6.23.** Un multiplexor analógico es un dispositivo que permite:

- a) Solapar en el tiempo distintas señales que proceden de distintos sensores.
- b) Solapar en el tiempo distintas señales procedentes de distintos conversores A/D.
- c) Amplificar las señales analógicas procedentes de distintos sensores.
- d) Solapar en el tiempo distintas señales procedentes de un único procesador para proporcionárselas a un único conversor D/A.

**T6.24.** Para poder procesar con un único procesador las distintas señales (8, por ejemplo) detectadas por los sensores de una planta química, utilizaría como dispositivo de entrada:

- a) Un escáner multiplexado (con 8 entradas).
- b) Un sistema de adquisición de datos de 8 canales.
- c) Un multiplexor digital y 8 efectores.
- d) Una tableta digitalizadora  $8 \times 8$ .

**T6.25.** Un conversor D/A es un circuito para:

- a) Transformar una señal eléctrica en otra señal de distinta naturaleza.
- b) Transformar una señal no eléctrica en eléctrica.
- c) Transformar una señal binaria en continua.
- d) Transformar una señal continua en una señal binaria.

## Problemas resueltos



## ESCÁNERES

**P6.1.** Un escáner de color tiene una resolución de 1.200 *dpi* y puede detectar 256 niveles para cada uno de los tres colores básicos. Estimar la memoria necesaria para almacenar una fotografía de  $10 \times 13$  cm.

## SOLUCIÓN

La fotografía tiene un total de:

$$\frac{10 \cdot 13}{2,54} = 51,18 \text{ pulgadas}^2$$

En una pulgada cuadrada ( $p^2$ ) hay:

$$1.200 \cdot 1.200 = 1.440.000 \text{ píxeles/pulgada}^2$$

Es decir, el número de puntos a digitalizar es:

$$51,18 \text{ pulgadas}^2 \cdot 1.440.000 \frac{\text{píxeles}}{\text{pulgada}^2} = 73.899.200 \text{ píxeles} = 70,93 \text{ Megapíxeles}$$

Para codificar cada color se necesitan 8 bits, ya que con 8 bits se pueden codificar  $2^8 = 256$  niveles. Como hay 3 colores se necesitan 24 bits, es decir 3 bytes, para codificar el color de cada píxel. Por lo que la capacidad de memoria necesaria para almacenar la imagen será:

$$70,23 \text{ Mpíxeles} \cdot 3 \frac{\text{Bytes}}{\text{píxel}} = 211 \text{ MB}$$

Recuérdese que existen técnicas especiales para comprensión de imágenes, tal y como se vio en la Sección 2.6.

## PANTALLAS

**P6.2.** En una pantalla de resolución X VGA con  $1.280 \times 960$  píxeles, suponiendo que con un determinado editor cada celda de almacenamiento de caracteres ocupa 36 filas  $\times$  25 columnas de píxeles, indicar cuántos caracteres se pueden visualizar por fila y por columna.

## SOLUCIÓN

Como la resolución X VGA utilizada es de  $1.280 \times 960$ , podríamos obtener:

- Número de columnas:

$$N_c = \frac{1.280}{16} = 80$$

- Número de filas:

$$N_f = \frac{960}{36} = 26$$

Es decir, se podrán visualizar 80 caracteres por fila y un total de 26 filas.

**P6.3.** Determinar el tamaño máximo de un píxel y de un punto de pantalla con una resolución Super VGA en una pantalla de color de 14 pulgadas.

## SOLUCIÓN

Se supone una relación de aspecto de la pantalla de  $4/3$ , con lo que, llamando  $a$  a la anchura de la imagen,  $h$  a la altura de la imagen y  $d$  a la diagonal, se verifica:

$$a^2 + h^2 = d^2 \quad ; \quad a^2 + h^2 = 14^2 \quad ; \quad \frac{a}{h} = \frac{4}{3}$$

Resolviendo el sistema anterior se tiene que  $a = 11,2'' = 284 \text{ mm}$  y  $h = 8,4'' = 213 \text{ mm}$ .

Para obtener una resolución Super VGA, debe haber 600 puntos de imagen por fila, con lo que cada punto de imagen deberá ser menor que:

$$\frac{284 \text{ mm}}{600} = 0,47 \text{ mm}$$

Como la pantalla es de color, cada punto de imagen deberá contener horizontalmente, por término medio, al menos 1,5 puntos de pantalla (ver Figura 6.3a), con lo que cada punto de pantalla deberá ser menor que:

$$\frac{0,47 \text{ mm}}{1,5} = 0,32 \text{ mm}$$

- P6.4.** Obtener el valor de la capacidad de la memoria de cuadro de una pantalla de  $25 \times 80$  celdas, con resolución de color EGA, y atributos especiales de parpadeo, vídeo inverso, iluminación tenue y sobreiluminación.

#### SOLUCIÓN

Según se indicó en la Tabla 6.5, los adaptadores de vídeo EGA tienen una resolución de  $320 \times 200$  y una paleta de 4 colores. Por tanto, el número de palabras de la memoria será:

$$320 \cdot 200 = 6,25 \text{ Kpalabras}$$

Cada palabra debe tener los siguientes bits:

$$8 \text{ bits de código del carácter} + 2 \text{ bits de color} + 4 \text{ bits de atributos} = 14 \text{ bits}$$

Con lo que la memoria será de la siguiente capacidad:

$$C = 6,25 \cdot 15 = 87,5 \text{ Kbits} = 11 \text{ KBytes}$$

- P6.5.** La propuesta de televisión de alta definición requiere una resolución de  $1.280 \times 1.024$  puntos de imagen, con 256 niveles por cada uno de los colores básicos. Determinar la capacidad de la memoria de cuadro que se necesitaría para este sistema.

#### SOLUCIÓN

Cada color necesita una codificación de 8 bits ( $2^8 = 256$ ); es decir, cada palabra de la memoria debe contener:

$$8 \text{ bits/color} \times 3 \text{ colores} = 24 \text{ bits} = 3 \text{ Bytes}$$

Por otra parte el número de palabras de la memoria debe coincidir con el número de puntos de imagen; es decir:

$$C = 1.280 \times 1.024 \text{ palabras} = 1.280 \text{ Kpalabras de 3 Bytes} = 3.840 \text{ KBytes} = 3,75 \text{ MB}$$

- P6.6.** En la televisión de alta definición se pretende obtener en pantalla una resolución de  $1.280 \times 1.024$  píxeles. Sabiendo que para reproducir correctamente los colores fotográficos se necesitarían 16.777.216 tonalidades diferentes, estimar la capacidad en KBytes de la memoria de imagen que se necesitarían.

#### SOLUCIÓN

Para codificar los colores se necesitan:

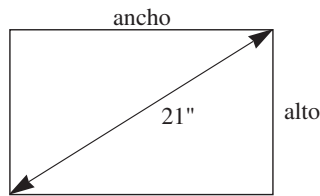
$$16777216 = 2^x$$

$$x = \log_2 16.777.216 = 3,32 \log 16.777.216 \approx 24 \rightarrow \text{se necesitan 24 bits}$$

$$1.280 \times 1.024 \text{ puntos} \times 24 \text{ bits/punto} = 1.280 \times 1.024 \times 24 \text{ bits} \times 1 \text{ Byte/8 bits} = 3.840 \text{ KBytes}$$

- P6.7.** Determinar la resolución,  $r$ , que tendría que tener una pantalla de 21 pulgadas para tener una densidad de puntos de imagen de  $\lambda = 300 \text{ d/i}$ .

## SOLUCIÓN



$$\text{Relación de aspecto } 4/3: \frac{4}{3} = \frac{\text{ancho}}{\text{alto}}$$

$$\text{Triángulo rectángulo: } \text{ancho}^2 + \text{alto}^2 = 21^2$$

$$\text{Alto} = 5,1''$$

$$\text{Ancho} = 6,8''$$

$$N_{\text{píxeles, columna}} = 300 \frac{\text{puntos}}{\text{pulgada}} \times 5,1 \text{ pulgadas} = 1.530 \text{ puntos}$$

$$N_{\text{píxeles, fila}} = 300 \frac{\text{puntos}}{\text{pulgada}} \times 6,8 \text{ pulgadas} = 2.040 \text{ puntos}$$

Con lo que la resolución será:  $2.040 \times 1.530$  píxeles.

## IMPRESORAS

**P6.8.** Una impresora tiene una memoria intermedia de 512 Bytes y escribe a 120 cps. Está conectada a un computador a través de una línea serie asíncrona que transmite a una velocidad de 1.200 bits/s. Teniendo en cuenta que a cada carácter (8 bits) se le añaden 4 bits adicionales de control, hacer una estimación del tiempo que se tarda en rellenar la memoria intermedia. ¿Cada cuánto tiempo tendría que rellenar la memoria intermedia el procesador?

## SOLUCIÓN

El número de bits a transmitir por carácter es:

$$N = 4 + 8 = 12 \text{ bits}$$

La capacidad de la memoria intermedia es de 512 B, pero para rellenarla hay que transmitir los bytes junto con los bits de control; es decir, el número de bits a transmitir es:

$$N_t = 512 \text{ B} \times 12 \text{ b/B} = 6.144 \text{ bits}$$

Entonces, el tiempo empleado en rellenar la memoria intermedia será:

$$T = \frac{6.400 \text{ b}}{1.200 \frac{\text{b}}{\text{s}}} = 5,12 \text{ s}$$

Por otra parte, el tiempo que tarda la impresora en imprimir todos los caracteres de la memoria intermedia será:

$$T = \frac{512 \text{ c}}{120 \frac{\text{c}}{\text{s}}} = 4,26 \text{ s}$$

Por tanto, cada 4,26 segundos hay que rellenar el *buffer*.

**P6.9.** Una impresora láser puede imprimir hasta 2,5 páginas por minuto (cada página contiene 40 líneas, de 80 caracteres cada una). La impresora está conectada a un computador a través de una línea de transmisión serie asíncrona. La comunicación se realiza añadiendo a cada carácter (8 bits) 3 bits de control para su adecuada transmisión. Determinar la velocidad de transmisión mínima en b/s desde el computador a la impresora para poder mantener la citada velocidad de impresión.

## SOLUCIÓN

El número de bits a transmitir por carácter es:

$$N_c = 8 + 3 = 11 \text{ b/c}$$



La velocidad de impresión de la impresora láser en caracteres por segundo se puede estimar en:

$$V_{\text{impresión}} = \frac{2,5 \text{ págs./m}}{60 \text{ s/m}} \cdot 40 \frac{l}{\text{pág.}} \cdot 80 \frac{c}{l} = 133,3 \frac{c}{s}$$

La velocidad mínima de transmisión desde el computador será, por tanto:

$$V_{\text{transmisión}} = 133,3 \frac{c}{s} \cdot 11 \frac{b}{c} = 1.466,6 \frac{b}{s} = 1,41 \frac{Kb}{s}$$

## PERIFÉRICOS MULTIMEDIA

**P6.10.** Se dispone de un adaptador SVGA (1.024 × 768 puntos de imagen, 256 colores) suponiendo que el único atributo de cada punto es el color:

- Hacer una estimación de la capacidad de la memoria de imagen.
- Con objeto de que la pantalla pueda utilizarse en un sistema multimedia que pretende visualizar imágenes en movimiento (25 Hz), indicar el ancho de banda del bus de datos de la memoria de imagen.

### SOLUCIÓN

- Resolución 1.024 × 768 puntos:

$$256 \text{ colores} = 2^8 \text{ colores} \rightarrow \text{se necesitan } 8 \text{ bits} = 1 \text{ Byte para codificar el color de cada punto}$$

$$768 \times 1.024 \text{ puntos} \times 1 \text{ Byte/punto} = 768 \times 1.024 \text{ Bytes} = 768 \text{ KBytes}$$

- Una frecuencia de 25 Hz quiere decir que se transfieren 25 imágenes por segundo:

$$768 \text{ KBytes/imagen} \times 25 \text{ imágenes/s} = 19.200 \text{ KB/s}$$

**P6.11.** Suponiendo que con una máquina de fotos digital<sup>8</sup> en un disquete convencional de 3½" (HD) se graban 20 fotografías, la resolución obtenida en cada foto es VGA (640 × 480 puntos de imagen), y la grabación de la imagen se efectúa utilizando el estándar de compresión JPEG de forma que la información codificada representa un 16,38 por 100 de la inicial, estimar el número de bits por punto de imagen que se utilizan para codificar el color.

### SOLUCIÓN

En el disquete una foto ocupa:

$$C_{\text{foto}} = \frac{1.44 \text{ MB}}{20 \text{ fotos}} = 0,072 \text{ MB/foto} \approx 75.497 \text{ Bytes}$$

Antes de comprimir una foto ocupa:

$$C'_{\text{foto}} = 640 \cdot 480 \cdot N \text{ bits} = 307.200 \cdot N \text{ bits} = 38.400 \times N \text{ Bytes}$$

Donde  $N$  es el número de bits para codificar el color del píxel. Por otra parte se debe verificar la siguiente relación:

$$75.497 = 0,1638 \cdot 38.400 \cdot N$$

Por tanto:

$$N = \frac{75.497}{0,1638 \cdot 38.400} \approx 12 \text{ bits}$$

**P6.12.** Una cámara digital de fotos realiza fotografías con una resolución de 2.272 × 1.704 píxeles. Las fotografías se almacenan en una tarjeta de memoria de 128 MB de capacidad usando el sistema de compresión de imágenes JPEG con una relación de compresión de 1 a 8.

- Después de almacenar en la memoria 92 fotografías nos quedan libres 636,125 KB en la memoria. ¿Cuántos bits de profundidad de color hemos utilizado en cada fotografía?

<sup>8</sup> SONY modelo Mavica MVC-FD5.

- b) La cámara dispone de una interfaz USB 1.1 para transferir las fotografías a un ordenador. ¿Cuánto tiempo tardará la transferencia de cada una de las fotografías suponiendo que la conexión USB está funcionando a su velocidad máxima (1,5 MB/s)?

## SOLUCIÓN

- a) Llamando  $n$  a la profundidad (n.º de bits utilizados para representar cada píxel) se tiene que cada foto ocupa:

$$C_{foto} = \frac{2.272 \times 1.704}{8} n = 483.936 n \text{ bits} = 0,4615 n \text{ Mbits} = 0,0576896 n \text{ MB}$$

Las 92 fotos ocuparán:

$$C_{92} = 0,0576896 n \times 92 = 5,3074432 n \text{ MB}$$

pero por otra parte sabemos que las 92 fotografías comprimidas ocupan en la memoria flash:

$$C_{92} = 128 \cdot 2^{10} \text{ KB} - 636,125 \text{ KB} = 127,38 = 5,3074432 n \Rightarrow n = 24 \text{ bits}$$

- b)  $C_{foto} = 0,0576896 n \text{ MB} = 0,0576896 \times 24 \text{ MB} = 1,3846 \text{ MB}$

con lo que tenemos que:

$$t = \frac{C_{foto}}{v_{transferencia}} = \frac{1,3846 \text{ MB}}{1,5 \text{ MB/s}} = 0,92 \text{ segundos}$$

- P6.13.** En un sistema multimedia se quiere transmitir por un módem una secuencia de imágenes de  $320 \times 200$  píxeles y 256 colores por punto imagen. La transmisión se ha configurado a una velocidad de 33.600 bps y añadiendo un bit de paridad con criterio par por cada byte de información.

- a) ¿Cuántas imágenes por segundo se pueden transmitir?  
 b) ¿Qué velocidad de transmisión se necesitaría si quisiéramos visualizar en el computador receptor 15 imágenes por segundo? (Nota: Suponer que no se transmite ninguna información adicional de control.)

## SOLUCIÓN

256 colores =  $2^8$  colores  $\Rightarrow$  el color se codifica con 8 bits = 1 Byte. La capacidad de una imagen es ( $p$  = píxel):

$$\begin{aligned} 320 \cdot 200 p \cdot 1 \text{ Byte/p} &= 320 \cdot 200 \cdot (8 \text{ bits/Byte} + 1 \text{ bit de paridad}) = \\ &= 320 \cdot 200 \cdot 9 \text{ bits} = 576.000 \text{ bits/imagen} \end{aligned}$$

En un segundo se transmiten:

$$\begin{aligned} \frac{33.600 \text{ b/s}}{576.000 \text{ b/imagen}} &= 0,058 \text{ imágenes/segundo} \\ 15 \text{ imágenes/seg} &= 15 \cdot 576.000 \text{ bits/s} = 8.640.000 \text{ bits/s} = 8.24 \text{ Mb/s} \end{aligned}$$

- P6.14.** En un monitor de 19" queremos obtener una resolución de  $1.280 \times 1.024$  píxeles de imagen. Además para cada uno de ellos tendremos 256 tonalidades distintas para cada uno de los tres colores básicos (rojo, verde y azul). Se desea visualizar en la pantalla una película DVD a 25 imágenes/segundo.

- a) Calcular el tamaño mínimo de la memoria de imagen que se necesita.  
 b) Obtener el ancho de banda mínimo requerido para el controlador de gráficos (AGP).

## SOLUCIÓN

- a) Para codificar 256 tonalidades de un color se necesita un Byte, y como se van a usar tres colores para completar una componente RGB, por cada píxel necesitaremos 3 Bytes. Así que si la pantalla tiene una resolución de  $1.280 \times 1.024$  píxeles, el total de Bytes necesarios para almacenar la imagen de pantalla será de:

$$1.280 \cdot 1.024 \text{ píxeles} \cdot 3 \text{ Bytes/píxel} = 3.932.160 \text{ Bytes} = 3,75 \text{ MBytes}$$

- b) Para poder visualizar la película adecuadamente necesitamos que el bus AGP sea capaz de transmitir 25 fotogramas por segundo, así que teniendo en cuenta el tamaño de un fotograma, calculado en el apartado anterior, tenemos que el ancho de banda necesario debe ser:

$$AB = 3,75 \text{ MBytes/fotograma} \cdot 25 \text{ fotogramas/segundo} = 93,75 \text{ MB/s}$$

**P6.15.** En un sistema multimedia se quiere transmitir una secuencia de imágenes de  $320 \times 200$  puntos imagen (píxeles) y 256 colores por punto imagen por un módem. La transmisión se ha configurado a una velocidad de 33.600 bps (bits por segundo) y añadiendo un bit de paridad con criterio par por cada Byte de información.

- a) ¿Cuántas imágenes por segundo se pueden transmitir?  
 b) ¿Qué velocidad de transmisión se necesitaría si quisiéramos visualizar en el ordenador receptor 15 imágenes por segundo? (Nota: Suponer que no se transmite ninguna información adicional de control.)

#### SOLUCIÓN

256 colores =  $2^8$  colores  $\Rightarrow$  el color se codifica con 8 bits = 1 Byte. La capacidad de una imagen es ( $p$  = punto de imagen):

$$320 \times 200 p \times 1 \text{ Byte}/p = 320 \times 200 \times (8 \text{ bits/Byte} + 1 \text{ bit de paridad}) = \\ = 320 \times 200 \times 9 \text{ bits} = 576.000 \text{ bits/imagen}$$

En un segundo se transmiten:

$$\frac{33.600 \text{ b/s}}{576.000 \text{ b/imagen}} = 0,058 \text{ imágenes/segundo}$$

$$15 \text{ imágenes/seg} = 15 \times 576.000 \text{ bits/seg} = 8.640.000 \text{ bits/s} = 8.24 \text{ Mb/seg}$$

#### PERIFÉRICOS PARA SEÑALES ANALÓGICAS

**P6.16.** Para el sistema de adquisición de datos del Ejemplo 6.8:

- a) Determinar el ancho de banda entre el sistema de adquisición y el computador.  
 b) El período mínimo de muestreo.  
 c) Indicar el motivo de por qué dispone de una memoria FIFO, en lugar de una RAM o LIFO.  
 d) El tiempo que se tardaría en completar la memoria FIFO suponiendo un muestreo continuo.

#### SOLUCIÓN

- a) Como la frecuencia máxima de muestreo es de 500 KS/s y cada muestra contiene 12 bits = 1,5 Bytes, el ancho de banda será:

$$AB = 500 \text{ KS/s} \cdot 1,5 \text{ Bytes/S} = 750 \text{ KB/s}$$

- b) El período mínimo de muestreo será:

$$T_s = \frac{1}{F_s} = \frac{1}{500 \text{ KS/s}} = 2 \mu\text{s}$$

Es decir, cada  $T_s = 2 \mu\text{s}$  se convertirá una muestra analógica en digital.

- c) Una señal analógica tiene sentido como una sucesión de valores consecutivos en el tiempo; es decir, que deben almacenarse y recuperarse en orden; con lo que resulta lógico su almacenamiento en estructuras FIFO: así las almacenamos en orden y las recuperamos en el mismo orden que han sido almacenadas.  
 d) Como la memoria FIFO tiene una capacidad de 4 KS, y en obtener una muestra se tardan  $2 \mu\text{s}$ , el tiempo en completarse dicha memoria será:

$$t = 4 \text{ KS} \cdot 2 \mu\text{s/S} = 8.192 \mu\text{s} = 8,2 \text{ ms}$$

**P6.17.** Un sistema de adquisición de datos analógicos de 16 canales se utiliza para captar otras tantas señales. Si la frecuencia de muestreo (para no perder información) de cada señal debe ser al menos de  $f_s = 2 \text{ KHz}$ , ¿cuál es el tiempo máximo de conversión posible para el conversor A/D?

## SOLUCIÓN

Para establecer el tiempo de conversión máximo permitido para el conversor debemos tener en cuenta que entre muestra y muestra de cada una de las señales de entrada debe haber sido capaz de haber convertido 16 muestras, ya que el multiplexor le irá presentando sucesivamente las siguientes muestras a convertir:

Muestra del canal 0 → Muestra del canal 1 → Muestra del canal 2 → Muestra del canal 3 →  
 Muestra del canal 4 → Muestra del canal 5 → Muestra del canal 6 → Muestra del canal 7 →  
 Muestra del canal 8 → Muestra del canal 9 → Muestra del canal 10 → Muestra del canal 11 →  
 Muestra del canal 12 → Muestra del canal 13 → Muestra del canal 14 → Muestra del canal 15 →  
 Muestra del canal 0 → Muestra del canal 1 → Muestra del canal 2 → .....

Obsérvese que el intervalo de tiempo entre dos muestras sucesivas del canal 0 (y de cualquier otro), debe ser:

$$T_s = \frac{1}{2.000 \frac{\text{muestras}}{s}} = 0,5 \text{ ms}$$

Y, según el esquema anterior, durante ese intervalo el conversor debe realizar 16 conversiones; luego el tiempo de conversión debe ser:

$$T_{\text{conversión}} \leq \frac{0,5 \text{ ms}}{16} = 31,25 \mu s$$

O sea, que la cadencia de conversión debe ser de:

$$F_s > \frac{1}{T_{\text{conversión}}} = 32 \text{ KHz}$$



## Problemas propuestos

## DETECTOR DE BARRAS IMPRESAS

**P6.18.** Un dentífrico fabricado en España utiliza para identificarse el sistema de barras EAN-13. El código del fabricante es 10372, el del producto 15400 y el dígito de verificación es 3. Si a la izquierda de los separadores centrales se utiliza el juego A y a la derecha el juego B (del sistema EAN-3), indicar las barras (negras o blancas) y su grosor que forman el identificativo, suponiendo que el primer dígito del código del estado no se representa.

## PANTALLAS

**P6.19.** Un terminal pantalla/teclado puede representar en imagen  $24 \times 80$  caracteres. Cada celda contiene  $7 \times 10$  puntos de imagen. Se pueden visualizar hasta 100 caracteres distintos.

- a) Indicar la longitud de palabra y número de palabras (mínimos) de la memoria de imagen, suponiendo que una celda puede tener hasta 28 combinaciones posibles de atributos distintos, y la que le corresponde se memoriza individualmente para cada celda.

- b) Indicar la memoria de imagen necesaria en los supuestos anteriores, en el caso de que la pantalla fuese gráfica y cada punto de imagen tuviese uno de 256 atributos.

**P6.20.** Obtener la capacidad de memoria de cuadro de un controlador gráfico UXGA.

**P6.21.** Supóngase que se desea disponer de un monitor con prestaciones Super-VGA para utilizar en una aplicación de edición de textos, con una pantalla que debe refrescarse a la frecuencia  $F_r = 74 \text{ Hz}$  (suponer que en la aplicación citada se requiere una frecuencia de actualización de cuadro de 5 cuadros/s).

- a) Si se utiliza un 12 por 100 del tiempo entre refrescos sucesivos para realizar la actualización de la memoria de imagen, ¿cuál debe ser la frecuencia de refresco mínima de la pantalla? Determinar los anchos de banda mínimos requeridos en la conexión con el bus de la computadora y con la pantalla.
- b) Suponiendo que se dispone de una memoria bipuerta, determinar los anchos de banda mínimos requeridos en la conexión de la memoria con el bus de la computadora y con la pantalla.

**IMPRESORAS**

**P6.22.** Un computador se comunica a través de una interfaz USB 2.0 (ancho de banda 480 Mbps) con una impresora láser que imprime a una velocidad máxima de 45 ppm, y tiene una memoria intermedia (*buffer*) de 48 MB. Hacer una estimación del tiempo que:

- a) Tarda el procesador en rellenar la memoria intermedia de la impresora.
- b) Transcurre entre dos solicitudes de interrupción sucesivas por parte de la impresora para solicitar que se vuelva a rellenar su memoria intermedia.

**P6.23.** Una impresora láser tiene una memoria intermedia (*buffer*) de 1 MB y escribe a una velocidad de 6 páginas por minuto. Suponga que la CPU funciona a una velocidad de 1 MIP y que la velocidad de transferencia de datos con la impresora fuese de 128 KB/s.

- a) Estimar el tiempo que se tarda en rellenar la memoria intermedia de la impresora.

- b) Estimar el número de instrucciones que podría ejecutar en paralelo la CPU mientras se imprime una página.

**PERIFÉRICOS MULTIMEDIA**

**P6.24.** En el manual de una cámara de vídeo para PC se especifican las siguientes características:

- Resolución máxima:  $352 \times 288$  píxeles.
- Velocidad de transferencia: hasta 30 cuadros por segundo.
- Máxima profundidad de color: 16 bits.
- Compresión AVI.
- Interfaz: USB 1.1.

Obtener el factor de compresión (porcentaje de reducción de la imagen), suponiendo que los parámetros anteriores actúan en sus valores máximos. (*Nota.* La velocidad máxima de transferencia de la interfaz USB 1.1 es de 1,5 MB/s.)

# Estructuras de computadores

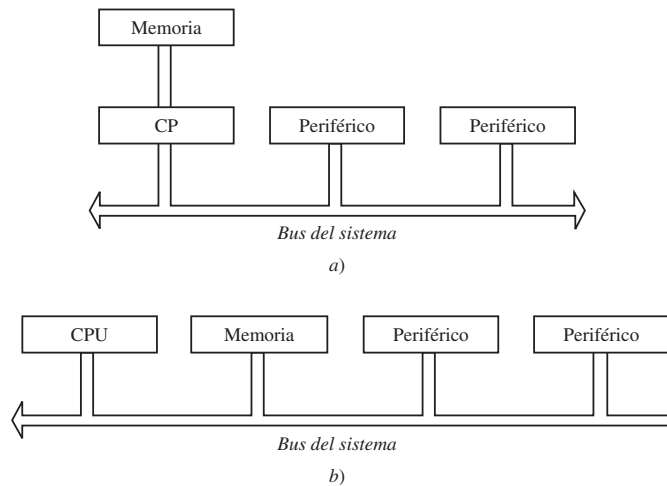
En los capítulos anteriores se han estudiado los diversos elementos con los que se configura un computador: procesador, memoria interna, memoria externa y periféricos de E/S. El presente capítulo se dedica a analizar cómo se interconectan unos con otros para formar un computador que funcione eficientemente. El problema es complejo, ya que las distintas unidades presentan peculiaridades muy diversas, como velocidades de funcionamiento muy divergentes, distintos niveles de tensiones o corrientes para representar los valores lógicos 0 y 1, etc. El capítulo se inicia (Sección 7.1) con una descripción de las estructuras básicas de interconexión. Posteriormente (Sección 7.2) se analizan sucintamente los tipos de buses estandarizados más usados. A continuación (Sección 7.3) tratamos de aplicar los conceptos vistos con anterioridad a un ejemplo concreto, y qué mejor elección, dada su popularidad, que un PC compatible. El capítulo concluye (Sección 7.4) con una sección dedicada al paralelismo en computadores.

## 7.1. Estructuras básicas de interconexión

Las unidades funcionales de un computador se interconectan de acuerdo con una determinada organización. Hay diferentes posibilidades, las más relevantes de las cuales se describen a continuación.

Las formas más sencillas de interconexión de unidades son las mostradas en la Figura 7.1. La primera forma (Figura 7.1a) dispone de un bus específico de interconexión procesador-memoria. Obsérvese que todo el tráfico de información entre periféricos y memoria necesariamente ha de hacerse a través del procesador. La segunda configuración (Figura 7.1b) es más sencilla, y utiliza un único bus (**estructura unibus**). Todas las unidades se conectan a él, y coincide con la estructura del computador CODE-2 analizado en el Capítulo 4. Este bus se denomina **bus del sistema**. Como en un instante dado sólo puede transmitirse una información por el bus, sólo una unidad (el procesador) puede tener el control del bus del sistema. La ventaja principal de esta estructura es su bajo coste y su flexibilidad para conectar periféricos.

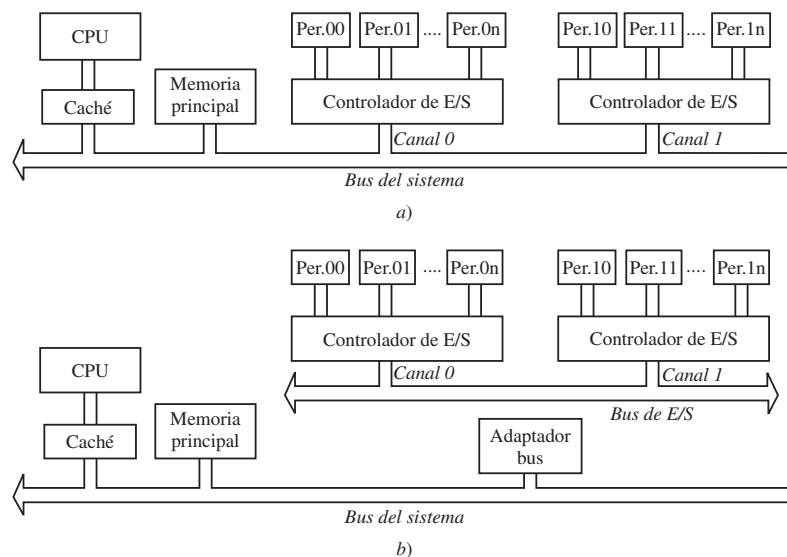
El esquema unibus tiene el gran inconveniente de que, en toda transferencia a través del bus, el elemento más lento es el que impone la velocidad de transmisión. Para reducir los efectos de la gran diferencia de velocidades entre el procesador y los periféricos, cada uno ellos contiene una **memoria intermedia** o *buffer*, que almacena la información durante la transferencia. El procesador puede cargar el *buffer* a alta velocidad, y el periférico, a su ritmo, grabar (caso de una unidad de disco) o im-



**Figura 7.1.** Organizaciones sencillas de un computador: a) con un bus específico para conexión procesador memoria y b) estructura de bus único.

primir (caso de una impresora) la información existente en el *buffer*. Soluciones complementarias son el uso de controladores de E/S y de controladores de acceso directo a memoria.

Un **controlador** (o **canal** o **procesador**) **de entradas/salidas** (o de **periféricos**) es un procesador con un amplio *buffer* y especializado en controlar las operaciones de transferencia de datos entre los periféricos conectados a él, y entre éstos y el procesador (Figura 7.2). Estas operaciones de control en principio son tarea del procesador, por lo que los controladores de E/S descargan a éste de las mismas: el procesador puede seguir trabajando con la memoria mientras los periféricos concluyen sus operaciones. No obstante, siempre es el procesador quien inicializa y cede el control al controlador, *programándolo* (en un puerto del controlador escribe una palabra de control indicando el tipo de transferencia —lectura/escritura—, los periféricos involucrados en la transferencia y el tamaño del bloque de datos). Una vez que un controlador de E/S finaliza la operación encomendada por el procesador, envía a éste una señal de interrupción indicando que está listo para realizar otra operación. Cada controlador de entradas/salidas monitoriza la actuación de distintos periféricos (Figura 7.2) y su conexión con el bus del sistema sue-



**Figura 7.2.** Configuraciones con controladores de E/S: a) de bus único y b) con un bus específico de E/S.

le denominarse **canal de entrada/salida** o sencillamente **canal**. En la Figura 7.2a se muestra una estructura con un bus único que incluye memoria caché y controladores de E/S. Esta estructura de bus único (como la de la Figura 7.1b) tiene el inconveniente de que una orden de transferencia de datos entre periféricos de dos canales distintos puede interferir con, por ejemplo, la captación de una instrucción de la memoria principal por el procesador. Para solventar este problema usualmente se utiliza una estructura con dos buses, tal y como se muestra en la Figura 7.2b, en la que aparece un bus específico de E/S.

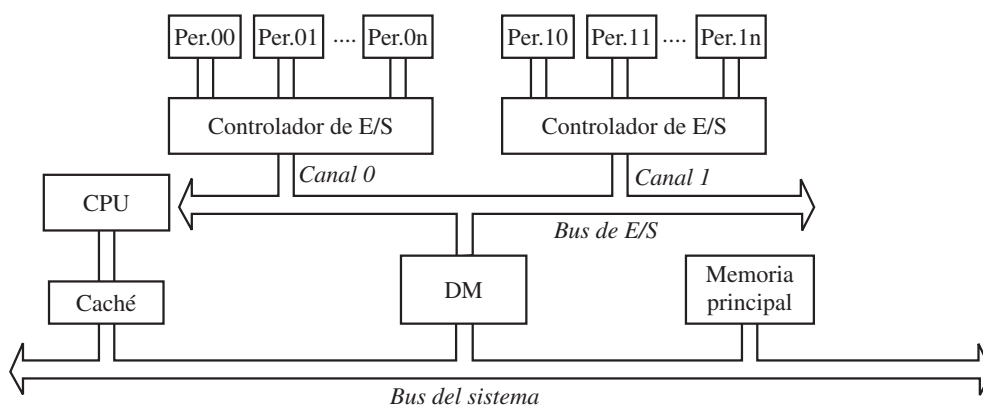
Los controladores de E/S evitan que el procesador tenga que adaptarse entre operaciones individuales de E/S a la velocidad y ritmo que marca el periférico con el que se hace la transferencia. No obstante, el procesador sigue siendo responsable de dichas operaciones. La mayoría de las operaciones de E/S se hacen entre la memoria y un periférico, y viceversa; como ocurre, por ejemplo, en la carga en memoria de un programa para ejecutarlo, o al salvar en disco un documento generado con ayuda de un procesador de textos. Pero, como el control del bus principal del sistema lo lleva el procesador, y las instrucciones de E/S de los lenguajes máquina comúnmente realizan estas operaciones entre el periférico y uno de los registros del procesador, en cada operación elemental de E/S debe intervenir el procesador.

### EJEMPLO 7.1

La transferencia de un bloque de 1.024 palabras de disco a memoria, necesitará un mínimo de 1.024 cargas en un registro del procesador y 1.024 almacenamientos en memoria, sucesivos. En el caso de CODE-2, supóngase que el puerto de entrada de datos asociado a una unidad de disco fuese  $IP(4)$ , y utilizásemos el registro  $rC$  del procesador para realizar la transferencia, habría que ejecutar las dos siguientes instrucciones tantas veces como datos hubiese (suponemos que en  $rI$  inicialmente hay almacenado 0001):

```
IN rC,IP4      ; Llevar dato de disco a procesador (registro rC).
ST v,rC        ; Almacenar en  $M(v + rD)$  el dato.
ADDS rD,rD,rI  ; Incrementar en 1 el puntero de memoria.
```

Para descargar al procesador de este trabajo se han ideado los **controladores DMA** (*Direct Memory Access*, **acceso directo a memoria**) (Figura 7.3). El controlador DMA (**DMAC**) es un procesador especializado que permite transferir datos entre memoria y un periférico (y viceversa) mientras el procesador puede realizar otras tareas. El DMAC es externo al procesador y actúa, combinadamente con el procesador, como controlador del bus. Cuando se hace una transferencia de un bloque de información controlado por DMA, previamente el procesador (a través del bus del sistema) programa al controlador inicializando un puerto de control del DMAC en el que se especifica: el tamaño del blo-



**Figura 7.3.** Estructura de un computador con DMA y canal específico de E/S.



que a transferir, la dirección inicial del bloque a leer en la unidad de origen, la dirección inicial donde se escribirá el bloque en la unidad de destino y el sentido de la transferencia (lectura o escritura). Una vez inicializado el DMAC, el procesador cede el control del bus al DMAC, encargándose éste de controlar la transferencia del bloque de información. Mientras tanto el procesador puede realizar otras tareas. Una vez finalizado el transvase del bloque, el DMAC envía a través del sub-bus de control una señal de interrupción al procesador, para que éste vuelva a tomar el control del bus del sistema.

Existen distintos buses normalizados, los más conocidos se describen en la siguiente sección.

## 7.2. Buses

Según indicamos en las Secciones 1.2.1, 3.1.3.6 y 7.1, los distintos elementos de un computador se interconectan por medio de buses, que proporcionan un camino de comunicación para el flujo de datos, direcciones y señales de control/estado entre los distintos elementos. Se presenta el problema de que cuando se comunican dos elementos a través de un bus, el más lento establece la velocidad de transmisión, y para solucionarlo y obtener un buen rendimiento en el funcionamiento global del computador resulta lógico que:

- Se utilicen distintos buses en un mismo sistema, pudiéndose así realizar transmisiones simultáneamente entre distintos elementos a través de buses diferentes.
- Se establezca una jerarquía de buses, en función fundamentalmente de la velocidad.
- Cuando la velocidad de transferencia a un periférico no es elevada se utilice una conexión serie (transmisión bit a bit), y cuando no sea así se utilice una conexión paralelo (varios hilos conductores que transmiten simultáneamente 8, 16, 32, 64 o 128 bits, dependiendo del tipo de bus).

Los circuitos integrados (chips) de cada subsistema se interconectan por medio de **tarjetas de circuito impreso** o **PCB** (*Printed Circuit Board*). Estas tarjetas son rectangulares, de un material aislante rígido y plano, y contienen las interconexiones eléctricas realizadas por medio de líneas (pistas) metálicas (usualmente de cobre) que van trazadas en las dos superficies de la tarjeta y en capas internas paralelas a su superficie. La tarjeta tiene orificios donde se insertan los distintos circuitos integrados, y regletas de conectores para la interconexión de otras tarjetas o elementos externos. Las conexiones eléctricas de los distintos elementos incluidos se realizan con soldaduras de estaño a las pistas de conducción (buses) correspondientes. En las regletas se insertan otras tarjetas (**tarjetas de expansión**) con las que la conexión eléctrica se hace por presión, de aquí que los contactos de cada regleta y de la tarjeta insertada en ella suela ser de un material conductor de muy baja resistividad (aleaciones con cobre o plata o incluso oro). En los equipos grandes las tarjetas se instalan en un chasis conectándose unas con otras por la parte posterior del mismo (**panel trasero**).

En general hoy día suelen considerarse los tipos de buses que se indican en la Tabla 7.1.

- |  |
|--|
| <ul style="list-style-type: none"> <li>• Buses internos a los circuitos integrados.</li> <li>• Bus delantero (FSB: <i>Front Side Bus</i>): une el procesador con la memoria y el chipset.</li> <li>• Buses locales para interconexión de elementos de una PCB.</li> <li>• Bus del panel posterior: conexión entre las PCB dentro de un mismo chasis.</li> <li>• Buses de expansión, para interconexión de subsistemas (unos 10 m).</li> <li>• Buses de entrada/salida, para periféricos:               <ul style="list-style-type: none"> <li>— Paralelo.</li> <li>— Serie.</li> </ul> </li> </ul> |
|--|

**Tabla 7.1.** Jerarquía de buses en un computador.

Unos buses con otros se interconectan por medio de adaptadores o puentes; éstos pueden ralentizar el funcionamiento de los periféricos, de manera que a la hora de configurar un sistema es necesario hacer un estudio de velocidades tanto de los distintos subsistemas que lo integran como de la transferencia de información en los buses.

Existen definiciones normalizadas o estandarizadas de buses, cuyas especificaciones puede utilizar el ingeniero a la hora de diseñar un computador. Estas normalizaciones especifican cuestiones en los siguientes niveles:

- **Nivel mecánico:** Soporte (chasis, circuito impreso, etc.), número de líneas, tipos de conectores, dimensiones de las tarjetas, etc.
- **Nivel eléctrico:** Alimentación eléctrica, impedancias, niveles de las señales de tensión o de corriente, etc.
- **Nivel lógico:** Número de señales, niveles de tensión o corriente correspondientes al 0 o 1 lógico, etc.
- **Nivel de temporización:** Intervalos de tiempo correspondientes a cuando son válidos los datos, direcciones o señales de control, etc.
- **Niveles de transferencias:** Protocolos de arbitraje, detección de errores, etc.

Algunas de las normalizaciones de buses más conocidos son (Tabla 7.2):

**Bus PC/XT.** Fue introducido en 1981 con el IBM-PC y define 62 hilos, constituidos por 3 de tierra, 5 de tensiones de alimentación, 20 líneas de dirección, 8 líneas de datos, 10 líneas para señales de interrupción y una gran variedad de líneas para usos específicos. Es un bus estrictamente para arquitecturas de 8 bits (esto es, datos de 8 bits).

**ISA Bus.** Fue introducido en 1984 con los IBM-PC AT (microprocesador 80286). Está ideado para arquitecturas de 16 bits, y los conectores que se insertan en él son de 98 contactos. El sub-bus de di-

Nombre	Anchura de datos	Velocidad máxima de transferencia
<b>ISA (Bus-PC)</b>	8 bits	< 1,2 MB/s.
<b>ISA-AT</b>	16 bits	5,33 MB/s.
<b>EISA</b>	32 bits	33 MB/s.
<b>VLB</b>	32, 64 bits	160 MB/s; 250 MB/s.
<b>PCI</b>	32, 64 bits	132 MB/s; 264 MB/s.
<b>PCI 2.0</b>	32, 64 bits	264 MB/s; 528 MB/s.
<b>PCI-X 2.0</b>	32, 64 bits	2,1 MB/s; 4,2 GB/s.
<b>QuickRing</b>	32 bits	350 MB/s.
<b>VME</b>	16 a 23 bits	27,9 MB/s.
<b>Multibus II</b>	32 bits	40 MB/s.
<b>Futurebus+</b>	64, 128, 256 bits	95,2 MB/s, 3,2 GB/s.
<b>SCSI</b>	8 bits	5 MB/s.
<b>SCSI-2</b>	8/16 bits	10 MB/s/20 MB/s.
<b>Ultra SCSI</b>	8/16 bits	20 MB/s/40 MB/s.
<b>Ultra640 SCSI</b>	16 bits	640 MB/s.
<b>ATA-1 (IDE)</b>	16 bits	3,3; 5,2; 8,3 MB/s.
<b>ATA-2/3 (EIDE)</b>	16 bits	13,3; 16,7 MB/s.
<b>ATA-7 (Ultra ATA-133)</b>	16 bits	133 MB/s.
<b>Port AGP 1x</b>	32 bits	264 MB/s.
<b>Port AGP 12x</b>	32 bits	2,98 GB/s.
<b>FireWire (IEEE 1394)</b>	1 bit	96; 192 y 384 Mbits/s.
<b>FireWire (IEEE 1394b)</b>	1 bit	800 Mbits/s; 1,6 Gbits/s y 3,2 Gbits/s.
<b>USB 1.0</b>	1 bit	1,5 Mbits/s; 12 Mbits/s.
<b>USB 2.0</b>	1 bit	1,5; 12 y 480 Mbits/s.
<b>PCI-Express</b>	1 bit	128 Gbits/s.

**Tabla 7.2.** Comparación de características de varios buses.

recciones es de 24 bits (puede direccionar hasta 16 MB) e inicialmente podía transferir información a velocidades de hasta 2 MB/s. Es compatible con el bus ISA inicial (es decir, admite tarjetas de 8 bits, además de las de 16 bits).

**MCA** (*Micro-Channel Architecture*). Fue introducido por IBM en 1987 en sus equipos PS/2. Aumenta la velocidad de los bus ISA hasta 20 MB/s, y es para arquitecturas de 32 bits. A diferencia de los buses citados anteriormente y el EISA, es un bus síncrono de forma que las transferencias de información se hacen de forma sincronizada con el reloj del sistema. La especificación inicial se proyectó para velocidades de transferencia de 160 MB/s.

**EISA** (*Extended Industry Standard Architecture*). Es un bus ideado por nueve fabricantes de computadores, para arquitecturas de 32 bits (direcciones y datos). La velocidad de transferencia que se puede conseguir es de 33 MB/s. Admite tarjetas de tamaño doble y de consumo de energía eléctrica también doble que las ISA AT y las MCA, y por tanto las ampliaciones son más baratas, ya que la densidad de componentes en la tarjeta puede ser menor, y el consumo de los chips mayor. A diferencia del bus MCA, es compatible con las tarjetas ISA; es decir, pueden conectarse directamente en las ranuras de un bus EISA tarjetas ISA. Este bus sólo puede ser controlado por los microprocesadores 80386, 80486 o Pentium, y es autoconfigurable.

**Buses locales.** Las tres definiciones más conocidas son: **VLB** (ideada por VESA, un consorcio de más de 120 empresas), **PCI** (*Peripheral Component Interconnected*, promovido por Intel) y **Quick-Ring** (propuesta por Apple). Los equipos con VLB y PCI, disponen de tan sólo tres o seis ranuras, respectivamente, ubicadas en las proximidades del procesador, en la propia placa base, para introducir los controladores correspondientes.

**Bus AGP** (*Accelerated Graphics Port*). Se introdujo en 1997 por Intel junto con el Pentium II. Está especializado en hardware de gráficos, y trata de aumentar el ancho de banda entre procesador-memoria y el sistema de vídeo, consiguiéndose acelerar la generación de imágenes 3D, la reproducción en pantalla de vídeo en tiempo real y liberar en parte al bus PCI del trabajo con gráficos.

**Futurebus+** (IEEE 896.1 e IEEE 896.2). Es una normalización proyectada para equipos de muy altas prestaciones, que puede considerarse como una evolución de las normas Multibus II y VME. Detalla las características de un panel posterior, para arquitecturas de 64 bits o superiores. Permite la construcción de sistemas multiprocesador (de hasta 32 procesadores) compartiendo memoria.

**ATA** (*Advanced Technology Attachment*) o **IDE** (*Integrated Drive Electronics*). Es un bus o interfaz específica para discos duros. Permite que el controlador de disco forme parte de la propia unidad de disco, siendo así la comunicación entre el bus y la unidad de disco digital (y no analógica, como ocurría con anterioridad). La unidad de disco se conecta directamente a la placa base por medio de una versión del mismo bus en forma de cinta o banda flexible que contiene en su interior los hilos conductores. En la versión inicial cada controlador admitía la conexión encadenada de dos unidades, cada una de ellas de hasta 528 MB.

**ATA-2** o **EIDE** (*Enhanced Integrated Drive Electronics*). Es una versión mejorada de IDE, que admite hasta cuatro dispositivos de almacenamiento intercambiables o no (discos duros, CD-ROM y cintas). Cada disco puede llegar a tener una capacidad de 8,4 GB y una velocidad de transferencia de hasta 16,6 MB/s. Se han desarrollado muchas versiones posteriores (ATA 2/3, ATA 4, ATA 5, ATA 6, etcétera). ATA 6 (también denominado Ultra ATA-100 o Ultra DMA-100) logra velocidades de transferencia máxima de 100 MB/s, y dispone cables de 80 hilos y conectores de 40 contactos.

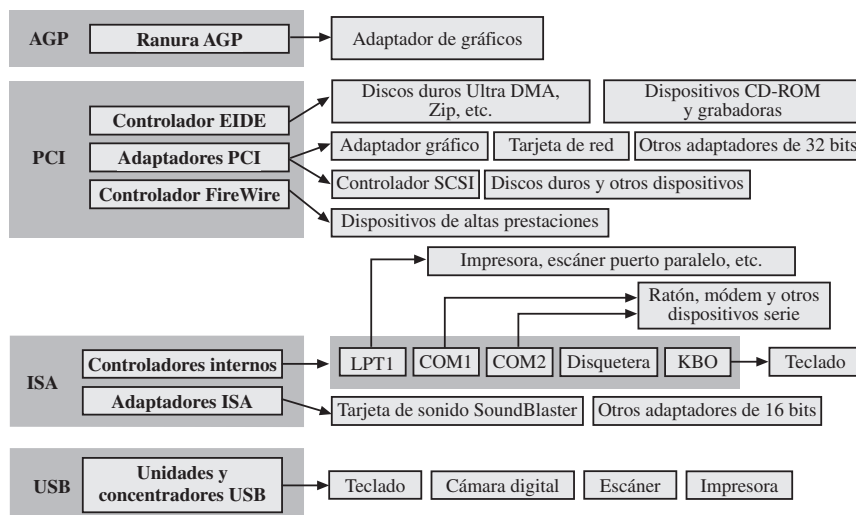
**SCSI** (*Small Computer System Interface*, interfaz de sistema para pequeñas computadoras). Es un estándar universal para la conexión paralela de una gran variedad de tipos de periféricos. Sue-

le utilizarse para unidades de discos magnéticos y ópticos, disquetes, impresoras y escáneres. El bus SCSI-1 admite hasta 7 dispositivos y fue ideado para entornos UNIX y Macintosh, permitiendo velocidades de transferencia de 5 MB/s, siendo el bus de datos de 8 bits (*narrow SCSI*). La versión SCSI-2 es para datos de 16 bits (*wide SCSI*), con velocidades de hasta 20 MB/s, y admite la conexión de hasta 15 dispositivos. La normalización SCSI-3 admite velocidades de transferencia del doble, 40 MB/s, y permite la conexión de hasta 127 dispositivos. Posteriormente se han desarrollado numerosas versiones, como la Ultra640 SCSI con velocidades de transferencia de 640 MB/s.

**USB** (*Universal Serial Bus*). Es un sistema de interconexión serie de muy bajo costo, ideado para gran cantidad de dispositivos periféricos de muy diversas características, y muy cómodo de utilizar con funcionamiento enchufar-y-funcionar (*plug-and-play*). Los cables y conectores de conexión incluyen tan sólo cuatro hilos: dos de transmisión de datos y dos de alimentación, no siendo necesario por lo tanto que el periférico tenga alimentación eléctrica independiente. La versión USB 1.0 admite dos velocidades: baja-velocidad (1,5 Mbits/s) y velocidad-completa (12 Mbits/s) y con él se pueden conectar encadenados hasta 127 dispositivos lentos, tales como ratón, cámaras de fotos, teclado, impresoras, escáneres de baja velocidad y unidades de disquetes. USB 2.0 admite alta-velocidad (480 Mbits/s).

**FireWire** (IEEE 1394), o *bus serie de altas prestaciones* (*High Performance Serial Bus*). Es un bus serie, alternativa a SCSI, al que se pueden conectar en el mismo puerto, en cadena, hasta 63 dispositivos y con el que se logran velocidades de transferencia máximas de 96 Mbits/s (versión S100), 192 Mb/s (S200) y 384 Mb/s (S400). Puede integrar equipos multimedia tanto de audio como de vídeo. Una de sus líneas es de suministro de energía eléctrica (60 vatios) eliminando la necesidad de utilización del cable de conexión a la red en dispositivos de bajo consumo. Debido a su bajo costo y sencillez, se utiliza, además de en computadores de uso general, en sistemas embebidos tales como televisiones de altas prestaciones, cámaras digitales, etc. La versión 1394b admite tres velocidades: 800 Mb/s (S800), 1,6 Gb/s (S1600) y 3,2 Gb/s (S3200).

En la Figura 7.4 se muestra un resumen de los distintos tipos de buses descritos, y su ámbito de aplicación.

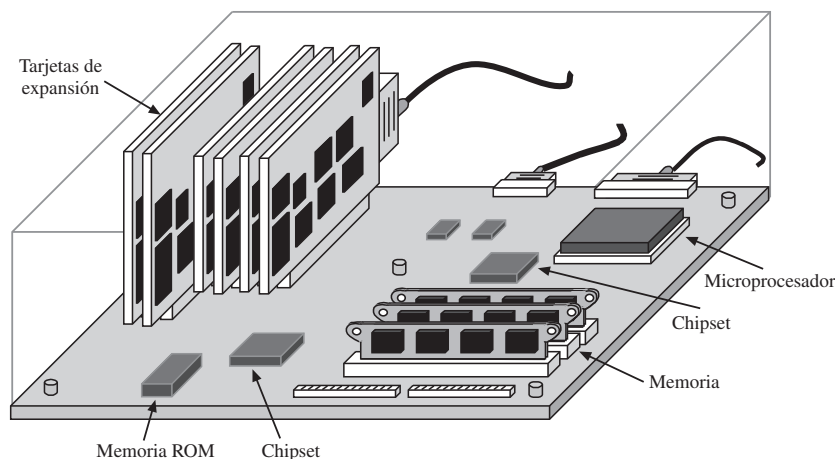


**Figura 7.4.** Buses normalizados y ámbitos de utilización.

### 7.3. Ejemplo de computador: un PC compatible

A continuación, y con objeto de perfilar mejor los conceptos vistos en las secciones anteriores, vamos a examinar brevemente la estructura interna de un PC compatible.

La mayor parte de los elementos de un computador personal (salvo los dispositivos de entrada/salida) se ensamblan dentro de un chasis. El sistema principal de interconexión es la **tarjeta madre** o **placa base**, que es una tarjeta de circuito impreso (PCB) donde están trazados los buses principales y donde se insertan los dispositivos electrónicos principales del computador y las regletas para conexión de tarjetas de expansión (Figura 7.5).



**Figura 7.5.** Vista general de la tarjeta base, y los elementos insertados en ella, de un PC compatible.

En los PC actuales los componentes de control principales en la tarjeta base son el microprocesador y dos o tres circuitos denominados genéricamente *chipset*. Los **chipset** son unos circuitos integrados (1, 2 o 3) que agrupan funciones diversas de la placa base. Dependiendo del tipo de microprocesador contienen el **reloj de tiempo real** y diversos adaptadores (puente PCI, por ejemplo) y controladores (de memoria, DMA, de teclado, del ratón, del acelerador de gráficos, de la interfaz USB, etc.). Estos circuitos determinan la velocidad máxima del bus de memoria y de otros circuitos externos al microprocesador.

En los PC se utilizan distintas tecnologías de memorias RAM que se diferencian fundamentalmente en la velocidad y capacidad, las más corrientes son: FPM, EDO, BEDO, SDRAM, RDRAM y Direct DRAM. Comercialmente las memorias se suministran en pequeñas tarjetas de circuito impreso o **módulos de memoria** que agrupan varios circuitos integrados y que se insertan directamente en las regletas de conectores apropiadas de la tarjeta base. Las tres agrupaciones más conocidas son:

- **SIMM** (*Single In-line memory Module*). Que pueden contener 8 chips de 32 o 64 Mbits cada uno totalizando una módulo de 32 o 64 MB, respectivamente. Hay versiones con conectores de 30 o 72 contactos, según sea de 8 o 32 bits el ancho de banda (Figura 7.6a).
- **DIMM** (*Dual In-line Memory Module*). Pueden almacenar 64 o 128 MB e incluso capacidades superiores. Los contactos están por las dos superficies de la tarjeta, teniendo 84 por cada lado (168 en total (Figura 7.6b). En un instante dado es capaz de leer o escribir datos de 64 bits (ancho del bus de datos de 64 hilos).
- **RIMM** (*Rambus In line Memory Module*). Son como los DIMM, pero tienen una asignación de conectores distinta, y se usan como módulos de las memorias Direct RDRAM.



interfaz flash BIOS, control del interfaz PCI, control de CODEC (compresor/decompresor de señales de audio y vídeo), etc.

- **Memoria SDRAM:** Módulos DDR (*Double-Data-Rate*) de 64 o 128 o 256 o 512 MB o 1 GB. Memoria máxima 2 GB. Permite capturar datos de 128 bits a una frecuencia de 400 MHz.
- **Acelerador de gráficos (AGP 8x):** Conector para tarjeta aceleradora de cambios de imágenes en pantalla de altas prestaciones, bus de 32 bits, frecuencia 66 MHz, y por ciclo se transmiten 8 palabras. Velocidad de transferencia: 2.112 MB/s.
- **Conectores PCI:** Cinco ranuras PCI de 32 bits, 33 MHz (132 MB/s), para insertar elementos tales como: tarjeta de vídeo, tarjeta de red, tarjeta de captura de vídeo (TV) y disco duro (SCSII).
- **Ultra DMA 33/66/100 Bus IDE:** Dos conectores que admiten 4 dispositivos IDE en dos canales.
- **Serial ATA (SATA):** El chipset IC5 dispone de una salida (150 MB/s) para dos puertos con funcionamiento independiente con DMA.
- **ATA-IDE:** El chipset IC5 dispone de una salida ATA 33/66/100 con dos canales que admiten DMA y cada uno cuatro dispositivos IDE (con controlador interior) tales como disco duro (HD), unidad de CD-ROM, unidad grabadora de CD-ROM y unidad de DVD-ROM.
- **CODEC AC97:** Que permite la inclusión en la placa base de un sistema de control completo de grabación y audición de audio.
- **USB 2.0:** Cuatro puertos en el panel trasero y dos interfaces USB 2.0 en la tarjeta, posibilitando la instalación de hasta ocho interfases USB 2.0 para conectar dispositivos tales como, teclado, ratón, módem, escáner, etc.
- **Disquetera:** conector para unidad de disquetes.
- **Conectores del panel frontal:**
  - Conexión para altavoz.
  - Interruptor para reiniciar el arranque del PC (*Reset*).
  - Indicador luminoso de encendido (*POWER LED*).
  - Indicador luminoso de disco duro en funcionamiento (*HD LED*).
  - Entrada para infrarrojos.
  - Indicador de que el sistema está con alimentación eléctrica pero en reposo; es decir, en estado suspendido (*Sleep*).
  - Interruptor para encendido (*Power On*).

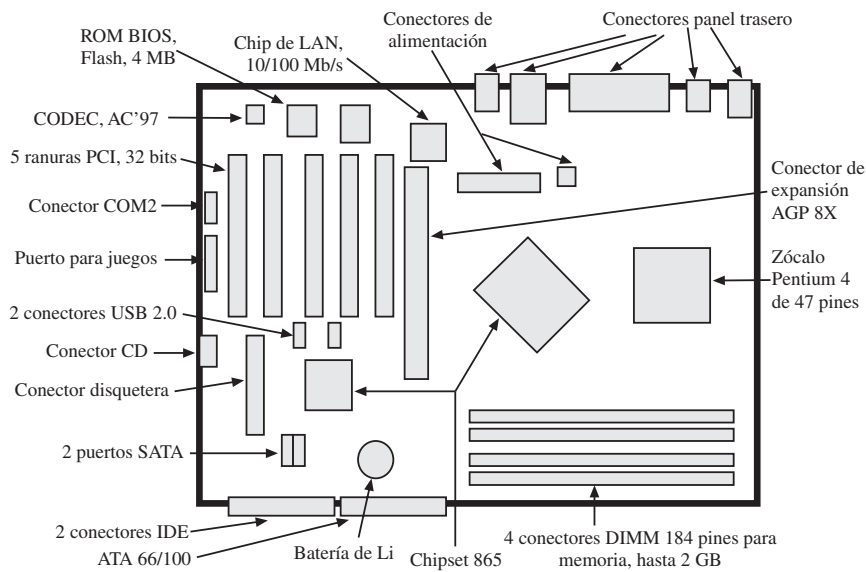
Además de los elementos anteriores, dentro del chasis se incluye una serie de ventiladores para que la temperatura de los circuitos integrados no supere un determinado valor, y una fuente de alimentación que transforma la corriente alterna de suministro de energía eléctrica en corriente continua a los niveles de tensión que requieren los distintos circuitos y elementos internos del chasis; éstos en un Pentium IV son: +5V (20A), -5V (0,5A), 12V (8A), -12V (0,5A) y 3,3V (14A); en donde entre paréntesis se indican los valores máximos de la corriente suministrada por cada una de las fuentes. La alimentación del PC requiere una tensión alterna dentro de los límites de 200-240V y consume del orden de 2,5A.

En la Figura 7.8 se muestra un esquema de cómo se ubican los distintos elementos en la tarjeta base.

## 7.4. Paralelismo en computadores

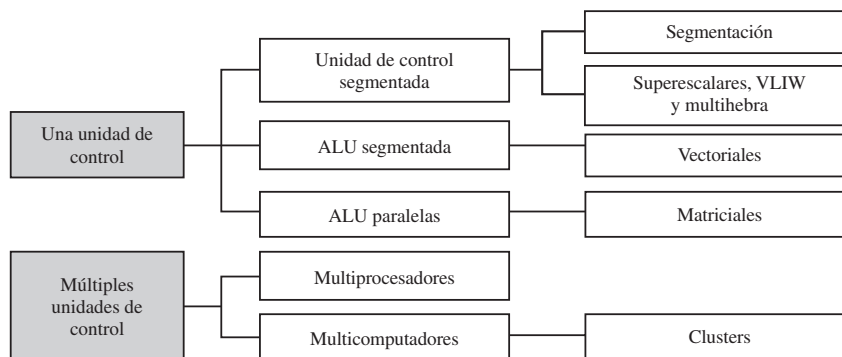
En esta sección se incluyen conceptos sobre arquitectura de computadores que han mejorado notablemente las prestaciones de los mismos y que se fundamentan en la utilización de paralelismo. Los





**Figura 7.8.** Esquema general de la distribución de los conectores de una tarjeta base de un PC compatible (AOpen i865).

aumentos de rapidez y potencia en los computadores se han y están logrando no sólo gracias a las mejoras tecnológicas (aumento de velocidad y de miniaturización en los circuitos integrados), sino además por la introducción de nuevos conceptos y diseños relacionados con la *arquitectura* del computador. Las mejoras de prestaciones debidas a la arquitectura se han conseguido fundamentalmente duplicando o segmentando adecuadamente los elementos o unidades del procesador de forma que se puedan ejecutar varias instrucciones a la vez, conociéndose el conjunto de estas técnicas como **procesamiento paralelo**. El paralelismo se ha introducido en muy diversas formas, como se indica en la Figura 7.9 y a continuación se describen.



**Figura 7.9.** Diversas formas de paralelismo.

### Paralelismo a nivel de instrucciones

Una primera forma de obtener paralelismo es entre instrucciones (**ILP**, *Instruction Level Parallelism*). Las técnicas que se aplican tratan de reducir el número de ciclos por instrucción que por término medio dedica el procesador a ejecutar una instrucción (CPI), con lo que, según la expresión [1.2], se reduce el tiempo de ejecución de los programas.



Como hemos visto en capítulos anteriores, la ejecución de una instrucción individual se realiza en varias etapas, y cada una de ellas consume un ciclo de reloj, de forma tal que es habitual que las instrucciones más sencillas consuman del orden de 3 a 6 ciclos. Estas etapas pueden ser, por ejemplo (Figura 7.10a): captación de instrucción (*F*), decodificación de instrucción y captación de operandos (*D*), ejecución de la operación especificada por la instrucción (*E*) y almacenamiento del resultado (*W*). No necesariamente todas las instrucciones hacen uso de estas cuatro etapas.

Una forma de acelerar el funcionamiento del procesador es diseñar la unidad de control de forma modular con diversos segmentos o módulos (cuatro en el ejemplo), de forma que cada uno de ellos esté especializado en la implementación de cada una de las etapas en que se descompone la ejecución de cada instrucción (Figura 7.10b). Con este sistema mientras un segmento se encarga de implementar la etapa que le corresponda de una instrucción, los otros pueden dedicarse a realizar otras etapas de otras instrucciones. En realidad las instrucciones se van ejecutando como en una cadena de producción o montaje, y en un instante dado estarán en ejecución tantas instrucciones como segmentos contenga la unidad de control. El conjunto de segmentos por los que tiene que pasar una instrucción para su ejecución completa se denomina **cauce**. A lo largo del tiempo, por término medio se podrá ejecutar una instrucción en un ciclo de reloj. Esta técnica se denomina **segmentación de cauce** (*pipelining*) o, abreviadamente, **segmentación**. Puede observarse que, en un funcionamiento normal, puede lograrse la ejecución de una instrucción por ciclo (CPI = 1).

Ciclo de reloj →		1	2	3	4	5	6	7	8	9	Tiempo →	
Instrucción ↓												
a)	I1, I2, I3, I4, ...	F1	D1	E1	W1	F2	D2	E2	W2	F3	.....	Procesador clásico CPI = 4
	I1, I5, I9, ...	F1	D1	E1	W1	F5	D1	E1	W1	F9	.....	
	I2, I6, I10, ...		F2	D1	E1	W1	F6	D1	E1	W1	.....	
	I3, I7, I11, ...			F3	D1	E1	W1	F7	D7	E7	.....	
b)	I4, I8, I12, ...				F4	D1	E1	W1	F8	D8	.....	Procesador segmentado CPI = 1
	Cauce 1; I1, I9 ...	F1	D1	E1	W1	.....					.....	
	Cauce 2; I2, I10, ...	F2	D2	E2	W2	.....					.....	
	Cauce 1; I3, I11, ...		F3	D3	E3	W3	.....				.....	
c)	Cauce 2; I4, I12, ...		F4	D4	E4	W4	.....				.....	Procesador superescalár o VLIW CPI = 0,5
	Cauce 1; I5, I13, ...			F5	D5	E5	W5	.....			.....	
	Cauce 2; I6, I14			F6	D6	E6	W6	.....			.....	

**Figura 7.10.** Esquema que muestra la descomposición de una instrucción en distintas fases y su ejecución en un procesador: a) clásico; b) segmentado, y c) superescalár o VLIW.

En el procesamiento con segmentación de cauce, debido a que se inicia la ejecución de una instrucción sin haberse completado las anteriores a ella, se presentan dos problemas:

1. Las **dependencias de datos**, consistentes en que cuando una instrucción B en su fase *F* necesite un resultado previo generado por una instrucción A anterior que no haya llegado a su fase *W* y por tanto dicho dato no esté aún disponible.
2. Las **dependencias de instrucciones**, que se producen en las bifurcaciones o saltos condicionales: si entra en el cauce una instrucción de salto condicional no se conocen las instrucciones que deben seguir entrando en el cauce sin haberse establecido el valor de la condición y sin haberse ejecutado la propia instrucción de salto.

Existen técnicas y métodos para prever y abordar los problemas anteriores, que, por supuesto, complican el diseño de la unidad de control.

Se puede lograr también paralelismo dentro del procesador con **arquitecturas superescalares**. Este tipo de computadores se caracteriza por tener varios cauces cuyas etapas pueden trabajar en un instante dado con distintas instrucciones de forma que se ejecutan en paralelo varias instrucciones sucesivas. En el ejemplo de la Figura 7.10c, sin tener en cuenta las dependencias, se podría lograr que por término medio el número de ciclos por instrucción fuese  $CPI = 0,5$ . En definitiva, los procesadores superescalares aumentan la complejidad del procesador al incluir más unidades y al tener que incluir recursos para ir seleccionando del flujo del programa las instrucciones que se pueden ejecutar en paralelo (*descubrir el paralelismo*).

### EJEMPLO 7.2

El Pentium tiene dos cauces con cinco segmentos cada uno; mientras que en el Pentium II hay un único cauce pero múltiples segmentos funcionales (ALU) que son los que propiamente ejecutan la instrucción.

Otro enfoque es el de los **procesadores VLIW** (*Very Long Instruction Word*, palabra de instrucción muy larga) que, como los procesadores superescalares, son segmentados que emiten (concluyen) varias instrucciones en cada ciclo y disponen de varias unidades de ejecución donde se pueden ejecutar operaciones simultáneamente. La diferencia fundamental reside en que el compilador crea las instrucciones VLIW incluyendo en cada una de ellas varias instrucciones escalares (las que ejecutan los procesadores superescalares) que el procesador VLIW puede ejecutar en paralelo. Al pasar la responsabilidad de planificar las instrucciones que se pueden ejecutar en paralelo al compilador, los procesadores VLIW no tienen que incluir el hardware que en un procesador superescalar se encargaba de dicha planificación, con lo que se reduce el consumo de potencia y queda liberado espacio en el chip del procesador para incluir otras unidades funcionales pudiéndose incrementar aún más el paralelismo. Un ejemplo de este tipo de procesadores es el Itanium 2 (IA-64).

Como veremos en la Sección 9.3.1, un programa, durante su ejecución, se puede descomponer en unidades denominadas **hebras** ejecutables concurrentemente unas con otras. Así, por ejemplo, un procesador de textos, en el momento de ejecutarse se descompone en pequeños procesos o hebras: una que atiende al teclado, otra que actualiza la imagen en pantalla, otra que atiende al corrector ortográfico, etc. Las hebras se ejecutan independientemente unas de otras. Una mejora adicional en las prestaciones de los procesadores superescalares se obtiene con los **procesadores multihebra**, cuya característica fundamental reside en que cada cauce se encarga de ejecutar una hebra distinta, evitándose así la dependencia entre instrucciones.

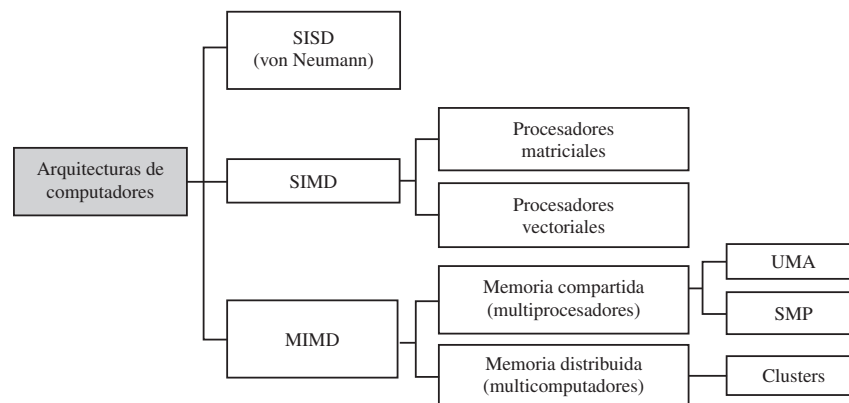
### EJEMPLO 7.3

El Pentium 4 dispone de hipersegmentación de 20 etapas. Las versiones HT son multihebra (Intel denomina a esta tecnología *Hyper-Threading*, HT). El procesador admite la ejecución simultánea de dos hebras que pueden corresponder a una misma tarea (programada en varias hebras) o a tareas diferentes, evitándose así las dependencias.

### Paralelismo a nivel de procesadores

Para obtener mejoras más substanciales en las prestaciones se han desarrollado diferentes tipos de computadores que contienen múltiples procesadores, clasificados por Flynn según se indica en la Figura 7.11, y que a continuación se describen:

- **Computadores de un único flujo de instrucciones y un único flujos de datos, SISD** (*Single Instruction Stream Single Data Stream*). Este tipo de sistemas se corresponde con los compu-



**Figura 7.11.** Clasificación de computadores atendiendo al paralelismo a nivel de procesadores.

tadores clásicos, tal y como se han descrito en la Sección 1.2.1 y se caracterizan por disponer de una única unidad de control (**monoprocesadores**), con segmentación de cauce o no. La unidad de control capta las instrucciones de la memoria, las decodifica y genera las señales de control que implementan la operación correspondiente a cada instrucción que debe realizar la unidad de tratamiento de datos. El flujo de datos se establece a partir de los operandos necesarios para realizar la operación codificada en cada instrucción, que se captan de los registros o de la memoria, y de los resultados generados por las instrucciones que se almacenan en los registros o en la memoria.

- **Computadores de un único flujo de instrucciones y múltiples flujos de datos, SIMD** (*Single Instruction Stream Multiple Data Stream*). Son computadores con una única unidad de control que procesa las instrucciones de una en una, pero cada una de ellas opera con múltiples datos, de forma que pueden realizar varias operaciones similares simultáneas con operandos distintos. Cada una de las secuencias de operandos y resultados utilizados por las distintas unidades del proceso define un flujo de datos diferente. Hay dos variantes (Figuras 7.9 y 7.11): los **procesadores matriciales**, que disponen de varias unidades de tratamiento de datos (ALU) que operan simultáneamente con la misma instrucción, y los **procesadores vectoriales**, que realizan operaciones con datos tanto escalares (valores individuales) como vectoriales (datos constituidos por varios elementos) y en los que se dispone de unidades funcionales segmentadas que operan con vectores de  $n$  elementos (del orden de 64 a 256, cada uno de 32 o 64 bits).
- **Computadores de múltiples flujos de instrucciones y múltiples flujos de datos, MIMD** (*Multiple Instruction Stream Multiple Data Stream*). Disponen de varias unidades de control que decodifican las instrucciones correspondientes a distintos programas. Cada uno de esos programas procesa conjuntos de datos diferentes, que definen distintos flujos de datos. Hay dos tipos de computadores MIMD (Figuras 7.9 y 7.11):
  - **Multiprocesadores**. Son sistemas con diversos procesadores pero que comparten la memoria. Los **multiprocesadores con acceso uniforme a memoria** o **UMA** (*Uniform Memory Access*) o **multiprocesadores simétricos (SMP)** se caracterizan porque cada procesador (y cada palabra) tiene el mismo tiempo de acceso a cualquier módulo de memoria, en contraste con los multiprocesadores **NUMA** (*NonUniform Memory Access*) en los que algunos accesos a memoria son más rápidos que otros, dependiendo del procesador que realice el acceso y la palabra a la que se acceda.
  - **Multicomputadores**. Son sistemas con múltiples procesadores, cada uno con su memoria local, y no compartiendo memoria. La comunicación entre procesadores se realiza a través de una red, enviando mensajes explícitos y esperando sus respuestas. Suelen dividirse en

dos categorías: **procesadores masivamente paralelos, MPP** (*Massively Parallel Processors*), que contienen más de cien procesadores interconectados por una red específica de muy alta velocidad (Cray T3E, IBM SP/2, Intel/Sandia, etc.), y **agrupaciones de estaciones de trabajo o de PC**, entre las que se encuentran los **clusters de computadores**, que están formados por PC o estaciones de trabajo convencionales interconectados por redes de comunicaciones.

#### EJEMPLO 7.4

Hay sistemas multiprocesador integrados en un único circuito integrado, denominándose **multiprocesadores en un chip (CPM)**. Un ejemplo es el Power 4 de IBM, que incluye dos procesadores superescalares a 1 GHz (o más), y cuya velocidad de transferencia con la caché es de 100 GB/s y la del bus de panel frontal 55 GB/s. Hay versiones que incluyen cuatro Power4 en un módulo multichip MCM con un ancho de banda de 40 GB/s a memoria o a otros módulos.

## 7.5. Conclusión

En el presente capítulo hemos presentado los conceptos que se siguen para interconectar las distintas unidades que configuran un computador. Para ello, en primer lugar (Sección 7.1) hemos mostrado las estructuras básicas de interconexión, a continuación (Sección 7.2) hemos descrito las características de los buses normalizados más comunes, y hemos estudiado (Sección 7.3) como ejemplo la estructura de un PC compatible. Por último (Sección 7.4) hemos introducido el concepto de paralelismo en computadores.

### Test



**T7.1.** Una estructura de computador con un solo bus:

- a) Es poco eficiente debido a que las distintas unidades tienen que compartir el mismo bus, y además los dispositivos más lentos ralentizan a los más rápidos.
- b) Es muy complejo diseñarla porque los buses deben ser muy largos y tener un elevado número de conexiones.
- c) Es muy eficiente, ya que todas las unidades de datos y direcciones son comunes para todos los dispositivos.
- d) Resulta muy económica, ya que se puede utilizar cualquier normalización de bus.

**T7.2.** Un controlador DMA es un subsistema para controlar la transferencia de información entre:

- a) Procesador y memoria caché (y viceversa).
- b) Memoria caché y memoria principal (y viceversa).
- c) Memoria principal y periféricos (y viceversa).
- d) Periféricos.

**T7.3.** Un controlador de E/S es un sistema que hace posible controlar directamente la transferencia de información entre:

- a) Procesador y memoria caché (y viceversa).
- b) Memoria caché y memoria principal (y viceversa).
- c) Memoria principal y periféricos (y viceversa).
- d) Periféricos del mismo rango de velocidades.

**T7.4.** Un bus ISA:

- a) Es el más rápido, ya que sirve para interconectar el procesador y la caché externa con la memoria principal.
- b) Es de velocidad media, e interconecta el bus más rápido con el controlador gráfico, controlador SCSI, y puente ISA.
- c) Es un estándar con el que se pueden interconectar periféricos en cadena, utilizando tan solo una conexión al bus interno del PC.
- d) Es el bus tradicional de los PC para conectar periféricos relativamente lentos.

**T7.5.** Un bus procesador-memoria:

- a) Es el más rápido, ya que sirve para interconectar el procesador y la caché externa con la memoria principal.

- b) Es de velocidad media, e interconecta el bus más rápido con el controlador gráfico, controlador SCSI, y puente ISA.
- c) Es un estándar con el que se pueden interconectar periféricos en cadena, utilizando tan solo una conexión al bus interno del PC.
- d) Es el bus tradicional de los PC para conectar periféricos relativamente lentos.

**T7.6.** Un bus local PCI:

- a) Es el más rápido, ya que sirve para interconectar el procesador y la caché externa con la memoria principal.
- b) Es de velocidad media, y puede interconectar el bus más rápido con el controlador gráfico, controlador SCSI, y puente ISA.
- c) Es un estándar con el que se pueden interconectar periféricos en cadena, utilizando tan solo una conexión al bus interno del PC.
- d) Es el bus tradicional de los PC para conectar periféricos relativamente lentos.

**T7.7.** Un bus SCSI:

- a) Es el más rápido, ya que sirve para interconectar el procesador y la caché externa con la memoria principal.
- b) Es de velocidad media, e interconecta el bus más rápido con el controlador gráfico, controlador SCSI, y puente ISA.
- c) Es un estándar con el que se pueden interconectar periféricos en cadena, utilizando tan solo una conexión al bus interno del PC.
- d) Es el bus tradicional de los PC para conectar periféricos relativamente lentos.

**T7.8.** La velocidad de transferencia en un bus procesador-caché es del orden de:

- a) GByte/s.
- b) Decenas de MBytes/s.
- c) MBytes/s.
- d) KBytes/s.

**T7.9.** La velocidad de transferencia en un bus local (PCI, por ejemplo) es del orden de:

- a) KByte/s.
- b) Centenas de MBytes/s y GBytes.
- c) Decenas de MBytes/s.
- d) MBytes/s.

**T7.10.** La velocidad de transferencia en los buses de conexión de discos duros (IDE, por ejemplo) y dispositivos DVD es del orden de:

- a) GByte/s.
- b) Centenas de MBytes/s.
- c) KBytes/s.
- d) MBytes/s y decenas de MBytes/s.

**T7.11.** La velocidad de transferencia en los buses serie actuales (USB 2.0 y FireWire) es del orden de:

- a) GByte/s.
- b) Centenas de MBytes/s.
- c) MBytes/s y decenas de MBytes/s.
- d) KBytes/s.

**T7.12.** ¿Cuál de los siguientes buses es de tipo paralelo?

- a) FireWire.
- b) USB.
- c) RS-232.
- d) PCI.

**T7.13.** ¿Cuál de los siguientes buses es de tipo serie?

- a) FireWire.
- b) SCSI.
- c) EISA.
- d) PCI.

**T7.14.** USB es:

- a) Una interfaz serie normalizada.
- b) Un controlador de vídeo.
- c) Un controlador para redes LAN.
- d) Una interfaz paralelo para impresoras.

**T7.15.** SCSI es:

- a) Una interfaz serie centralizada.
- b) Un controlador de vídeo centralizado.
- c) Un controlador para una estación central de una red LAN.
- d) Una interfaz para conexión paralela de periféricos.

**T7.16.** El término AGP, utilizado para designar un tipo de controlador, son las iniciales de:

- a) Adaptive Graphics Protocol.
- b) Advanced Graphics Port.
- c) Attachment General Peripheral.
- d) Architecture for General Purpose.

**T7.17.** Un procesador matricial es un sistema que:

- a) Dispone de varios procesadores de datos que operan simultáneamente bajo el control de una única instrucción.
- b) Contiene diversos procesadores que comparten una memoria común.
- c) Dispone de varios procesadores de datos que operan simultáneamente con datos escalares o vectoriales, bajo el control de una única instrucción.
- d) Contiene diversos procesadores, cada uno de ellos con su memoria local, y no compartiendo ninguna memoria.

**T7.18.** Un procesador vectorial es un sistema que:

- a) Contiene diversos procesadores, cada uno de ellos con su memoria local, y no compartiendo ninguna memoria.
- b) Dispone de varias unidades aritmético-lógicas que operan simultáneamente bajo el control de una única instrucción.
- c) Dispone de una o varias unidades aritmético-lógicas segmentadas que operan simultáneamente con datos escalares o vectoriales, bajo el control de una única instrucción.
- d) Contiene diversos procesadores que comparten una memoria común.

**T7.19.** Un multiprocesador es un sistema que:

- a) Dispone de varios procesadores de datos que operan simultáneamente bajo el control de una única instrucción.

- b)** Dispone de varios procesadores de datos que operan simultáneamente con datos escalares o vectoriales, bajo el control de una única instrucción.
- c)** Contiene diversos procesadores que comparten una memoria común.
- d)** Contiene diversos procesadores, cada uno de ellos con su memoria local, y no compartiendo ninguna memoria.
- T7.20.** Un multicomputador es un sistema que:
- a)** Dispone de varios procesadores de datos que operan simultáneamente bajo el control de una única instrucción.
- b)** Dispone de varios procesadores de datos que operan simultáneamente con datos escalares o vectoriales, bajo el control de una única instrucción.
- c)** Contiene diversos procesadores que comparten una memoria común.
- d)** Contiene diversos procesadores, cada uno de ellos con su memoria local, y no compartiendo memoria.

## Problemas resueltos



- P7.1.** Considere una versión de CODE-2 que funciona a una frecuencia de 1 GHz. Suponiendo que las transferencias en los buses no ralentizan el funcionamiento de CODE-2 (es decir, en un ciclo de reloj se hace una transferencia de información), hacer una estimación de la velocidad mínima de transferencia (MB/s) en los buses externos de dirección y de datos.

### SOLUCIÓN

Si la frecuencia de reloj es 1 GHz quiere decir que se generan  $10^9$  ciclos por segundo. Como el sub-bus de datos es de 16 bits = 2 Bytes, la velocidad de transferencia será:

$$AB = 2 \frac{\text{Bytes}}{\text{ciclo}} \cdot 10^9 \frac{\text{ciclos}}{\text{s}} = 1,86 \text{ GB/s}$$

- P7.2.** Suponiendo que en CODE-2 el puerto de salida para datos a visualizar en pantalla es el OPF1, hacer una estimación del tiempo que se tardaría en visualizar una imagen que se encuentra en la memoria principal a partir de la dirección A000 y que ocupase 20 KB. (*Sugerencia:* Utilizar los resultados del problema anterior, y realizar un programa para realizar la transferencia y calcular el tiempo que tardan en ejecutarse las instrucciones correspondientes.)

### SOLUCIÓN

Realmente, para transferir 20 KB tendremos que transferir 10 Kpalabras, ya que en cada posición de CODE-2 se memorizan 2 Bytes. En primer lugar pasaremos 10 K a hexadecimal:

$$10 \text{ K} = D'10 \cdot 2^{10} = B'10 \ 10 \ 00 \ 0000 \ 0000 = H'2800$$

El programa sería el siguiente:

	Explicación	Instrucción	N.º de ciclos	
	$r0 \leftarrow 0000$	LLI $r0,00$	6	Número de datos a transferir.
	$r1 \leftarrow 0001$	LLI $r1,01$	6	
		LLI $r2,00$	6	
	$r2 \leftarrow 2800$	LHI $r2,28$	8	Inicialización dirección de memoria.
		LLI $r3,00$	6	
	$r3 \leftarrow A000$	LHI $r3,A0$	8	
<i>a</i>	$rD \leftarrow r3$	ADDS $rD,r3,r0$	$7 \times 10.240$	Llevar dato a $r4$ . Escribir dato en puerto OPF1. Actualizar número datos que faltan. Dirección de fin. Saltar a fin si se ha acabado. Actualizar dirección de memoria. Dirección de salto. Salto a <i>a</i> .
	$r4 \leftarrow M(r3)$	LD $r4,00$	$9 \times 10.240$	
	$OPF1 \leftarrow r4$	OUT $F1,r4$	$8 \times 10.240$	
	$r2 \leftarrow r2 - 1$	SUBS $r2,r2,r1$	$7 \times 10.240$	
		LLI $rD,f$	$6 \times 10.240$	
		BZ	$6 \times 10.240$	
	$r3 \leftarrow r3 + 1$	ADDS $r3,r3,r1$	$7 \times 10.240$	
		LLI $rD,a$	$6 \times 10.240$	
		BR	$6 \times 10.240$	
<i>f</i>		HALT	6	Final.
Total			634.926	



El número total de ciclos máquina es 634.926, con lo que el tiempo total será:

$$t = NC \cdot T = 634.926 \text{ ciclos} \cdot 10^{-9} \text{ s/ciclo} = 635 \text{ ms}$$

**P7.3.** Suponiendo que se instalase un controlador DMA en CODE-2 con los siguientes registros:

- AR, dirección del próximo dato, asignado al puerto F2.
- WR, número de palabras que quedan por transferir, asignado al puerto F3.
- CR registro de control/estado, asignado al puerto de entrada F4 y al de salida F4.

Hacer una estimación del tiempo necesario para hacer la transferencia de un bloque de 20 KB ubicado a partir de la posición A000 de memoria al puerto de salida F1 indicada en el apartado anterior, suponiendo que cuando el DMAC tiene que realizar una transferencia se realiza una parada del procesador. Suponer que la palabra de control del CDMA para hacer la transferencia entre memoria y al periférico F1 es OPF1. (*Sugerencia:* Utilizar los resultados del problema anterior y realizar un programa para realizar la transferencia y calcular el tiempo que tardan en ejecutarse las instrucciones correspondientes.)

#### SOLUCIÓN

Realmente, para transferir 20 KB tendremos que transferir 10 Kpalabras, ya que en cada posición de CODE-2 se memorizan 2 Bytes. En primer lugar pasaremos 10 K a hexadecimal:

$$10 \text{ K} = D'10 \cdot 2^{10} = B'10 \ 10 \ 00 \ 0000 \ 0000 = H'2800$$

Sencillamente el programa tendrá que inicializar los registros del DMAC y dar a éste la orden de transferir. Un programa podría ser el siguiente:

Explicación	Instrucción	N.º de ciclos
$r2 \leftarrow 2800$	LLI $r2,00$	6
	LHI $r2,28$	8
$r3 \leftarrow A000$	LLI $r3,00$	6
	LHI $r3,A0$	8
$r4 \leftarrow 00F1$	LLI $r4,F1$	6
Inicializar puerto OPF2	OUT $F2,r3$	8
Inicializar puerto OPF3	OUT $F3,r2$	8
Inicializar puerto OPF4	OUT $F4,r4$	8
<b>Total</b>		<b>58</b>

Suponemos que el DMA consume dos ciclos en transferir una palabra desde memoria al periférico: uno en proporcionar el DMAC la dirección de memoria y el otro en escribir el dato (2 Bytes), depositarlo en el bus por la memoria y cargarlo en el puerto OPF1.

Además de los ciclos anteriores (inicialización del DMAC) hay que contabilizar los  $2 \times 10 \times 1.024 = 20.480$  ciclos que utiliza el DMA en transferir los 20 KB, donde hemos tenido en cuenta que cada palabra de CODE es de 2 Bytes.

En total, el número de ciclos consumidos serán:

$$NC = 20.480 + 58 = 20.538 \text{ ciclos}$$

Con lo que el tiempo total será:

$$t = NC \cdot T = 20.538 \text{ ciclos} \cdot 10^{-9} \text{ s/ciclo} = 20,5 \mu\text{s}$$

Según el resultado del problema anterior, el tiempo es  $\frac{635 \cdot 10^{-3}}{20 \cdot 10^{-6}} = 31.750$  veces menor que si no se utilizase DMAC.

**P7.4.** Se encuentra grabado con calidad de CD y con un factor de compresión de 11:1 un archivo de estéreo de audio que ocupa 3,79 MB y corresponde a una canción que dura 4:08 minutos. Obtener la velocidad de transferencia que debería tener como mínimo el bus que actúa sobre los dos conversores D/A de la salida de la tarjeta de audio. *Nota:* Recordar (Tabla 2.3) que en calidad CD las muestras de cada canal deben ser de 16 bits.

## SOLUCIÓN

Dividiendo el tiempo que dura la canción por el número total de muestras que deben presentarse ante cada canal de salida obtendremos la frecuencia de salida, y a partir de ésta la velocidad de transferencia.

La capacidad del archivo descomprimido será:

$$C_{des} = 3,79 \text{ MB} \cdot 11 = 41,69 \text{ MB}$$

El número total de muestras ( $N_s$ ) por canal será:

$$N_s = \frac{41,69 \text{ MB}}{2 \frac{B}{\text{muestra}} \cdot 2 \text{ canales}} = 10.928.784 \frac{\text{muestras}}{\text{canal}}$$

Ahora bien, esas muestras deben escucharse en 4:08 minutos = 248 segundos; luego la frecuencia de conversión digital/analógica (D/A) debe ser:

$$F = \frac{10.928.784 \text{ muestras}}{248 \text{ s}} = 44,1 \text{ KHz}$$

Por otra parte, en cada período de conversión hay que presentar en la entrada del conversor D/A 4 Bytes (dos por cada canal), con lo que la velocidad de transferencia debe ser:

$$AB = 44,1 \cdot 10^3 \frac{\text{ciclos}}{\text{s}} \cdot 4 \frac{B}{\text{ciclo}} = 172 \frac{KB}{s}$$

**P7.5.** A través de un módem se van a enviar mensajes a un computador remoto. La transmisión se realizará utilizando 7 bits de datos, 1 bit de inicio, 1 bit de parada y 1 bit de paridad impar por cada carácter. Cada mensaje puede tener un máximo de 250 caracteres.

- Calcular la velocidad en bits/s necesaria para transmitir un máximo de 1.000 mensajes por minuto.
- Debido a errores en la transmisión hay que fijar la velocidad en 14.400 bits/s. ¿Qué cantidad de mensajes podremos transmitir entonces en 1 minuto?

## SOLUCIÓN:

- En primer lugar calcularemos el número de bits que tenemos que transferir por cada mensaje:

$$T_{mensaje} = 250 \text{ caracteres} \cdot (7 + 1 + 1 + 1) \frac{b}{\text{carácter}} \cdot 2.500 \frac{b}{\text{mensaje}}$$

Para poder transferir 1.000 mensajes en un minuto necesitaremos la siguiente velocidad de transferencia:

$$v = \frac{1.000 \text{ mensajes} \cdot 2.500 \frac{b}{\text{mensaje}}}{60 \text{ s}} = 41.666,66 \frac{b}{s}$$

- El número de mensajes que podemos transferir a 14.400 b/s durante 1 minuto es:

$$N = \frac{14.400 \frac{b}{s} \cdot 60 \text{ s}}{2.500 \frac{b}{\text{mensaje}}} = 345,6 \text{ mensajes}$$

**P7.6.** La memoria del ejemplo de PC descrito en la Sección 7.3 permite capturar datos de 128 bits a una frecuencia de 400 MHz. ¿Cuál es su velocidad de transferencia?

## SOLUCIÓN

Como en cada ciclo de reloj se captan:

$$128 \text{ bits} = \frac{128}{8} = 16 \text{ B}$$



la velocidad de transferencia será:

$$AB = 400 \cdot 10^6 \frac{\text{ciclos}}{s} \cdot 16 \frac{B}{\text{ciclo}} = 5,96 \text{ GB/s}$$

- P7.7.** Un acelerados de gráficos AGP 4x, de 32 bits y similar al descrito en el ejemplo de PC de la Sección 7.3, funciona a una frecuencia de 66 MHz y su velocidad de transferencia es de 1.007 MB/s. ¿Cuántas transferencias se realizan por ciclo?

SOLUCIÓN

En un ciclo se transmiten el siguiente número de bytes:

$$N = \frac{1.007 \cdot 2^{20} B/s}{66 \cdot 10^6 \text{ ciclos/s}} = 16 B$$

Ahora bien, como el bus es de 32 bits = 4 Bytes, el número de transferencias que se realizan por ciclo serán  $16/4 = 4$  transferencias por ciclo.



## Problemas propuestos

**P7.8.** Cuánto tiempo se tardará en transferir una imagen de 4 MB a través de los siguientes sistemas:

- Un puerto AGP 1x.
- Un bus PCI 2.0 de 32 bits.
- Un bus Ultra ATA-133.
- Una línea de comunicaciones ADSL que transmite a 1 Mbps.

**P7.9.** Sabiendo que en la especificación del bus estándar ISA se determina que las transferencias son de 16 bits de datos, que se necesitan 3 ciclos para realizar cada una de ellas y la frecuencia de reloj es de 8 MHz, ¿qué velocidad de transferencia se obtiene?

**P7.10.** Suponiendo que se actualiza la información de una pantalla a una frecuencia de 30 imágenes por segundo a través de un puerto AGP 2x (528 MB/s) que actúa sobre los tres conversores D/A (uno para cada color) que controlan la pantalla, indicar la máxima resolución posible de la pantalla en los dos siguientes casos:

- Si la información transferida por el puerto corresponde a 8 bits por cada color.

- Si la información transferida por el puerto corresponde a un color de una paleta de 256 mezclas.

**P7.11.** Realizar el problema anterior suponiendo que la información de pantalla se transfiriese a través de un bus PCI de 132 MB/s.

**P7.12.** Audio Codec'97 (AC'97) es una especificación de Intel para sistemas de audio de alta calidad para equipos PC de sobremesa que es capaz de reproducir señales de audio en dos modalidades: 96 KHz/20 bits en estéreo y 48 KHz, 20 bits/muestra, en modo multicanal (4 o 6 canales). Un circuito de este tipo se incluye en la Figura 7.7. Indicar la velocidad de transferencia de chipset ICH5 para satisfacer la especificación mencionada. *Nota:* Más información puede obtenerse en

[http://www.intel.com/design/chipsets/audio/ac97\\_r23.pdf](http://www.intel.com/design/chipsets/audio/ac97_r23.pdf)

**P7.13.** Obtener el tiempo medio de ejecución de una instrucción en un procesador superescalar de 6 cauces, suponiendo que no existiesen dependencias de datos ni de instrucciones.

# Redes de computadores e Internet

La primera parte de este capítulo (Sección 8.1) pretende dar una visión general sobre las redes de computadores; para ello, en primer lugar se presentan distintas topologías de interconexión (Sección 8.1.1), para, seguidamente, describir los conceptos básicos del modelo de referencia OSI que establece un marco sobre la forma en la que sistemas diferentes cualesquiera pueden comunicarse entre sí (Sección 8.1.2). Las dos secciones siguientes se dedican a los dos tipos de redes más relevantes: las de área local o LAN (Sección 8.1.3) y las de área amplia o WAN (Sección 8.1.4). La información en una red es controlada y dirigida por medio de distintos dispositivos o procesadores de red, que se describen en la Sección 8.1.5.

La Sección 8.2 se dedica íntegramente a Internet. Se inicia explicando cómo se realiza el direccionamiento en Internet (Sección 8.2.1), seguidamente se describe el conjunto de protocolos TCP/IP (Sección 8.2.2), que es el que ha permitido el desarrollo de Internet. Posteriormente se dedican dos secciones a aplicaciones: una a aplicaciones básicas (Sección 8.2.3) y otra a la World Wide Web (www), conocida abreviadamente por web (Sección 8.2.4).

## 8.1. Redes de computadores

Una **red de computadores** está formada por una serie de computadores autónomos interconectados a través de algún medio de comunicación (hilos conductores, cable coaxial, enlace de microondas, enlace vía satélite, fibra óptica, etc.) para intercambiar información y compartir recursos.

Todo computador conectado a una red debe de disponer de:

- *Hardware*, constituido por una tarjeta de red<sup>1</sup>, adaptador de red o **NIC** (*Network Interface Card*) que interconecta físicamente un equipo a la red.
- *Software* específico para realizar la interconexión.

---

<sup>1</sup> Aunque a los circuitos para conectar un equipo a la red se suelen denominar *tarjeta* de red o NIC, no siempre se encuentran constituyendo una tarjeta de circuito impreso; en efecto, en la actualidad este hardware tiende a integrarse en el propio chip del microprocesador, o en los chipset asociados.

Hoy en día prácticamente han desaparecido los sistemas centralizados, dando paso a los sistemas distribuidos interconectados por red, ya que éstos aportan una gran cantidad de beneficios, entre los que se encuentran:

- Permitir que muchos usuarios compartan información y recursos.
- Posibilitar el uso remoto de aplicaciones.
- Poder realizar tareas en paralelo en distintos computadores de la red.
- Permitir la gestión de bases de datos distribuidas.
- Aumentar la seguridad gracias a la redundancia.
- Posibilitar el diseño de sistemas de control industrial con control remoto.
- Suponer otro medio de comunicación entre personas (correo electrónico, radio y televisión a través de Internet, *blogs*, etc.).

El desarrollo de las redes de comunicaciones ha sido fuertemente impulsado gracias a la labor de organizaciones como la Confederación Internacional de Normalizaciones (**ISO**) y el Comité Consultivo Internacional sobre Telegrafía y Telefonía (**CITT**), que han establecido reglas y normalizaciones (estándares) sobre interconexión de sistemas abiertos.

### 8.1.1. TOPOLOGÍA DE REDES

Los computadores de una red se interconectan a través de **redes de comunicaciones**, que están formadas por líneas de interconexión y nodos (dispositivos o procesadores de red) (Figura 8.1). Los equi-

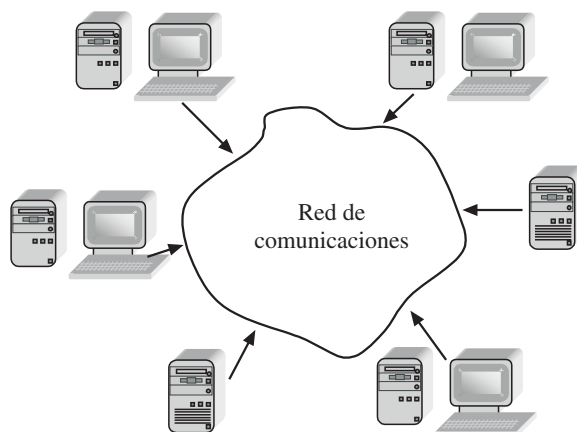


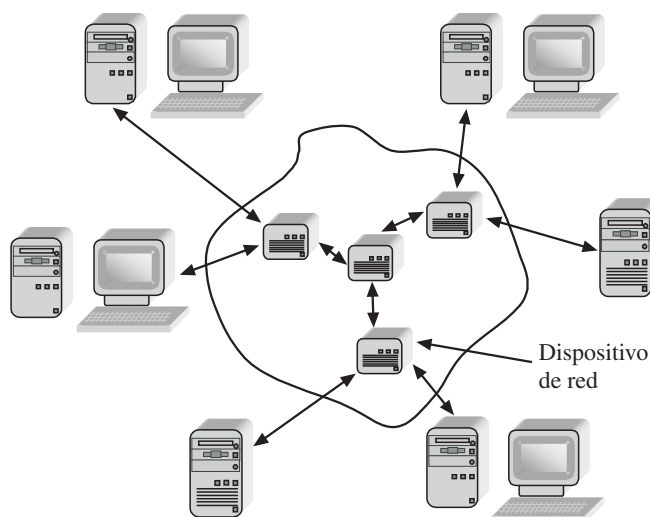
Figura 8.1. Ejemplo de red de comunicaciones.

pos de los usuarios se suelen denominar **computadores huésped**. Las **líneas de interconexión** constituyen el canal que conecta los distintos nodos de la red, y los **dispositivos de red** son procesadores que se dedican a seleccionar el camino de salida de la información y a tareas relacionadas con el transporte de la información (Figura 8.2). La Sección 8.1.5 se dedica a estudiar distintos dispositivos de red.

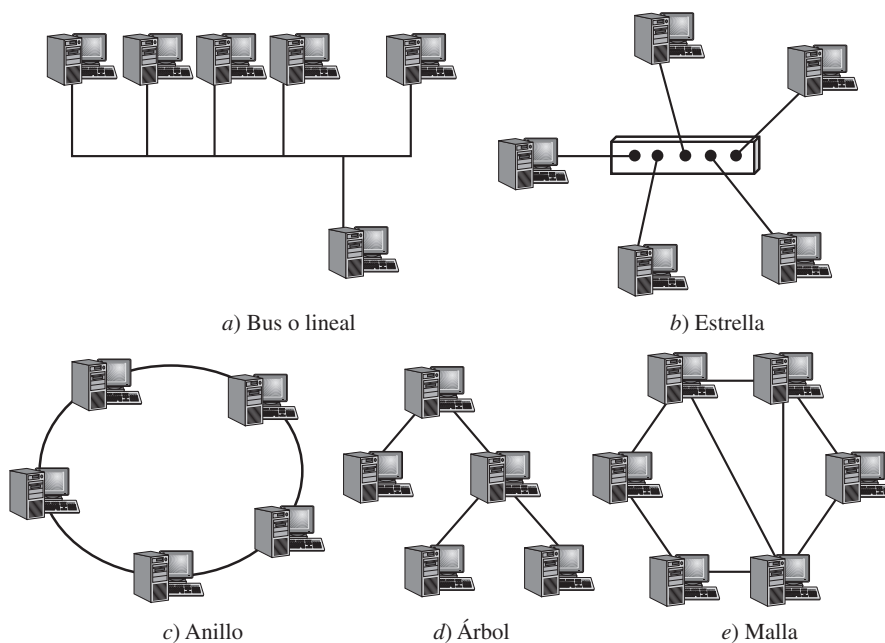
Las redes pueden ser **punto a punto**, si la transmisión se realiza sólo entre dos nodos, o **multi-punto**, cuando hay varios sistemas que comparten la línea.

Las topologías más usuales son (Figura 8.3):

- **Bus o lineal:** todos los computadores se conectan a un mismo cable, línea o canal.
- **Estrella:** todos los computadores huésped se conectan a un concentrador (*hub*) que gestiona la comunicación.



**Figura 8.2.** Ejemplo de red de comunicaciones, mostrando tres nodos de la red (dispositivos de red).



**Figura 8.3.** Posibles topologías de una red de computadores.

- **Anillo:** todos los computadores huéspedes se conectan de forma circular, de manera de que cada uno de ellos está conectado con uno anterior y otro siguiente.
- **Árbol:** los equipos se conectan en forma de árbol.
- **Malla o irregular:** los computadores huéspedes se van conectando formando una malla. Un caso particular es la malla de barras cruzadas, en la que los nodos forman una cuadrícula y hay un camino directo entre cualquier par de computadores huéspedes.

Las redes punto a punto suelen utilizar las topologías en estrella, árbol y anillo. Las redes multi-punto suelen ser en bus, anillo o estrella.

Teniendo en cuenta la extensión geográfica de una red, se consideran tres tipos de redes:

- **LAN** (o **redes de área local**). Son redes en las que los computadores que la forman están relativamente próximos unos de otros. Suelen utilizar una topología en bus, estrella o anillo.
- **MAN** (o **redes de área metropolitana**). Este tipo de redes son redes de alta velocidad que ofrecen servicios a nivel metropolitano. Son redes intermedias entre redes de área local y redes de área amplia. Las compañías telefónicas suelen ofrecer este tipo de servicios.
- **WAN** (o **redes de área amplia**). Son redes que se extienden a lo largo de amplias zonas geográficas. Utilizan topologías irregulares complejas, combinando los tipos estrella, árbol y malla.

En las Secciones 8.1.3 y 8.1.4 se describen con más detalle las redes de tipo LAN y WANN.

### 8.1.2. MODELO OSI

Para que una red de computadores funcione eficientemente es necesario establecer cómo van a ser las relaciones y funcionamiento de todos sus componentes. Para ello, la organización ISO diseñó el **modelo OSI** (*Open Systems Interconnection*). El modelo OSI es un modelo teórico que muestra cómo se pueden comunicar entre sí dos sistemas diferentes cualesquiera.

El **modelo de referencia OSI** considera siete capas o niveles conceptuales (Figura 8.4). En cada uno de estas capas se procesan unidades de información denominadas **PDU** (*Unidad de Datos de Protocolo*). Un bloque de información original que envía el usuario o programa de aplicación se suele denominar **mensaje**, constituyendo éste una PDU de la capa de aplicación. El mensaje va pasando hacia las capas inferiores hasta llegar a la capa física, que es la que realmente transmite la información en forma de señales (eléctricas, infrarrojas, microondas u ópticas). Al pasar de una capa a otra inferior, el mensaje se trocea o transforma en las PDU correspondientes a la capa, así obtenemos **segmentos**, **paquetes** y **tramas**, que son unidades de información de distintas capas donde se trocean las PDU de la capa superior y se les añade información adicional de control, tales como cabeceras y colas (AH, PH, SH, TH, NH, DH y DT, en la Figura 8.4) para identificar el destino y el origen (remi-

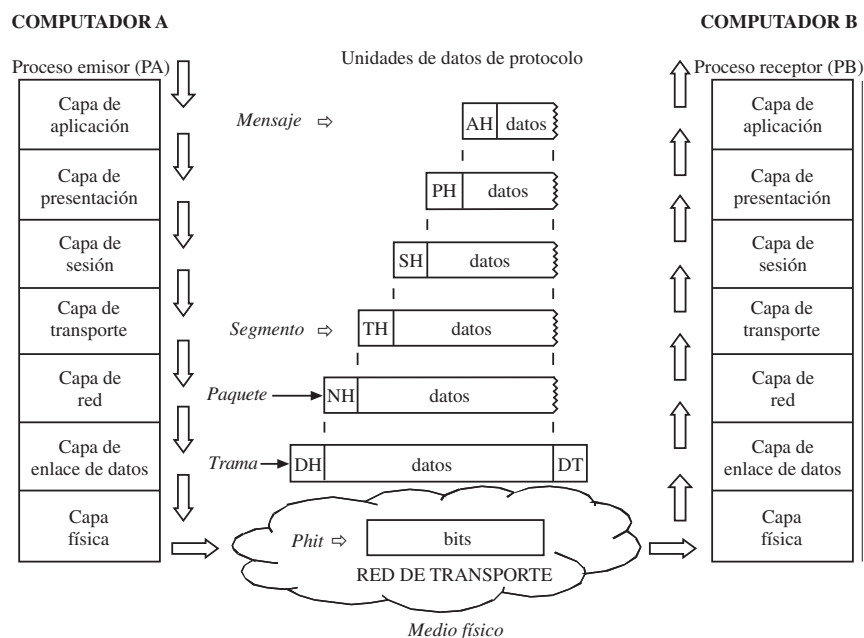


Figura 8.4. Esquema del modelo OSI.

tente), introducir redundancias de detección de errores, identificar la PDU en el receptor, etc. Cuando las señales llegan a través del medio físico a un nudo (dispositivo de red, Sección 8.1.5) o a un computador anfitrión se realiza el proceso inverso: la información va pasando de los niveles inferiores a los superiores, comprobándose si las PDU correspondientes a cada capa son correctas, eliminándose las cabeceras o colas de control, y recomponiendo el mensaje enviado; es decir, los datos de los paquetes se unen formando segmentos, y, a su vez, los segmentos se unen recomponiendo el mensaje original, que es entregado al proceso que se esté ejecutando en la capa de aplicación del computador receptor. Si en alguna capa se detecta alguna anomalía en la información que llega, se ejecutan las acciones adecuadas para informar de ella al computador origen o destino, o se da la orden de retransmitir de nuevo la PDU errónea. Muy esquemáticamente las tareas básicas que se realizan en cada capa son las que se describen a continuación.

### Nivel 7. Capa de aplicación

Esta capa es la que hace posible que el usuario (persona o programa de aplicación) pueda acceder a la red. En ella residen las aplicaciones que faciliten el trabajo del usuario. Dos computadores, A y B, se intercomunican a través de procesos, PA y PB, correspondientes a unas determinadas aplicaciones. El intercambio de información entre los dos procesos se efectúa por medio de algún protocolo de la capa de aplicación. Así por ejemplo, si se utiliza una aplicación de correo electrónico (Eudora o Outlook, por ejemplo) la máquina A necesitará en algún momento intercambiar información con otro computador B, para lo cual utilizará, por ejemplo, el protocolo SMTP de la capa de aplicación (Sección 8.2.2). Las aplicaciones de los usuarios que se intercomunican a través de una red con otras necesitan servicios (protocolos) para realizar tareas tales como acceso remoto, transferencia de archivos y correo electrónico. La unidad de datos (PDU), o conjunto de información a transmitir en un momento dado, de esta capa se denomina **mensaje** o **transacción**.

### Nivel 6. Capa de presentación

Se encarga de homogeneizar la sintaxis (formatos) y la semántica (cohesión y significado) de los intercambios de información, ya que los sistemas que se comunican pueden utilizar diferentes formas de codificar la información (ASCII, Unicode, etc.). También puede comprimir/descomprimir los datos y cifrar/descifrar los datos para que las transmisiones sean más seguras. Los programas correspondientes a esta capa suelen incluirse en el propio sistema operativo, o, alternativamente, constituir una biblioteca de programas específicos de la capa.

### Nivel 5. Capa de sesión

Es la interfaz entre el usuario y la red, gestionando el establecimiento de la conexión entre procesos remotos. Se encarga de identificar a los usuarios de procesos remotos así como de controlar y sincronizar el diálogo entre los sistemas que están comunicándose, procurando que la comunicación sea lo más fiable posible. Para sincronizar adecuadamente la transmisión de mensajes, éstos se dividen en **segmentos** o secciones, de forma que si en esta capa se detecta un fallo, en vez de tener que retransmitir todo el mensaje se retransmite sólo el segmento fallido. En los sistemas actuales las funciones de esta capa se suelen incluir en la capa de aplicación.

### Nivel 4. Capa de transporte

Esta capa es responsable de asegurar que el mensaje del origen se entregue completo al destino. Para ello fragmenta los segmentos que recibe de la capa de sesión en unidades, denominadas **paquetes**, que los entrega a la capa de red. Los paquetes pueden llegar desordenados al receptor, siendo la capa de transporte de éste la encargada de ordenarlos (o de reclamarlos si se pierden o llegan deteriorados) y

de componer segmentos que pasarán a la capa de sesión. También se encarga de optimizar el transporte.

### Nivel 3. Capa de red

Las PDU de esta capa son los paquetes, recibidos de la capa de transporte. La capa de red incluye las direcciones lógicas (direcciones IP, en caso de Internet, Sección 8.2.1) del origen y destino entre los que se realiza la transmisión. También se encarga de realizar el encaminamiento de los paquetes a través de los distintos dispositivos de la red, para lo cual se utilizan algoritmos eficientes que seleccionan la ruta más adecuada en cada momento, y de acuerdo con ella los paquetes se reexpiden en cada uno de los nudos que deba atravesar. También esta capa se encarga de prever la producción de bloques así como la congestión en los nudos que pudiesen producirse en hora punta por la llegada de oleadas de paquetes.

### Nivel 2. Capa de enlace de datos

Esta capa descompone cada PDU que recibe del nivel superior, en **tramas** o **bloques** de información, en los que añade una cabecera (DH) e información redundante para control de errores (DT, Figura 8.4). La cabecera contiene la dirección de dos estaciones (nudos de red o computadores huéspedes) contiguas, y al llegar a una nueva estación se cambian estas direcciones. La capa de enlace de datos, en definitiva, es la responsable de la transmisión entre dos nodos consecutivos.

### Nivel 1. Capa física

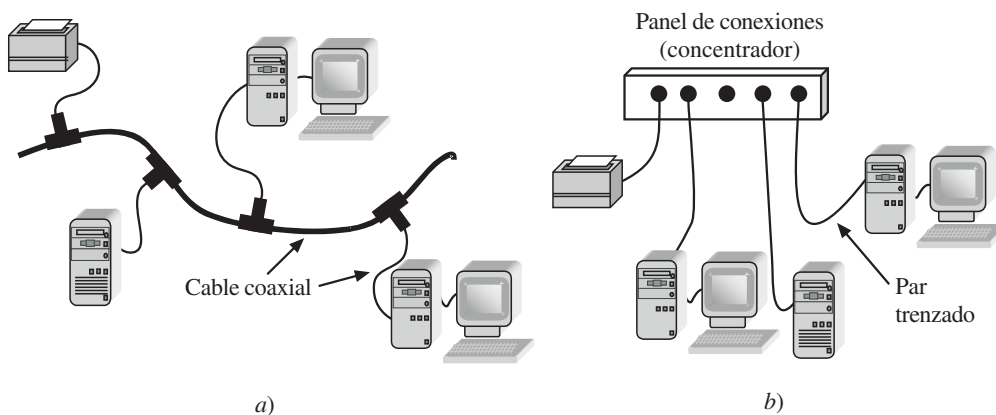
En esta capa se transmiten las cadenas de bits a través del medio físico, de acuerdo con los parámetros mecánicos y físicos de los dispositivos. En esta capa es donde se especifican los parámetros mecánicos (grosor de los cables, tipo de conectores, etc.), eléctricos (temporización de las señales, niveles de tensión, por ejemplo), etc., de las conexiones físicas. La unidad de información que se considera es el **phit** (unidad física, *PHysical unit* o *digIT*), que es sencillamente una cadena de bits que se transfiere en un ciclo de red por un por un enlace (par trenzado de cobre, cable coaxial, radio, infrarrojos o fibra óptica). En definitiva esta capa es responsable de que si en el emisor se envía un 1, al receptor le llegue un 1.

Conviene señalar que el modelo OSI es un modelo conceptual que no define ni especifica interfaces ni protocolos, únicamente establece criterios generales (un “marco”) sobre cómo concebir las redes de comunicaciones de datos y cómo estructurar y organizar las distintas funciones a realizar.

## 8.1.3. REDES DE ÁREA LOCAL (LAN)

En las redes de área local los computadores que la integran están relativamente próximos unos de otros, como en una habitación, un edificio o un campus. Una red local puede estar conectada a otras redes locales o de otro tipo. Estas redes se diseñan para poder compartir recursos entre todos los computadores conectados a la red. Entre las tecnologías más utilizadas para configurar redes LAN se encuentran:

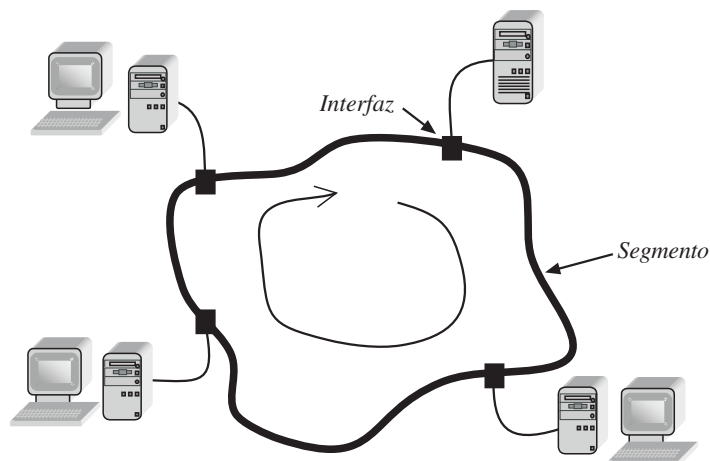
- **Ethernet.** Inicialmente este tipo de redes se implementaba con una topología en bus con cable coaxial (Figura 8.5a), pero en la actualidad se utilizan más topologías en estrella (Figura 8.5b) con par trenzado o fibra óptica. En el caso de usar esta última topología, el centro de la estrella consiste en un concentrador (*hub*) que reexpide la información que recibe a todos sus ramales. Con esta tecnología se consiguen velocidades de transmisión de hasta 10 Mbps. También existe la **Fast Ethernet** que utiliza tarjetas de interfaz de red que permiten alcanzar velocidades de hasta 100 Mbps. La **Gigabit Ethernet** se basa en la Ethernet original pero puede alcanzar velocidades de 10 Gbps. Una red Ethernet se caracteriza porque cuando un equipo



**Figura 8.5.** Red Ethernet: a) con topología de bus (cable coaxial) y b) con topología en estrella (pares trenzados).

desea emitir, primero escucha y si la línea (el canal) está libre emite su mensaje. Todos los computadores de la red captan el mensaje, analizan la dirección de destino y hacen caso omiso del mensaje, salvo si va destinado a él (aquel cuya dirección física coincide con la del mensaje). Este último computador capta (lee) el mensaje y seguidamente lo destruye. Cuando dos líneas tratan de transmitir a la vez se produce una **colisión**. La red Ethernet en este caso destruye los paquetes en conflicto y se reintentará la transmisión.

- **Anillo con paso de testigo (Token ring, Figura 8.6).** Esta tecnología fue creada por IBM y utiliza la topología en anillo. Inicialmente alcanzaba una velocidad entre 4 y 16 Mbps, aunque hoy día alcanza los 100 Mbps. En principio sólo circula un mensaje por el anillo, que está formado por segmentos conectados por los computadores de una red. Cuando no hay datos que transmitir circula un **testigo** con un código que indica que la red está *libre*. Cuando un computador quiere enviar un mensaje espera a que le llegue el testigo, lo cambia indicando *transmisión* y a continuación transmite su mensaje (paquete) al siguiente computador del anillo. El mensaje va llegando sucesivamente a los distintos computadores de la red, que lo abren, y si no son sus destinatarios lo regeneran y reenvían al siguiente computador. Al llegar al destino, el paquete es captado, se le incluye una marca de *reconocimiento*, y se le vuelve a reexpedir de forma que sigue circulando hasta completar el anillo; es decir, hasta llegar al emisor que detecta la marca de reconocimiento y lo elimina.



**Figura 8.6.** Esquema simplificado de una red en anillo.



### 8.1.4. REDES DE ÁREA AMPLIA (WAN)

Las redes de área amplia, como su nombre indica, son redes que se extienden a lo largo de extensas zonas geográficas (un país, un continente, etc.); y son redes públicas gestionadas por empresas de comunicaciones. Las redes de comunicación utilizadas por los computadores y para transmisión de datos pueden ser, entre otros, de los tipos que se indican a continuación.

#### 8.1.4.1. Línea telefónica convencional (analógica)

La línea telefónica convencional es una enorme red que pertenece a las compañías telefónicas. Los terminales telefónicos transforman la voz en señales eléctricas (analógicas) que se transmiten a través de la línea telefónica. Cualquier persona con un computador y una línea telefónica convencional puede conectarse con cualquier otra persona en cualquier parte del mundo, que esté conectada de la misma forma. Sin embargo, la línea telefónica está diseñada para transmitir señales de voz analógicas y, por tanto, no es adecuada para transmisiones digitales. De hecho, la transmisión de datos a través de estas líneas es muy lenta (56.000 bps frente a 10.000.000 de bps de una red Ethernet, por ejemplo). Para poder utilizar la línea telefónica para transmitir datos digitales hay que disponer de un hardware y un software específicos que permitan estas transmisiones.

Las señales que genera un computador son digitales, así que hay que convertirlas en analógicas para ser transmitidas, y en el receptor, al revés, las señales analógicas recibidas deben ser convertidas a digitales. El dispositivo que lleva a cabo estas conversiones se llama **módem**. Un módem es una tarjeta de circuito impreso o un circuito integrado que se conecta a un conector PCI (si es interno) o al puerto serie (si es externo) del computador y a la línea telefónica. Su *velocidad de transmisión* (la velocidad a la que puede enviar datos) se mide en bits por segundo (bps) y suele ser de 56.000 bps (56 Kbps). Los módems utilizan tecnologías de *compresión de datos* para transmitir los datos con menos bits (Sección 2.6). El software que utilizan contiene *protocolos para detectar o corregir los errores* introducidos por el ruido en las transmisiones.

#### 8.1.4.2. Líneas digitales

En los últimos años las compañías telefónicas se afanan en transformar sus líneas analógicas en líneas digitales, con objeto de ofrecer mejores prestaciones y nuevos servicios. Esto supone utilizar teléfonos digitales y adaptadores especiales en lugar de los módems tradicionales. Existen distintos tipos de líneas digitales:

- **RDSI (Red Digital de Servicios Integrados)**. Ofrece a los usuarios tres canales de comunicación: dos para voz o datos a 64 Kbps y 56 Kbps, que se pueden usar simultáneamente, y otro a 19 Kbps para voz. También se pueden combinar los canales para transmitir datos a 128 Kbps. Es posible contratar la **RDSI de banda ancha**, cuyos canales transmiten a 156 Mbps (salida)/622 Mbps (entrada) y 622 Mbps.
- **DSL (Digital Subscriber Line, línea de abonado digital)**. Las compañías telefónicas han desarrollado una nueva técnica que es más barata y permite aprovechar las instalaciones telefónicas tradicionales. Ofrecen varias alternativas:
  - **ADSL**: línea de abonado digital asimétrica.
  - **RADSL**: línea de abonado digital con velocidad adaptativa.
  - **HDSL**: línea de abonado digital de alta velocidad.
  - **IDSL**: línea de abonado digital RDSI.
  - **SDSL**: línea de abonado digital simétrica.
  - **VDSL**: línea de abonado digital de muy alta velocidad.

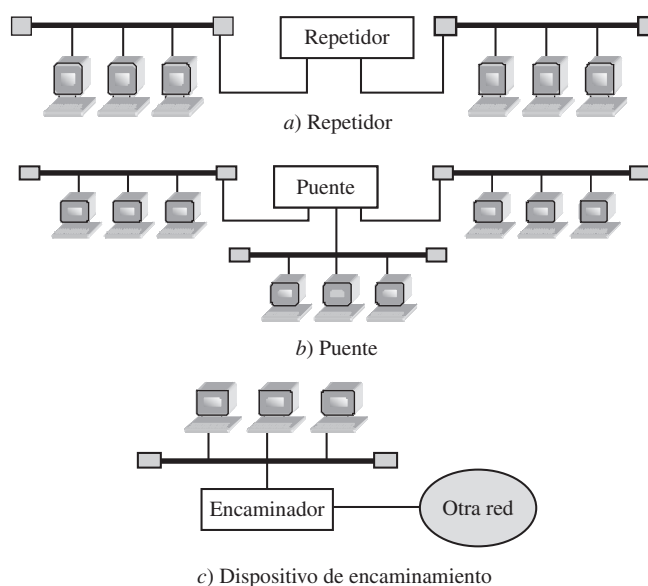
La más usual es la **ADSL**, que es una línea digital que permite conectarse a tiempo completo y precio asequible empleando la línea telefónica convencional sin impedir realizar comunicaciones telefónicas simultáneamente, mediante la instalación de un **separador** (*splitter*) que segmenta la señal y permite utilizar a la vez la línea de voz y datos. La técnica de transmisión se basa en que un usuario recibe más datos que envía, de forma que la velocidad de recepción ofertada es mucho mayor que la de envío de datos. En España se disponen velocidades desde los 256 Kbps a los 2 Mbps de recepción y de 128 Kbps a 300 Kbps de emisión.

- **ATM** (*Asynchronous Transfer Mode*, modo de transmisión asíncrono). ATM es un protocolo diseñado para enviar voz, vídeo y datos de forma muy eficiente a través de una red. Es un protocolo parecido a Ethernet o Token Ring, que consigue aumentar la velocidad de transmisión reduciendo los controles en las cabeceras de los paquetes, y utilizando unidades de transmisión más pequeñas y de tamaño fijo más fáciles de tratar por los dispositivos de la red. Así se consiguen velocidades entre 50 Mbps y 100 Gbps.
- **Cable módem**. El cable módem es una tecnología que permite conectarse a una red utilizando cable coaxial de televisión, consiguiéndose así velocidades más altas que con una línea convencional. Es una alternativa a RDSI o ADSL. Se pueden conseguir velocidades de hasta 27 Mbps, aunque en España se ofertan velocidades de 128 Kbps, 300 Kbps, 600 Kbps y 1 Mbps. Una ventaja de las transmisiones por cable coaxial es que por él se transmiten simultáneamente señales de teléfono, televisión y datos.

### 8.1.5. DISPOSITIVOS DE INTERCONEXIÓN EN REDES

Como indicamos en la Sección 8.1.1, los **dispositivos de red** son procesadores especializados que se dedican a seleccionar el camino de salida de la información que les llega y a tareas relacionadas con el transporte de la información a través de la red. Según la función que realicen reciben distintos nombres.

- **Repetidores** (*repeater*). Dispositivo electrónico que regenera las señales que recibe y las envía al resto de la red. Se encargan de conectar dos segmentos de una red, transfiriendo el tráfico de uno a otro (Figura 8.7a). Realiza únicamente funciones asociadas a la capa inferior del modelo OSI.



**Figura 8.7.** Procesadores de red: a) repetidor, b) puente y c) dispositivo de encaminamiento.

- **Puente (bridge).** Conecta dos segmentos de red, regenerando la señal y seleccionando el tráfico que pasa de uno a otro. De esta forma, si hay un tráfico de datos excesivo en un bus se puede dividir éste en dos, y en cada uno de los subbuses sólo habrá el tráfico de los computadores conectados a él y el que específicamente deje pasar el puente procedente del otro segmento (Figura 8.7b). Realiza operaciones asignadas a las dos capas inferiores del modelo OSI. Un **conmutador** es un puente optimizado con varias interfaces, de forma que puede controlar el tráfico entre diversos segmentos. Así, por ejemplo, una red con 80 equipos PC puede dividirse en 4 segmentos de 20 PC, controlando el conmutador el paso de paquetes de un segmento a otro (sólo pasarán cuando sea necesario).
- **Dispositivo de encaminamiento, encaminador o enrutador (router)** (Figura 8.7c). Es un dispositivo que no sólo selecciona el tráfico, sino que también direcciona la información de acuerdo con la mejor ruta, utilizando **algoritmos de encaminamiento**. Se utilizan para conectar redes independientes, por ejemplo dos de tipo LAN con otra WAN, formando **interredes**. Realiza funciones asignadas a las tres capas inferiores del modelo OSI.
- **Pasarela (gateway).** Es un dispositivo de encaminamiento que permite interconectar redes con distintos protocolos, en las distintas capas del modelo OSI. En definitiva, funciona como conversor de protocolos, realizando las funciones de las siete capas del modelo OSI. Físicamente una pasarela es un computador con el software adecuado para hacer la transformación de protocolos. Hoy en día se habla indistintamente de dispositivos de encaminamiento y pasarelas, ya que se incluyen en ellos las funciones de ambos.

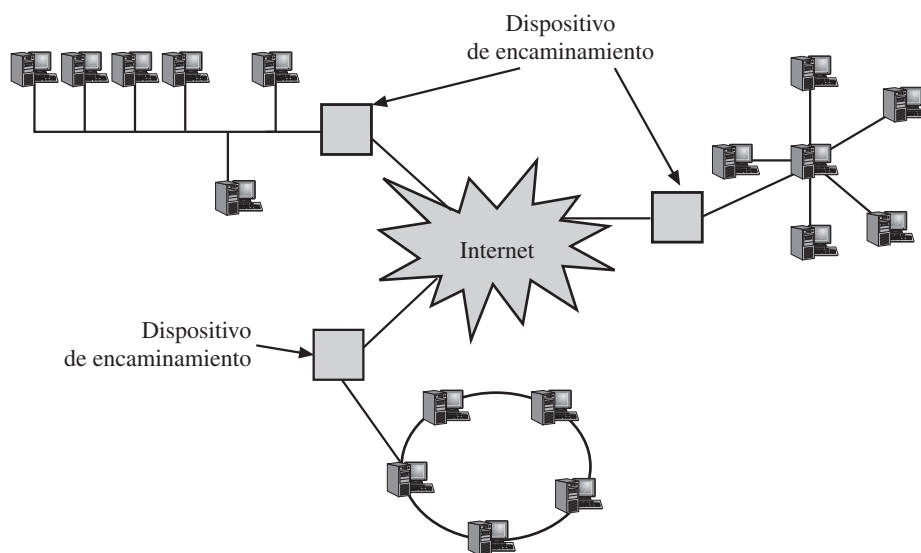


Figura 8.8. Topología de Internet.

## 8.2. Internet

Internet es la combinación de prácticamente la totalidad de redes del mundo, interconectadas por medio de dispositivos de encaminamiento. Ofrece la posibilidad de intercambiar texto, archivos y programas con cualquier otro usuario de cualquier parte del mundo que esté también conectado a Internet.

Bajo el punto de vista del usuario de Internet, las estaciones de la red son computadores (independientemente de su capacidad), cada uno de ellos identificado por una dirección, y la información se transmite de acuerdo con el conjunto de protocolos TCP/IP. El conjunto TCP/IP hace abstracción

de las redes individuales y considera todos los computadores de la red como si estuviesen conectados a una red única. Con precisión podríamos definir **Internet** como *la red formada por la interconexión de redes a lo ancho de todo el mundo que utiliza el protocolo TCP/IP*.

### 8.2.1. DIRECCIONAMIENTO EN INTERNET

Todos los computadores conectados a Internet están unívocamente identificados mediante una dirección, denominada **dirección IP**. Cada dirección IP es una secuencia o patrón de 32 bits. Para facilitar la transcripción de direcciones IP se suelen dar en notación decimal; para ello se forman cuatro grupos de 8 bits con los 32 bits originales y se representa cada grupo con el valor decimal correspondiente, separados por puntos. Como cada grupo es de 8 bits, el valor de las cifras decimales variará entre 0 y 255.

#### EJEMPLO 8.1

Obtener la dirección IP en decimal correspondiente a la dirección binaria:

1001 0110 0110 1101 0101 1011 0010 0101

Primero formamos cuatro grupos con los bits:

1001 0110 . 0110 1101 . 0101 1011 . 0010 0101

y ahora pasamos a decimal cada grupo:

1001 0110 = 150  
0110 1101 = 214  
0101 1011 = 191  
0010 0101 = 37

Luego la dirección IP en notación decimal es: **150.214.191.37**.

Las direcciones IP se dividen en dos partes, una que identifica el dominio en el que reside el computador (**identificador de red**) y la otra identifica el computador en concreto dentro del dominio (**dirección del computador o equipo anfitrión**).

#### EJEMPLO 8.2

La dirección IP 150.214.191.37 consta del identificador de red 150.214.191, que identifica un dominio de la Universidad de Granada, y 37 se refiere a un computador en concreto.

Para las personas es difícil de trabajar y memorizar estas cadenas de números (aunque sean decimales), así que a cada dirección IP se le asigna un nombre denominado **dirección del sistema de nombres de dominio (DNS)**. Así, a la Universidad de Granada se le ha asignado la DNS ugr.es (más fácil de recordar que su identificador de dominio). Las DNS también tienen dos partes, el nombre del servidor y un sufijo que indica el tipo de institución o procedencia. A esta segunda parte se le llama **dominio de alto nivel (top-level domain, TLD)**. Así, “.es” se refiere a “España”. En la Tabla 8.1 se muestran los TLD más usuales. En instituciones grandes también se suelen utilizar **subdominios**. Por ejemplo, un departamento dentro de la universidad puede tener un subdominio, de forma que su DNS podría ser, por ejemplo, atc.ugr.es (DNS del Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Granada).

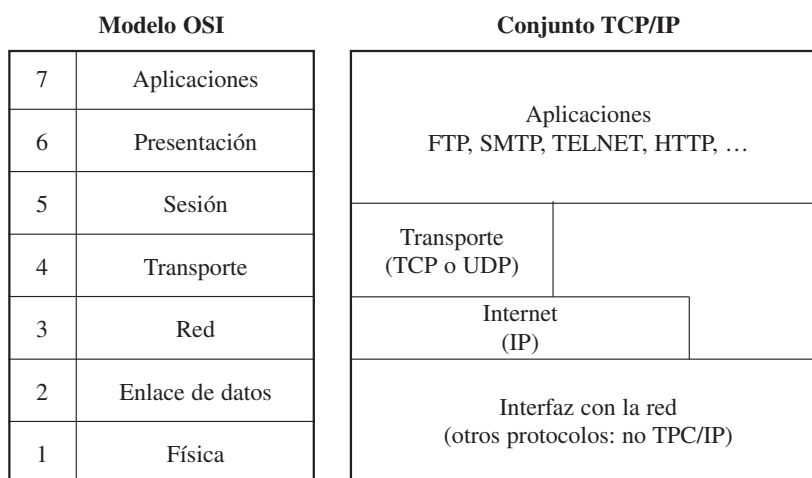
Dominio	Significado	Ejemplo
.es	España	<i>www.ugr.es</i>
.com	Uso comercial	<i>www.yahoo.com</i>
.org	Una organización	<i>www.isoc.org</i>
.net	Servicios de la red	<i>www.php.net</i>
.edu	Educación	<i>www.harvard.edu</i>
.info	Información	<i>www.microbes.info</i>

**Tabla 8.1.** Dominios más usuales.

### 8.2.2. PROTOCOLO TCP/IP

Como se ha comentado anteriormente, la peculiaridad que caracteriza a Internet es el uso del conjunto de **protocolos TCP/IP (Protocolo de control de transmisiones/Protocolo de Internet)**. Su desarrollo es anterior al modelo OSI, y es un modelo jerárquico en el que la capa de aplicación puede utilizar los servicios de cualquier capa inferior (Figura 8.9). Se consideran cuatro capas:

- **Capa de procesos y aplicaciones.** Esta capa se corresponde con las tres superiores del modelo OSI: aplicación, sesión y presentación; y en el conjunto TCP/IP las aplicaciones no siempre interactúan con la capa inmediatamente inferior (transporte o TCP), sino que también pueden relacionarse directamente con la capa de red (IP), e incluso con la capa de interfaz con la red (Figura 8.9). La interacción entre aplicaciones se realiza usualmente utilizando el **modelo**



**Figura 8.9.** Equivalencia entre el modelo OSI y el protocolo TCP/IP.

**cliente/servidor.** Según este modelo, un proceso de uno de los computadores (**cliente**) de la red solicita un servicio a otro proceso de otro computador (**servidor**), y el servidor suministra las respuestas a las peticiones del cliente. El protocolo HTTP, por ejemplo, especifica las reglas de cómo el cliente y el servidor interactúan para recuperar un documento: el cliente solicita la visualización de una determinada página web y el servidor se la proporciona o da los mensajes de error oportunos. Las aplicaciones y protocolos básicos que se incluyen en esta capa, y que se analizarán con más detalles en la Sección 8.2.3, son:

- **SMTP: protocolo sencillo para correo electrónico** (SMTP, *Simple mail transfer protocol*).

- **FTP: transferencia de archivos** (FTP, *File transfer protocol*).
- **TELNET**: conexión de terminal remoto o terminal virtual.
- **HTTP**: envío de mensajes usando TCP.
- **DNS y RTP**: envío de mensajes utilizando UDP.
- **Capa de transporte**. Se encarga de controlar las transmisiones. El conjunto de protocolos TCP/IP contiene dos protocolos alternativos para implementar el nivel de transporte: el **TCP** (**Protocolo de control de transmisión** o *Transmission Control Protocol*) y el **UDP** (**Protocolo de datagrama de usuario** o *User Datagram Protocol*).

El nivel de transporte recibe de la capa de aplicación **mensajes**, generando a partir de ellos **segmentos** o **datagramas de usuario**, dependiendo de si se utiliza TCP o UDP, respectivamente.

El protocolo TCP, antes de transmitir datos, envía un segmento de control a la capa de transporte de la estación destino solicitando que se disponga a recibir datos para una aplicación (proceso) determinada de la capa de aplicación y otras variables que son usadas por el protocolo; después espera a recibir un segmento de reconocimiento de la estación destino, para posteriormente enviar los segmentos de datos. Este tipo de funcionamiento, según el cual antes de enviar los datos se abre la comunicación, se denomina **orientado a conexión**. El software de esta capa descompone los mensajes en segmentos (de 1 a aproximadamente 1.500 caracteres de capacidad), que numera adecuadamente con objeto de que el protocolo TCP del receptor pueda recomponer el mensaje original. En cada segmento se incluye una cabecera con campos que identifican el proceso de la capa de aplicación que lo envía, el proceso de la capa de aplicación que lo debería recibir e información para detección de errores. Por otra parte el protocolo TCP de las dos estaciones se encarga de regular el flujo de información, asegurando que los segmentos lleguen sin error y, en secuencia, generando acuses de recibo y retransmitiendo los paquetes que eventualmente se pierdan. Por este motivo se dice que TCP es un **protocolo fiable**. El nivel TCP del receptor comprueba si los segmentos llegan intactos, los reensambla reconstruyendo así el mensaje original y suministra éste a la aplicación destino.

UDP es un **protocolo extremo-a-extremo**, mucho más sencillo que TCP, de tipo, sin conexión y no fiable. Antes de emitir los datos no se establece una conexión, sino que meramente se envían los datos (**datagramas de usuario**) a la dirección de destino, incluso aunque esta última no esté funcionando. UDP tampoco se encarga de la confirmación de la adecuada recepción de los datagramas de usuario. UDP se utiliza cuando la fiabilidad y seguridad son menos importantes que la velocidad.

- **Capa internet (IP)**. La capa internet gestiona la transferencia de información (segmentos o datagramas) a través de múltiples redes por medio del uso de encaminadores (o pasarelas). La capa IP de la computadora origen encapsula los datagramas de usuario o segmentos recibidos de la capa de transporte en **paquetes** (también denominados **datagramas IP**). Las capas IP de los dispositivos de encaminamiento actúan conjuntamente para encaminar los paquetes desde el origen al destino (de acuerdo con las direcciones IP). En efecto, a partir de la dirección destino y consultando unas tablas de rutas, se establece la línea de salida del encaminador por donde se debe reexpedir el paquete para acercarse a su destino.
- **Capa de red**. Es responsable de aceptar paquetes IP y transmitirlos a través de una red específica de comunicaciones. Existen diversas interfaces disponibles para redes de tipos tales como X.25, ATM, retransmisión de tramas, Ethernet y anillo con paso de testigo. Con esta capa se independizan las funciones de la capa internet y superiores de los detalles de la red subyacente. TCP/IP no define ningún protocolo específico para las capas de enlace de datos y física, admitiendo cualquier protocolo.

### 8.2.3. APLICACIONES BÁSICAS DE INTERNET

En la capa de aplicación podemos incluir las siguientes aplicaciones básicas: conexión de terminal remoto, transferencia de archivos y correo electrónico. A partir de estas aplicaciones se han desarrollado otras más complejas como la web. A continuación se describen brevemente.

### Conexión de Terminal remoto o TELNET (*TERminal NETwork*)

Este servicio permite, con el software apropiado (protocolo TELNET), conectarse de forma remota a cualquier computador. Esta conexión se hace emulando, en el equipo del usuario, un terminal virtual de forma que pueda utilizar el computador remoto como si se utilizase localmente.

### FTP (*File Transfer Protocol*)

El **protocolo de transferencia de archivos (FTP)** es la herramienta utilizada en Internet para transferir archivos entre computadores. Cualquier usuario puede copiar archivos de ámbito público con un programa cliente de FTP. Estos programas requieren el nombre del computador que se desea utilizar y un nombre de usuario y contraseña, que en el caso de acceder a archivos públicos es *anonymous*. Una vez identificado el usuario se tiene acceso a los archivos de la máquina deseada.

### Correo electrónico

Es uno de los servicios más utilizados en Internet. Permite enviar mensajes y archivos adjuntos de un computador a otro. Para que un usuario pueda usar este servicio debe de disponer de una cuenta y una dirección de correo electrónico. La **cuenta de correo electrónico** está formada por un nombre de usuario (que es parte de la dirección de correo electrónico) y una contraseña (para que sólo el propietario pueda acceder a los mensajes). El equipo del usuario debe conectarse a un **servidor de correo electrónico**, que gestiona el envío y recepción de correo. La **dirección de correo electrónico** identifica unívocamente a cada usuario y se forma uniendo, con @, el nombre de usuario con el nombre del servidor (usuario@servidor). Entre los protocolos más usados por los servidores de correo electrónico se encuentran el **SMTP (protocolo sencillo de transferencia de ficheros)** y el **POP (protocolo de oficina de correos, post office protocol)**. Para poder usar este servicio hay que instalar en el computador del usuario un programa cliente (Outlook, Eudora, etc.) que interactúe con el servidor de correo electrónico, aunque una alternativa que no exige este programa es acceder a través de la web a un servicio **web-mail**.

### Grupos de noticias o News

Un grupo de noticias es un sitio web en el que se discute sobre un tema en concreto. Para participar en ellos, los usuarios envían opiniones y documentos sobre el tema. Cada usuario puede leer y contestar a las opiniones de los demás, de forma que así se va formando el grupo. El programa de lectura de noticias consigue artículos e información de los servidores de noticias. Estos servidores utilizan el **protocolo de transferencia de noticias de red (NNTP, Network News Transfer Protocol)**. Los grupos de noticias se clasifican por categorías denominadas dominios, que a su vez se componen de temas individuales.

### Tertulias Internet o Chat o IRC (*Internet Relay Chat*)

Es una forma muy usual de que los usuarios de Internet se comuniquen en tiempo real. El *Chat* es un sistema multi-usuario en el que los usuarios entran en distintos canales para hablar de forma pública o privada sobre diversos temas. Los usuarios pueden leer, contestar, ignorar o crear mensajes.

### Servicios en línea

Hay empresas de servicios en línea que ofrecen correo electrónico, grupos de discusión, información en tiempo real, juegos en línea, comercio electrónico, etc. También suelen ofrecer acceso a Internet.

### Servicios paritarios (*peer-to-peer*)

Los servicios paritarios (o **P2P, peer-to-peer**) permiten que se conecten directamente dos computadores en concreto, realizando no sólo comunicaciones, sino también compartiendo sus archivos. Ejemplos de software P2P son las aplicaciones de mensajería instantánea. Estas aplicaciones permiten hacer chat privado así como transferir archivos de cualquier tipo.



### 8.2.4. WORLD WIDE WEB (WWW)

La red mundial *World Wide Web* (o abreviadamente **www**, o sencillamente **web**) es un conjunto de documentos (archivos) distribuidos a lo largo de todo el mundo y con enlaces entre ellos.

Los documentos se visualizan incluyendo algunas palabras, frases o imágenes resaltadas (en negrita o en un color destacado) debajo de las cuales se ocultan enlaces a otros documentos ubicados en el propio servidor o en otros. Seleccionando y activando con el ratón un elemento con enlace, automáticamente se hace una llamada al documento apuntado, visualizándose éste.

Un puntero a otro documento se denomina **hiperenlace**, y un texto con hiperenlaces se denomina **hipertexto**. Un hipertexto disponible en la web se denomina **página web**. Un conjunto de páginas web forma un **sitio web**, y las páginas web están alojadas en **servidores web**. La primera página que aparece al acceder a una dirección web se denomina **página principal** (*homepage*). Por lo general la página principal se organiza como un índice para acceder a otras páginas sobre el mismo tema en el propio servidor, y cualquier página puede tener hiperenlaces a cualquier servidor web. En la web no sólo se puede acceder a documentos de tipo hipertexto, también hay documentos de **hipermedia** que contienen gráficos, imágenes y sonidos. Se denomina genéricamente **hiperdocumento** a un hipertexto o a un hipermedia. El hecho de ir accediendo (“saltando”) de un documento a otro a través de la web se denomina **navegar a través de Internet**.

A continuación se describen brevemente diversos conceptos relacionados con el uso y funcionamiento de la web, como direccionamiento, el protocolo http, navegación y aplicaciones.

#### 8.2.4.1. Direccionamiento en la web (URL)

Obviamente uno de los conceptos fundamentales de la web es la forma de localizar los documentos. Esto se efectúa utilizando el **Localizador Uniforme de Recursos (URL)**, que puede referenciar cualquier tipo de documento en la red. Un localizador URL se compone de tres partes, separadas con rayas inclinadas:

*Protocolo-de-recuperación://computador/ruta-y-nombre-del-archivo*

Existen distintos protocolos para recuperación de un documento, entre los que se encuentran HTTP, FTP, Gopher, SMTP y TELNET. Computador es el nombre DNS del computador, y el último campo es el nombre completo del archivo que contiene la página buscada, incluyendo su ruta o camino de acceso dentro del computador donde se encuentra.

Con frecuencia el computador concreto donde se encuentra la página principal de un dominio se le denomina **www.nombre-del-dominio**.

#### EJEMPLO 8.3

Una dirección de una página web puede ser:

*http://www.ugr.es/Informatica*

lo que significa:

http	El protocolo de transferencia usado es el http.
www	La página forma parte de la World Wide Web (www es un alias del nombre del computador).
ugr.es	La página pertenece a la Universidad de Granada.
Informatica	Documento o camino hasta el documento solicitado.

Los hiperenlaces son sencillamente localizadores URL, ya que éstos determinan exactamente tanto el archivo y el computador donde se encuentra la página como el protocolo utilizado.



#### 8.2.4.2. Protocolo HTTP

Uno de los pilares de la web se encuentra en el **Protocolo de transferencia de hipertextos (HTTP)**, que sigue el modelo cliente/servidor, que como ya se ha comentado es el usado entre un navegador y un servidor web. Realmente lo que establece HTTP es cómo recuperar hiperdocumentos distribuidos y enlazados a través de la web.

Cuando un usuario, a través de su navegador, hace una petición de acceso a un documento de una dirección dada, el HTTP solicita una conexión TCP con la que transmite al servidor la **petición HTTP**, en la que se incluye, entre otras cosas, el URL, información sobre el cliente, y la orden concreta con sus parámetros. El servidor responde realizando la operación requerida y enviando una **respuesta HTTP**, que contiene información sobre el servidor y de control, y el documento (hiperarchivo) solicitado. Una vez dada la respuesta, automáticamente se cierra la conexión TCP. Si a continuación seleccionamos un hiperenlace del documento visualizado por el navegador, se vuelve a repetir todo el proceso (nueva conexión TCP/IP, etc.), ahora con el URL asociado al enlace.

#### 8.2.4.3. Navegadores web

Un **navegador** o **explorador** (*browser*) es un programa que permite captar, interpretar y visualizar documentos web.

La captación del documento se realiza con ayuda de uno de los protocolos vistos con anterioridad, que permiten la obtención de archivos a través de Internet: HTTP, FTP, Gopher, TELNET, etc.

En un documento de texto se pueden diferenciar dos cuestiones: los caracteres que contiene y el formato y presentación de los mismos (márgenes de la página, tipo y tamaño de letra, sangrías, tabulaciones, etc.). Para especificar los caracteres existen convenciones mundialmente aceptadas como son la codificación ASCII y Unicode; sin embargo, el formato y presentación de un documento se crea por muy diferentes procedimientos que dependen notablemente del editor o procesador de textos de que se trate. Esto presenta un grave inconveniente cuando se capta un documento a través de la web, ya que es necesario visualizarlo en el computador local con el aspecto y formato correcto, independientemente del procedimiento que se utilizó para darle formato en el servidor que contiene la página. Para solucionar el problema se ha definido el estándar **Lenguaje de marcado de hipertextos (HTML)**, que contiene sólo caracteres ASCII, entre los que se insertan órdenes de formato (marcas), también en ASCII. Las marcas de formato van embebidas en el texto principal y entre ángulos (<>).

#### EJEMPLO 8. 4

La frase:

Era de **noche** y sin *embargo* llovía.

en lenguaje HTML resulta ser:

Era de <b>noche</b> y sin <i>embargo</i> llovía.

Puede observarse que con la marca <b> se da la orden de escribir en negrita, y la </b> significa final de negrita. Por otra parte <i> es inicio de letra cursiva (o itálica) y </i> es el final de este tipo de letra.

Hay navegadores que integran el servicio de transferencia de ficheros (FTP, Sección 8.2.3) de manera que en la URL sólo hay que usar en vez del protocolo **http** el protocolo **ftp** (ftp://dirección/camino).

Los documentos web pueden clasificarse en tres tipos:

- **Documentos estáticos.** Son documentos HTML con información fija. Cuando el cliente solicita la página, el servidor se la envía y el navegador interpreta su contenido que es fijo. Con frecuencia las páginas HTML se generan con pequeños programas (*scripts*) escritos en **lenguaje JavaScript**<sup>2</sup>. Existen versiones mejoradas del HTML original como la **XHTML** (HTML ampliado).
- **Documentos dinámicos.** Son documentos cuya información cambia constantemente de un momento a otro. Piénsese, por ejemplo, en una página web que presenta los valores de la bolsa en tiempo real, o la hora y temperatura de una determinada localidad. En este caso, cada vez que el usuario solicita una de estas páginas se lanza en el servidor la ejecución un programa que genera el documento a transmitir o actualiza sus partes dinámicas. Una vez generada la página se transmite al cliente.
- **Documentos activos.** Son documentos que también se generan en el momento de visualizarse, pero en lugar de hacerlo en el servidor se crean en el propio cliente. En este caso el servidor remite el programa y los datos, el cliente ejecuta el programa que genera la página y los movimientos se van actualizando por el propio programa. Los documentos activos se suelen programar en lenguaje Java.

#### 8.2.4.4. Aplicaciones de la web

Internet en general, y la web en particular, no sólo está sirviendo para obtener información como si se tratase de consultar una biblioteca, sino que ha dado lugar a nuevas posibilidades como las siguientes:

- Intercambios comerciales a través de Internet (comercio electrónico).
- Telefonía a través de Internet.
- Videoconferencia.
- Difusión de radio y televisión (en vivo).
- Audio bajo demanda.
- Vídeo bajo demanda (películas, documentales, etc.).
- Etcétera.

La web puede considerarse un medio de difusión completamente revolucionario. Cualquier empresa o persona particular, con muy pocos recursos, puede crear su propia página web que será accesible desde cualquier parte del mundo. No obstante, existen graves problemas en cuanto a que la información que se incluye en la web no pasa por ningún control o contraste de calidad o ético; y, por otra parte, existen muchos documentos incompletos (en construcción) y además la información es muy volátil ya que las referencias no son estables (cambian de lugar) y los documentos pueden ser eliminados en cualquier momento por sus creadores, o, sencillamente porque el computador que las contiene se haya dado de baja. En contraste con la web, por ejemplo, un artículo científico publicado en una revista de prestigio es revisado y contrastado por un equipo de especialistas, y la referencia de su publicación (título de la revista, año, mes y páginas) permanece en el tiempo, pudiendo ser localizado en cualquier momento.

---

<sup>2</sup> No confundir con el lenguaje Java, con el que, salvo el nombre, no tiene nada en común.

## 8.3. Conclusiones

En este capítulo se han estudiado aspectos básicos sobre las redes de computadores (Sección 8.1) e Internet (Sección 8.2).

En primero lugar hemos presentado distintas topologías para la interconexión de computadores en red (Sección (8.1.1) y el modelo de referencia OSI para interconexión de sistemas de distintas características (sistemas abiertos) (Sección 8.1.1). A continuación, en las Secciones 8.1.3 y 8.1.4 se han analizado los conceptos de redes de área local (LAN) y de redes de área amplia (WAN), mostrando las peculiaridades y características más notables de cada una de ellas.

Las redes actuales están formadas por segmentos de red y subredes más o menos complejas. Todos estos sistemas se interconectan por medio de los dispositivos o procesadores de red que se han analizado en la Sección 8.1.5.

Debido a la importancia que ha adquirido la red de redes (Internet) se ha dedicado a ella la segunda parte de este capítulo. Concretamente se ha estudiado el direccionamiento en Internet (Sección 8.2.1), los protocolos básicos que determinan su funcionamiento (Sección 8.2.2) y sus aplicaciones tanto las básicas (Sección 8.2.3) como la más compleja de web (Sección 8.2.4).

Podemos concluir que existe una importancia creciente en la interconexión de computadores, siendo un factor esencial la velocidad de comunicación, ya que cada vez más tanto la información como las aplicaciones se encuentran distribuidas en la red en lugar de en los computadores locales.



### Test

**T8.1.** El modelo OSI considera la transmisión de datos según un modelo de:

- a) Cinco capas.
- b) Seis capas.
- c) Ocho capas.
- d) Siete capas.

**T8.2.** En el modelo OSI la capa que agrupa los bits en tramas es la:

- a) Física.
- b) De enlace de datos.
- c) De red.
- d) De transporte.

**T8.3.** En el modelo OSI la capa que descompone los mensajes en segmentos es la:

- a) Sesión.
- b) De enlace de datos.
- c) De red.
- d) De transporte.

**T8.4.** En el modelo OSI la capa que hace de interfaz entre la red y el usuario final es la de:

- a) Sesión.
- b) Aplicación.
- c) Presentación.
- d) Transporte.

**T8.5.** En el modelo OSI la capa que comprime o descomprime la información es la de:

- a) Sesión.
- b) Aplicación.
- c) Presentación.
- d) Transporte.

**T8.6.** En el modelo OSI la capa que cifra o descifra la información es la de:

- a) Sesión.
- b) Aplicación.
- c) Presentación.
- d) Transporte.

**T8.7.** En el modelo OSI la capa que emite o recibe cadenas de bits (*phits*) a través del medio es la:

- a) De enlace de datos.
- b) De aplicación.
- c) Física.
- d) De transporte.

**T8.8.** En el modelo OSI la capa cuyo objetivo es transportar una trama entre nodos adyacentes es la:

- a) De enlace de datos.
- b) De aplicación.
- c) Física.
- d) De transporte.

**T8.9.** En el modelo OSI la capa cuyo objetivo es la transmisión del mensaje completo entre fuente y destino es la de:

- a) Sesión.
- b) Aplicación.
- c) Presentación.
- d) Transporte.

**T8.10.** En el modelo OSI la capa cuyo objetivo es la transmisión de un paquete entre fuente y destino es la de:

- a) Sesión.
- b) Aplicación.
- c) Red.
- d) Transporte.

**T8.11.** En el modelo OSI la capa cuyo objetivo es controlar el diálogo entre usuarios es la de:

- a) Sesión.
- b) Aplicación.
- c) Presentación.
- d) Transporte.

**T8.12.** Una NIC es:

- a) Un comité consultivo internacional sobre telegrafía y telefonía.
- b) Una confederación internacional de normalizaciones.
- c) Una unidad de datos dentro de una capa del modelo OSI.
- d) Un adaptador hardware entre un computador y una red de comunicaciones.

**T8.13.** ISO es:

- a) El comité consultivo internacional sobre telegrafía y telefonía.
- b) La confederación internacional de normalizaciones.
- c) Un modelo para interconexión en red de sistemas abiertos.
- d) Un adaptador hardware entre un computador y una red de comunicaciones.

**T8.14.** OSI es:

- a) El comité consultivo internacional sobre telegrafía y telefonía.
- b) La confederación internacional de normalizaciones.
- c) Un modelo para interconexión en red de sistemas abiertos.
- d) Un adaptador hardware entre un computador y una red de comunicaciones.

**T8.15.** Una PDU es:

- a) Un comité consultivo internacional sobre telegrafía y telefonía.
- b) Una confederación internacional de normalizaciones.
- c) Una unidad de datos dentro de una capa del modelo OSI.
- d) Un adaptador hardware entre un computador y una red de comunicaciones.

**T8.16.** Las topologías que utilizan concentradores o conmutadores son:

- a) Las en bus.
- b) Las en estrella.
- c) Las en anillo.
- d) Todas.

**T8.17.** En una red Ethernet, cuando un equipo quiere emitir:

- a) Coge el testigo que circula por la línea y envía su mensaje.
- b) Contacta con el equipo receptor y cuando éste le da paso envía el mensaje.
- c) Primero escucha y si la línea está libre coge el primer testigo libre que circule y el envía su mensaje.
- d) Primero escucha y si la línea está libre emite. Si se produce una colisión se destruyen los paquetes en conflicto y se reintenta la transmisión.

**T8.18.** En una red en anillo con paso de testigo:

- a) Un testigo indica cuándo la línea está libre.
- b) Cuando se quiere enviar un mensaje se escucha y si la línea está libre se envía el mensaje en un testigo.
- c) Se solicita al receptor autorización para enviar el mensaje, y cuando éste lo acepta envía un testigo al emisor para que envíe el mensaje.
- d) El receptor confirma al emisor la recepción de un mensaje por medio de un testigo.

**T8.19.** El dispositivo de interconexión que se limitan a transferir el tráfico entre subredes sin hacer ninguna operación adicional es el:

- a) Repetidor (*repeaters*).
- b) Puente (*bridge*).
- c) Pasarela (*gateway*).
- d) Dispositivo de encaminamiento (*router*).

**T8.20.** El dispositivo de interconexión de redes que trasfiere el tráfico pero seleccionando cuando pasa de un segmento a otro es el:

- a) Repetidor (*repeaters*).
- b) Puente (*bridge*).
- c) Pasarela (*gateway*).
- d) Dispositivo de encaminamiento (*router*).

**T8.21.** El dispositivo de interconexión de redes que direcciona los paquetes de acuerdo con la mejor ruta en un momento dado es el:

- a) Repetidor (*repeaters*).
- b) Puente (*bridge*).
- c) Pasarela (*gateway*).
- d) Dispositivo de encaminamiento (*router*).

**T8.22.** El dispositivo que interconecta redes que utilizan distintos protocolos es:

- a) El repetidor (*repeaters*).
- b) El puente (*bridge*).
- c) La pasarela (*gateway*).
- d) El dispositivo de encaminamiento (*router*).

**T8.23.** Un repetidor es un dispositivo de interconexión de redes que actúa en las siguientes capas del modelo OSI:

- a) Física.
- b) Física y de enlace de datos.
- c) Física, enlace de datos y red.
- d) Todas.

**T8.24.** Una pasarela es un dispositivo de interconexión de redes que actúa en las siguientes capas del modelo OSI:

- a) Física.
- b) Física y de enlace de datos.
- c) Física, enlace de datos y red.
- d) Todas.

**T8.25.** Un encaminador es un dispositivo de interconexión de redes que actúa en las siguientes capas del modelo OSI:

- a) Física.
- b) Física y de enlace de datos.
- c) Física, enlace de datos y red.
- d) Todas.

**T8.26.** Un puente es un dispositivo de interconexión de redes que actúa en las siguientes capas del modelo OSI:

- a) Física.
- b) Física y de enlace de datos.
- c) Física, enlace de datos y red.
- d) Todas.

**T8.27.** Una red de área local interconecta los computadores.

- a) De un campus.
- b) De una localidad (ejemplo: pueblo, ciudad).
- c) De las unidades centrales de un ordenador con sus periféricos externos.
- d) A través de líneas de una misma central telefónica.

**T8.28.** Una red de área amplia es una red:

- a) Cuyos terminales se interconectan mediante una línea Ethernet.
- b) Cuyos terminales se interconectan mediante un red en anillo.
- c) Cuyos terminales están ubicados en una zona geográfica muy extensa.
- d) De computadores ubicados en una ciudad o mancomunidad.

**T8.29.** Un módem analógico es un dispositivo que permite que un computador se conecte a una red:

- a) Telefónica digital.
- b) Telefónica analógica.
- c) ADSL.
- d) ATM.

**T8.30.** La red que utiliza un protocolo de transferencia asíncrono, especialmente diseñado para enviar voz, vídeo y datos de forma más eficiente, se denomina:

- a) RDSI.
- b) ADSL.
- c) ATM.
- d) Cable-módem.

**T8.31.** La red que utiliza cable coaxial como medio de transmisión se conoce como:

- a) RDSI.
- b) ADSL.
- c) ATM.
- d) Cable-módem.

**T8.32.** Una dirección IP es una dirección:

- a) Para identificar unívocamente a cada computador o sistema conectado a Internet.
- b) Que se escribe en la barra de direcciones de un navegador para poder visualizar una página web (<http://...>).
- c) Que sólo tienen los servidores de páginas web para estar unívocamente identificados en Internet.
- d) Para asignar a cada subred o subdominio de Internet.

**T8.33.** El Sistema de Nombres de Dominio (DNS) asigna un nombre simbólico a cada:

- a) Dirección IP.
- b) Dirección de correo electrónico.
- c) A cada red o subred de Internet.
- d) A cada página web (<http://...>).

**T8.34.** La capa de transporte de TCP/IP utiliza el protocolo:

- a) TCP o UDP.
- b) IP.
- c) FTP y TELNET.
- d) HTTP.

**T8.35.** La capa de red del conjunto TCP/IP se denomina:

- a) TCP o UDP.
- b) Internet (IP).
- c) HTTP.
- d) TELNET.

**T8.36.** Un equipo informático para conectarse a Internet necesariamente debe utilizar:

- a) El modelo OSI.
- b) El conjunto de protocolos TCP/IP.
- c) El protocolo Ethernet.
- d) Una tarjeta de red (NIC) de tipo ATM.

**T8.37.** La aplicación TELNET tiene por objeto:

- a) Interconectar redes de telecomunicaciones (TELEcommunications NETworks).
- b) Intercambiar archivos entre computadores remotos.
- c) Enviar correo electrónico.
- d) Emular un terminal virtual para acceder desde él a un computador remoto.

**T8.38.** SMTP es un protocolo para:

- a) Transferencia de hipertextos, que permite recuperar páginas web.
- b) Conexión de terminal remoto.
- c) Correo electrónico.
- d) Transferencia de ficheros (en general).

**T8.39.** Un protocolo específico para transmisión de archivos es:

- a) SMTP.
- b) FTP.
- c) HTTP.
- d) TELNET.

**T8.40.** Un protocolo específico para transferencia y acceso a documentos web es:

- a) SMTP.
- b) FTP.
- c) HTTP.
- d) IP.

**T8.41.** Un protocolo para transmisión de correo electrónico es:

- a) SMTP.
- b) FTP.
- c) HTTP.
- d) TELNET.
- e) IP.

**T8.42.** Un protocolo para conexión de terminal remoto es:

- a) SMTP.
- b) FTP.
- c) HTTP.
- d) TELNET.

**T8.43.** FTP es un protocolo ideado para:

- a) Transferencia de hipertextos, que permite recuperar páginas web.
- b) Conexión de terminal remoto.
- c) Acceder y editar datos en computadores remotos.
- d) Transferencia de archivos.

**T8.44.** ¿Cuál es el nombre de dominio de la dirección de correo electrónico jose\_rodriguez@adt.ugr.es?

- a) jose\_rodríguez.
- b) adt.ugr.es.
- c) ugr.
- d) rodríguez.

**T8.45.** La www (World Wide Web) es:

- a) Un conjunto de aplicaciones que permite la conexión a terminales remotos, transferencia de archivos, correo electrónico, chats, etc.
- b) Un conjunto de redes interconectadas con el protocolo TCP/IP.
- c) Un sistema formado por un conjunto de documentos (hipertextos) con enlaces distribuidos a lo largo de todo el mundo.
- d) Un sinónimo de Internet.

**T8.46.** En el contexto de Internet una URL:

- a) Es una Unión de Redes Locales (agrupación de redes locales para constituir una red de área amplia).
- b) Es el Localizador Uniforme de Recursos que sirve para localizar documentos en la red.
- c) Son las abreviaturas que identifican un navegador.
- d) Es el protocolo para transferencia de archivos en la red.

**T8.47.** En el contexto de Internet, HTTP:

- a) Es el lenguaje en el que se escriben las páginas web.
- b) Es el protocolo de transferencia de hipertextos, que establece cómo recuperar páginas web.
- c) Es el formato que permite hacer hiperenlaces entre los documentos web.
- d) Forma parte de la dirección IP de un ordenador conectado a Internet.

**T8.48.** Un navegador (*browser*) es un programa que permite:

- a) Elegir el computador donde se ejecuta una aplicación determinada.
- b) Transferir archivos entre computadores.
- c) Ver el contenido de un ordenador.
- d) Captar, interpretar y visualizar documentos web.

**T8.49.** Un documento web estático es un documento:

- a) Con información fija.
- b) Cuya información es imposible cambiarla, una vez creado.
- c) Que se genera estáticamente en el computador del cliente en el momento de visualizarse.
- d) Que se genera en el servidor, en el momento de solicitarlo, para visualizarlo en el cliente.

**T8.50.** Un documento web activo es un documento:

- a) Con información fija.
- b) Cuya información es imposible cambiarla, una vez creado.
- c) Que se genera en el computador del cliente en el momento de visualizarse.
- d) Que se genera en el servidor, en el momento de solicitarlo, para visualizarlo en el cliente.

**T8.51.** Un documento web dinámico es un documento:

- a) Con información fija.
- b) Cuya información es imposible cambiarla, una vez creado.
- c) Que se genera en el computador del cliente en el momento de visualizarse.
- d) Que se genera en el servidor, en el momento de solicitarlo, para visualizarlo en el cliente.



## Problemas resueltos

- P8.1.** Explicar las ventajas tiene realizar una compresión de información antes de transmitir un mensaje. ¿Por qué se realiza en la capa de presentación y no en una inferior?

### SOLUCIÓN

Para un canal y sistemas dados el tiempo de transmisión depende del tamaño del mensaje a transmitir, cuanto mayor es éste, más tiempo dura la transmisión. Podemos reducir el tamaño de un mensaje comprimiéndolo, luego en el receptor se descomprime recuperándose el mensaje original.

La compresión/descompresión se realiza en la capa de presentación definida por el modelo OSI, o en el conjunto TCP/IP en la capa de aplicación. La compresión/descompresión debe realizarse en una de las capas superiores, pues así el trabajo que deben realizar las capas restantes es mucho menor: para un mensaje dado habrá menos segmentos, paquetes, tramas, phits, etc.

- P8.2.** Suponga que un protocolo de una red establece unas tramas en las que, por término medio, un 60 por 100 de las mismas contienen los datos del usuario y el resto es información de control (cabeceras, información para detección de errores, etc.). La velocidad de trámites de datos (del usuario) que se obtiene en la red anterior es de 500 Mbps.

- Suponiendo que se modificase el protocolo de forma tal que el porcentaje de datos del usuario se incrementase hasta el 90 por 100, obtener la nueva velocidad de transmisión de datos obtenible.
- Dibuje una gráfica que muestre la dependencia de la velocidad de transmisión de los datos con el porcentaje de datos útiles en las tramas.
- ¿Qué modo de transmisión de redes públicas se fundamenta, entre otros conceptos, en la reducción de la sobrecarga de información de control para ofrecer a los usuarios mayores velocidades de transmisión?

### SOLUCIÓN

- Incrementar el 90 por 100 los datos de usuario en cada trama supone reducir del 40 al 10 por 100 la información de control o sobrecarga de información. Por otra parte, para una misma cantidad de información transmitida a través de la red, los datos de usuario se incrementa del 60 al 90 por 100; es decir, los datos de usuario se incrementan en un 50 por 100; con lo que la velocidad de transmisión de datos se incrementa a:

$$v_{\text{datos}} = 500 + 0,5 \times 500 = 750 \text{ Mbps}$$

- Obviamente la velocidad total de transmisión de información por el canal,  $v_t$ , será la suma de las velocidades de transmisión de información de control,  $v_{\text{control}}$ , y de los datos,  $v_{\text{datos}}$ ; es decir:

$$v_t = v_{\text{datos}} + v_{\text{control}}$$

Pero, por otra parte, la tasa de datos transmitidos por segundo,  $\tau$ , con respecto al total de información transmitida por segundo será:

$$\tau = \frac{v_{\text{datos}}}{v_{\text{datos}} + v_{\text{control}}}$$

Eliminando  $v_{\text{control}}$  de las dos ecuaciones anteriores, se tiene:

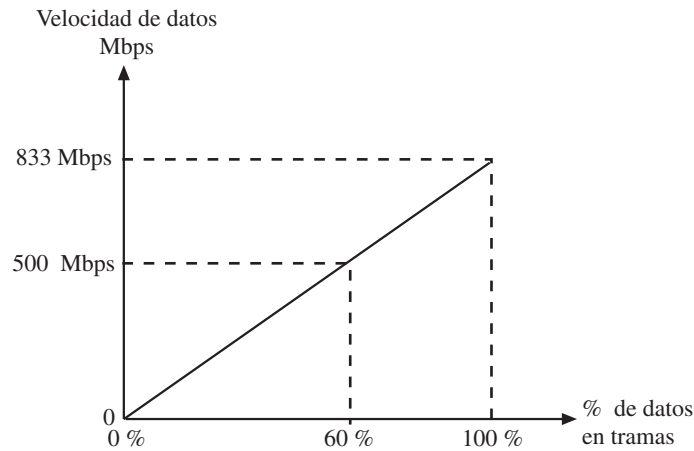
$$v_{\text{datos}} = \tau \times v_{\text{total}}$$

Conclusión que es evidente, ya que nos indica que la velocidad de transmisión de datos es directamente proporcional al porcentaje que haya de ellos. En nuestro caso, sabemos por el enunciado del problema que con  $\tau = 0,6$  se tiene  $v_{\text{datos}} = 500$  Mbps, con lo que de la expresión anterior se tiene que  $v_t \approx 833$  Mbps. En consecuencia, la gráfica que debemos representar es la correspondiente a la siguiente ecuación:

$$v_{\text{datos}} = 833 \times \tau$$



En la Figura 8.10 se representa la gráfica solicitada.

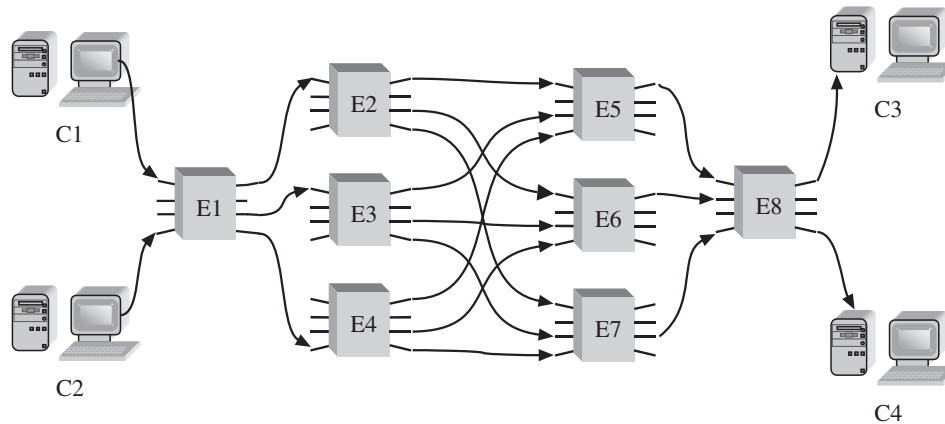


**Figura 8.10.** Variación de la velocidad de transmisión de datos con el aprovechamiento del contenido de las tramas por datos.

c) El Modo de Transmisión Asíncrono (ATM).

**P8.3.** Dada la porción de red mostrada en la Figura 8.11:

- Indicar los caminos posibles de interconexión entre los equipos C1 y C3.
- ¿Qué dispositivos de red y en qué capa del modelo OSI se decide el camino o ruta de un paquete?



**Figura 8.11.** Ejemplo de interconexión de equipos a través de encaminadores.

#### SOLUCIÓN

- Primer camino: C1 → E1 → E2 → E5 → E8 → C3

Segundo camino: C1 → E1 → E2 → E6 → E8 → C3

Tercer camino: C1 → E1 → E2 → E7 → E8 → C3

Cuarto camino: C1 → E1 → E3 → E5 → E8 → C3

Quinto camino: C1 → E1 → E3 → E6 → E8 → C3

Sexto camino: C1 → E1 → E3 → E7 → E8 → C3

Séptimo camino: C1 → E1 → E4 → E5 → E8 → C3

Octavo camino: C1 → E1 → E4 → E6 → E8 → C3

Noveno camino: C1 → E1 → E4 → E7 → E8 → C3



- b) Los dispositivos que establecen el camino a seguir (E1 a E8, en la Figura 8.11) se denominan encaminadores (*routers*). Los caminos a seguir se determinan en la capa de red. La capa de enlace se encarga de que un paquete llegue bien entre dos encaminadores sucesivos.

**P8.4.** Indicar las diferencias entre una tarjeta de red y un módem.

#### SOLUCIÓN

Una tarjeta de red contiene los circuitos necesarios para conectar un equipo informático a una red de transmisión de datos digital. Un módem, sin embargo, es un adaptador para transmitir o recibir señales digitales a través de una red analógica.

**P8.5.** Suponiendo una red en anillo de 32 computadores, en la que cada computador recibe los paquetes del computador anterior, los analiza y luego los reexpide al computador siguiente, indicar los problemas de fiabilidad que presenta este tipo de red.

#### SOLUCIÓN

El problema es que como se averíe un equipo de la red, deja de funcionar toda ella, hasta que no se repare o se desconecte el equipo deteriorado de la red.

**P8.6.** Cuántos bits deberían tener las direcciones IP para poder disponer de un equipo conectado a Internet por cada metro cuadrado del globo terráqueo. *Nota:* Considerar que el metro, aproximadamente, es la diez milonésima parte del cuadrante de un meridiano terrestre.

#### SOLUCIÓN

Primero debemos calcular la superficie terrestre en metros cuadrados; este valor nos dará el número,  $m$ , de direcciones IP que necesitamos. Conociendo  $m$ , el número de bits,  $n$ , será el menor entero que verifique  $m = 2^n$ .

Longitud de la circunferencia de la Tierra:

$$L = 2 \cdot \pi \cdot R$$

luego

$$R = \frac{4 \cdot 10 \cdot 10^6}{2 \cdot \pi} = 6,37 \cdot 10^6 \text{ m}$$

con lo que la superficie de la Tierra, aproximadamente, es:

$$S = \pi \cdot R^2 = 127,32 \cdot 10^{12} \text{ m}^2$$

Es decir, hay que codificar en binario,  $127,32 \cdot 10^{12}$  combinaciones. Es decir, el número  $n$  de bits será:

$$n \geq 3,32 \cdot \log(127,32 \cdot 10^{12}) = 3,32 \cdot 12 \cdot \log(127,32) = 3,32 \cdot 12 \cdot 2,1 = 83,85$$

Esto es, serían necesarios 84 bits. En consecuencia, con 84 bits podríamos tener direcciones suficientes para ubicar en la superficie terrestre un equipo IP por metro cuadrado.

**P8.7.** Comparar las ventajas e inconvenientes de los protocolos TCP y UDP.

#### SOLUCIÓN

Ambos protocolos actúan en la capa de transporte. El protocolo TCP es más fiable y seguro, ya que se hacen más comprobaciones y controles para que la transmisión sea segura. Sin embargo, el protocolo UDP es menos fiable, no ocupándose, por ejemplo, de confirmar al emisor la llegada de la información al receptor. La ventaja de UDP es que las transmisiones con este protocolo son más rápidas que con TCP. Cuando la información a transmitir no es crítica se usa UDP.

**P8.8.** La InterNIC es una organización que coordinada y asigna conjuntos de direcciones IP. Las asignaciones pueden ser de tres tipos:

- **Clase A.** El primero de los cuatro bytes empieza siempre por 0 e identifica la red (dominio). Los restantes bits, hasta 32, especifican una dirección de computador dentro de cada una de las redes.

- **Clase B.** Todas las direcciones empiezan por 10, y los dos primeros bytes especifican la red. Los restantes bits, hasta 32, especifican una dirección de computador dentro de cada una de las redes.
- **Clase C.** Todas las direcciones empiezan por 110, y los tres primeros bytes identifican la red. El último byte especifica un computador dentro de cada una de las redes.

Dada la dirección IP

7B.4A.54.37

indicar:

- a) La clase a la que pertenece la dirección.
- b) El dominio al que pertenece esa dirección.
- c) El código del computador dentro de su dominio.

SOLUCIÓN

- a) El primer byte de la dirección es 01111011, con lo que como comienza por 0 es de Clase A.
- b) Al ser de clase A, el resto de bits del primer byte especifican el dominio al que pertenece la dirección: 1111011 = dominio 124.
- c) Esta subred o dominio puede tener un gran número de dispositivos conectados a ella, ya que se reservan 3 Bytes para asignar direcciones. En este caso la dirección del computador dentro de la subred es:
- d)  $4A.54.37 \rightarrow 0100\ 1010\ 0101\ 0100\ 0011\ 0111$

## Problemas propuestos



**P8.9.** Indicar las capa que según el modelo OSI deberían encargarse de las siguientes tareas:

- a) Envío y recepción de una transacción o mensaje.
- b) Cambios de códigos.
- c) Comprimir/descomprimir la información.
- d) Establecer y finalizar una sesión.
- e) Formar paquetes a partir de segmentos y formar segmentos a partir de paquetes.
- f) Realizar el encaminamiento de los distintos paquetes a través de los dispositivos de la red.
- g) Transmitir sin error cadenas de bits en forma de señales ópticas, eléctricas, etc.

**P8.10.** Reproducir esquemáticamente el intercambio de información entre dos equipos B y D de una red local en anillo (*token ring*) que está constituida por cinco equipos (A, B, C, D, E). Suponer la transmisión de varias tramas (algunas de ellas llegando al receptor incorrectamente).

**P8.11.** Dada la porción de red mostrada en la Figura 8.11, indicar los caminos posibles de interconexión entre los equipos C2 y C4.

**P8.12.** En un domicilio particular al que llega cable-coaxial de una compañía que ofrece servicios de Internet se ha instalado un cable-módem, al que se ha conectado un punto de acceso que actúa de encaminador inalámbrico. De esta forma se dispone de una red con 4 equipos PC y una impresora, todos

ellos con conexión de red inalámbrica. Indicar, de forma razonada:

- a) La topología de la red.
- b) El tipo de red (LAN, WAN o MAN).

**P8.13.** Se dispone de una LAN de tipo Ethernet, bus único, en la que están conectados 64 equipos. Indicar de forma razonada de qué dependerá la frecuencia de que se den colisiones. ¿Cómo se podría reducir el número de colisiones?

**P8.14.** Indicar, justificándolo, cuál es la topología (estrella, bus único o en anillo) de una red LAN más robusta frente a una avería de una parte de la red.

**P8.15.** Hacer una relación de las funciones que debe realizar:

- Un repetidor.
- Un puente.
- Un enrutador.
- Una pasarela.

**P8.16.** Se observa que el porcentaje de tráfico (paquetes intercambiados) entre los 8 equipos de una red de área local es el que se indica en la tabla que se muestra en la página siguiente:

Para mejorar el sistema se decide dividirla en dos subredes e interconectarlas con un puente. Obtener:

- a) Una tabla con el porcentaje de tráfico en que está involucrado cada equipo.

- b) Los equipos que deben interconectarse a la subred A y a la subred B para que ambas subredes estén equilibradas.
- c) El porcentaje del tráfico total que tendrá que dejar pasar el puente de la subred A a la subred B, y viceversa.
- d) La disminución de tráfico de cada subred, con respecto a la red inicial.

	E0	E1	E2	E3	E4	E5	E6	E7	TOTAL
E0	—	—	—	—	—	—	—	—	—
E1	3	—	—	—	—	—	—	—	3
E2	2	4	—	—	—	—	—	—	6
E3	3	1	4	—	—	—	—	—	8
E4	9	6	5	4	—	—	—	—	24
E5	2	2	1	5	7	—	—	—	17
E6	1	1	2	9	9	2	—	—	24
E7	0	1	2	2	7	3	3	—	18
Total	20	15	14	20	23	5	3	—	100

**P8.17.** Indicar las diferencias y similitudes entre el modelo OSI y el conjunto de protocolos TCP/IP.

**P8.18.** Teniendo en cuenta la asignación de direcciones IP que hace la InterNIC (Problema P8.8), obtener:

- a) El número de redes direccionables dentro de cada clase.
- b) El número total de computadores direccionables dentro de una red de cada una de las clases.
- c) El número total de redes y de computadores direccionables por Internet.

**P8.19.** Teniendo en cuenta la asignación de direcciones IP que hace la InterNIC (Problema P8.8), dada la dirección IP:

138.205.154.8

indicar:

- a) La clase a la que pertenece.
- b) El dominio al que pertenece esa dirección.
- c) El código del computador dentro de su dominio.

**P8.20.** Obtener la dirección en notación de punto decimal de la dirección IP binaria:

3C7F C352

**P8.21.** Obtener la dirección IP binaria de la siguiente dirección IP en notación de punto decimal:

123.241.122.15

**P8.22.** Indicar las diferencias y similitudes entre una dirección IP, una dirección de correo electrónico y una dirección URL.

**P8.23.** Dada la de la siguiente dirección URL:

<http://www.rne.es/rc/index.htm>

acceder a ella e indicar:

- a) El significado de los elementos que contiene.

b) El tipo de documento que es (estático, dinámico o activo).

c) Institución a que pertenece.

**P8.24.** Dada la de la siguiente dirección URL:

<http://www.la-moncloa.es/web/pg06b.htm>

acceder a ella e indicar:

- a) El significado de los elementos que contiene.
- b) El tipo de documento que es (estático, dinámico o activo).
- c) Institución a que pertenece.

**P8.25.** A qué capas y en qué modelo se utilizan las siguientes unidades de datos de protocolo (PDU):

- Transacción.
- Mensaje.
- Paquete.
- Segmento.
- Datagrama de usuario.
- Datagrama IP.
- Phit.
- Trama.

**P8.26.** Indicar el significado de las siglas y para qué sirven los siguientes protocolos:

- HTTP.
- FTP.
- Gopher.
- SMTP.
- DNS.
- RTP.
- UDP.
- TCP.
- IP.
- TELNET.

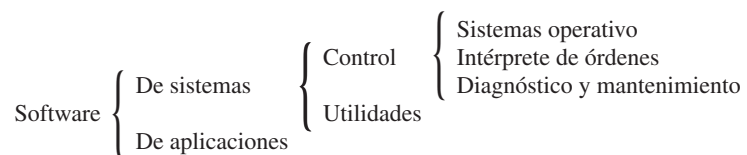
# Sistemas operativos

Este capítulo se dedica al sistema operativo, que puede considerarse como una interfaz entre la máquina y los usuarios o programas de aplicación. El objetivo fundamental de los sistemas operativos es gestionar y administrar eficientemente los recursos hardware, permitiendo que se ejecuten concurrentemente varios programas, sin que haya conflictos en el acceso de cada uno de ellos a los distintos recursos que necesiten, y sin que ningún programa monopolice ninguno de ellos.

El capítulo se inicia (Sección 9.1) encuadrando a los sistemas operativos dentro del software general de un computador. Posteriormente (Sección 9.2) se define el concepto de Sistema Operativo y se da una pequeña pincelada sobre la evolución de los mismos. El resto de apartados se dedican a analizar cómo el sistema operativo gestiona los distintos recursos de un computador: el procesador (Sección 9.3), la memoria principal (Sección 9.4), las entradas/salidas (Sección 9.5) y los archivos (Sección 9.6).

## 9.1. El software de un computador

El **software o soporte lógico de un computador** es el conjunto de programas asociados a dicho computador. Dentro de estos programas se incluyen los suministrados por el constructor, los adquiridos en empresas especializadas en venta de programas y los redactados por los propios usuarios. Según se muestra en la Figura 9.1, los programas o componentes que forman el software de un computador pueden agruparse en dos grandes apartados: software del sistema y software de aplicación.



**Figura 9.1.** Componentes del software de un computador.

### SOFTWARE DEL SISTEMA

El software del sistema incluye todos los programas que realizan tareas comunes al computador, considerado en su globalidad. Está constituido por el software de control y las utilidades.

- **Software de control o sistema de explotación.** Controla el funcionamiento de los programas que se ejecutan y administra los recursos hardware, facilitando el uso del computador de la forma más eficiente posible. Dentro de este apartado se incluye el sistema operativo, el intérprete del lenguaje de control y el software de diagnóstico y mantenimiento. A los dos primeros les dedicaremos el resto del capítulo.

El **software de diagnóstico y mantenimiento** está formado por los programas que utilizan las personas responsables del mantenimiento e instalación del hardware y del software del computador, para localizar automáticamente las averías o las causas de un mal funcionamiento de algún dispositivo o de algún módulo del sistema operativo. Los programas de mantenimiento sirven también para instalar un nuevo sistema.

- **Utilidades.** Son un conjunto de programas de servicio que, en cierta medida, pueden considerarse una ampliación del sistema operativo. Incluyen programas para realizar tareas tales como compactación de discos o reubicación de archivos dentro de un disco para poder acceder a ellos más rápidamente, gestión de comunicaciones a través de tarjeta de red o de módem, visualizadores y navegadores de Internet, programas para respaldo de seguridad, compresores de datos, recuperación de archivos erróneamente borrados, antivirus, salvapantallas que evitan imágenes fijas durante largos períodos de tiempo que pueden deteriorar la pantalla, etc. También se incluyen aquí herramientas generales que facilitan la construcción de las aplicaciones de los usuarios, sea cual sea la naturaleza de estas, tales como intérpretes, compiladores, editores de texto, y cargadores/montadores.

## SOFTWARE DE APLICACIÓN

Está constituido por el conjunto de programas que realizan tareas y aplicaciones concretas y que son el objeto último de la utilización del computador por parte de los usuarios. Incluye programas relacionados con aplicaciones específicas, como pueden ser programas de nóminas, control de existencias, control de clientes, contabilidad, procesadores de texto, bibliotecas de programas para resolver problemas estadísticos (BMDP, STAT-PACK, SPSS, por ejemplo) o de cálculo numérico (IMSL, MATH-PACK, por ejemplo), sistemas de administración de archivos y de bases de datos, etc. También aquí se incluyen los propios programas realizados por los usuarios.

## 9.2. Definición y evolución de sistemas operativos

Un **sistema operativo** es un *programa* (o conjunto de programas) *de control* que tiene por objeto *facilitar* el uso del computador y conseguir que éste se utilice *eficientemente*.

Es un *programa de control*, ya que se encarga de gestionar y asignar los recursos hardware que requieren los programas. Los recursos hardware son: el procesador, la memoria principal, los discos y otros periféricos. El sistema operativo *facilita el uso del computador*, haciendo *transparente* al usuario las características hardware concretas de los dispositivos. El sistema operativo también *hace que el computador se utilice eficientemente*; así, por ejemplo, un sistema operativo de multiprogramación hace que cuando un *Programa A* se esté ejecutando, y tenga una operación de E/S, el tiempo muerto de uso del procesador sea aprovechado para la ejecución de otro *Programa B*.

El sistema operativo puede considerarse como un programa formado por una serie de módulos, que se lanzan a ejecución por medio de **llamadas al sistema**. Las llamadas al sistema pueden ser realizadas por los usuarios (a través de las órdenes de un **lenguaje de control**), por los programas o por las interrupciones. Las órdenes son generadas por el usuario por medio de líneas de texto (**líneas de órdenes**), o escogiendo opciones de un menú o por medio de la selección de iconos. Una orden dada (por ejemplo, para visualizar los archivos contenidos en una carpeta) es captada por el **intérprete de órdenes** o **concha** (*shell*), que es un programa independiente del sistema operativo que traduce o des-

compone la orden en llamadas al sistema. Los intérpretes de órdenes modernos utilizan interfaces de usuario gráficas (**GUI**, “*Graphical User Interface*”).

Se pueden considerar cuatro etapas o generaciones relacionadas con los sistemas operativos.

### Primera etapa

En la primera etapa (1943 a 1955), o *prehistoria* de los sistemas operativos, la introducción y control de la ejecución de programas se hacía manualmente, uno a uno: no existían sistemas operativos.

### Segunda etapa

En la segunda etapa (aproximadamente de 1956 a 1963) en vez de utilizar sólo lectoras de tarjetas e impresoras, se hace uso de soportes de información intermedios, como cintas magnéticas. Los trabajos, al introducirse en el computador, se iban grabando en una cinta magnética, uno detrás de otro, formando una cola de trabajos. Un programa, denominado **monitor**, se encargaba de leer el primer trabajo de la cola, cargarlo en memoria y lanzarlo a ejecución, así como de gestionar y contabilizar el consumo de recursos, y cuando concluía daba paso al siguiente programa de la cola. Los resultados de los programas también se iban grabando en una cinta magnética formando una cola de salida (*spooler*), cuyo contenido posteriormente se imprimía (a veces incluso en otro computador). Este tipo de funcionamiento, denominado por *lotes* o *lotes-serie* o *cola-serie* o por *cola de trabajos* (en inglés, *batch* o *batch-serial*), era posible gracias a la existencia de un sistema operativo; es decir, un programa de control. Este programa de control que planifica los trabajos se suele denominar **monitor**. Con el tiempo las colas de trabajos de entrada y colas de impresión de salida se almacenaban en tambores o discos magnéticos, mejorando notablemente el rendimiento de los computadores.

Poco a poco los monitores de lotes se perfeccionaron, incluyendo rutinas de gestión de E/S y bibliotecas de programas utilizables por cualquier usuario, módulos controladores del tiempo máximo de procesador (para evitar que un programa se ejecute por tiempo indefinido, a causa de un error de programación), módulo de contabilidad del uso del computador por los usuarios, cargadores, etc. Los monitores constituyeron los primeros sistemas operativos.

### Tercera etapa

La tercera etapa (1963 a 1979) se caracteriza fundamentalmente por el perfeccionamiento del sistema de trabajos por lotes, el desarrollo de sistemas operativos en **multiprogramación** (1963), que permite ejecutar fragmentos sucesivos de distintos programas cargados en la memoria aprovechando mejor los distintos recursos, la introducción del concepto de **memoria virtual** (1972) y la utilización compartida de dispositivos de entrada salida (*spooler*). Estas técnicas se describirán más adelante, en este capítulo. Los sistemas operativos debían planificar o asignar los recursos a los distintos programas en ejecución.

### Cuarta etapa

La cuarta etapa (aproximadamente entre 1980 y la actualidad) se caracteriza por el desarrollo de sistemas operativos, de uso ampliamente extendido, para computadores personales (CP/M, MS-DOS, OS/2, etc.) y para estaciones de trabajo (UNIX, Windows-NT, etc.).

Se desarrollan los siguientes tipos de sistemas operativos:

- **Sistemas operativos en red**, que incluyen programas de control de interfaz con la red, permitiendo establecer una sesión de trabajo con un computador remoto, acceder a los recursos de éste y copiar archivos de un computador a otro.
- **Sistemas operativos de multiprocesamiento**, que actúan en computadores con varios procesadores (**multiprocesadores**) compartiendo la misma memoria principal y facilidades de E/S.

Estos sistemas distribuyen y planifican los programas y trabajos entre los distintos procesadores, sincronizando las relaciones entre ellos, dando la apariencia al usuario de disponer de un solo procesador.

- **Sistemas operativos distribuidos**, que permiten a un usuario ejecutar, de forma transparente como si estuviese actuando con un único computador, uno o varios programas en varios computadores (**multicomputadores**) y con datos distribuidos en diferentes servidores de archivos, lo cual requiere que el sistema operativo disponga de un sistema de gestión de archivos distribuido y de sofisticados algoritmos de planificación que paralelicen adecuadamente el trabajo a realizar.

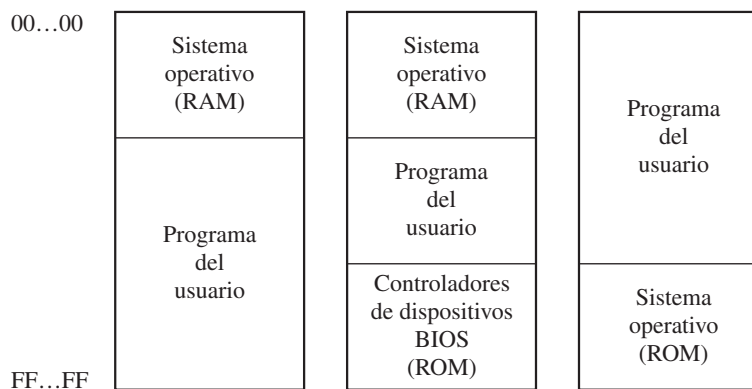
Los sistemas operativos actuales tienen por objeto gestionar o administrar los siguientes recursos: el procesador, la memoria principal, los periféricos de entrada y salida y los archivos. Las siguientes secciones se dedican a describir cómo realizan estas actividades.

### 9.3. Gestión del procesador

Existen dos formas básicas de trabajar con un computador: **por lotes** (o cola de trabajos, Sección 9.2) y en forma **interactiva**. En esta última el procesador está constantemente atendiendo al usuario en una forma que podría denominarse de *diálogo*.

La gestión del procesador por parte del sistema operativo se centra en la abstracción de proceso. Un **proceso** es un programa en el que se ha iniciado su ejecución y que es tratado por el sistema operativo como un todo. Un programa por sí mismo es un ente pasivo, mientras que un proceso es un ente activo.

Los primeros sistemas operativos se denominaban de **monoprogramación** o **serie**, en ellos se ubicaba un programa en memoria principal, además del sistema operativo (Figura 9.2), y hasta que no finaliza la ejecución de dicho programa no empezaba a ejecutarse otro.



**Figura 9.2.** Ejemplo de ocupación de memoria en un sistema monoprogramación.

Un sistema operativo de monoprogramación realiza las siguientes funciones:

- Hace una llamada al sistema operativo siempre que un programa realiza una operación de E/S, para que aquél se encargue de ésta.
- Cuando finaliza una operación de E/S el periférico correspondiente genera una interrupción que provoca una llamada al sistema operativo, y continúa la ejecución del programa.
- Cuando acaba la ejecución de un programa, el computador queda listo para aceptar un nuevo programa, ya sea interactivo o de la cola serie. En este último caso, el **planificador a largo plazo del sistema operativo** (también denominado **planificador de trabajos**), selecciona (de



acuerdo con algún criterio de prioridad) un programa de la cola, lo carga en la memoria principal e inicia su ejecución.

Claramente se observa que la monoprogramación no es eficiente, ya que se desaprovechan tanto la memoria principal como el procesador y los periféricos. Por lo general una gran parte de la memoria estará sin utilizar, y cuando hay una operación de entrada o salida en algún periférico el procesador y el resto de periféricos se encuentran desocupados; y al revés, si el procesador está ejecutando el proceso, todos los periféricos estarán inactivos.

Para conseguir mayor eficiencia en la utilización de los distintos recursos que ofrece un computador se desarrolló la multiprogramación.

### 9.3.1. MULTIPROGRAMACIÓN

En el contexto de la multiprogramación, los componentes típicos de un proceso son:

1. **Código máquina** del programa del usuario a ser ejecutado.
2. **Datos** del programa, que contiene las variables y parámetros del programa.
3. **Pila**, cada proceso tiene asociadas una o varias pilas, utilizadas para almacenar direcciones de retorno de subrutinas u otros parámetros.
4. **Bloque de control del proceso (PCB, Process Control Block)**, que contiene parámetros y datos que el sistema operativo necesita para controlar el proceso (identificador, estado, privilegios y prioridades del proceso, identificador del usuario, contenidos de los registros del procesador, etc.).

El sistema operativo UNIX introdujo el concepto de **hebra** (*thread*), soportado por los sistemas operativos modernos. Un proceso puede descomponerse en distintas hebras o tareas diferentes e independientes, que se pueden ejecutar “en paralelo” (**concurrentemente**) entre ellas e incluso con las de otros procesos. Cada hebra tiene asociado su estado, prioridad, privilegios, etc. En adelante la mayor parte de los conceptos que se refieran a los procesos son también de aplicación a las hebras.

#### EJEMPLO 9.1

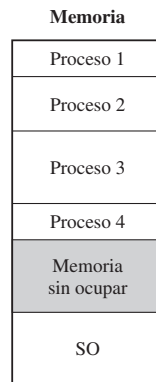
El Windows 95 asocia a cada ventana de carpeta visualizada en pantalla una hebra independiente, de forma que, por ejemplo, pueden estar realizándose en paralelo dos copias de archivo en ventanas diferentes. También, por ejemplo, al lanzar a ejecución el procesador de textos Word se generan distintas hebras, cada una de ellas con distinto cometido: una para visualizar el texto en la pantalla, otra para atender las entradas del teclado, otra para el corrector ortográfico, otra para visualizar los menús, etc.

Se dice que un proceso entra en **estado bloqueado** cuando el procesador no puede continuar trabajando con él a causa de tener que esperar la realización de una operación de entrada/salida o de algún otro evento de naturaleza similar. Un proceso se dice que está en **estado activo** cuando el procesador está ejecutando instrucciones de él. Se dice que un proceso (o hebra) está en **estado preparado** cuando el procesador puede iniciar o continuar su ejecución.

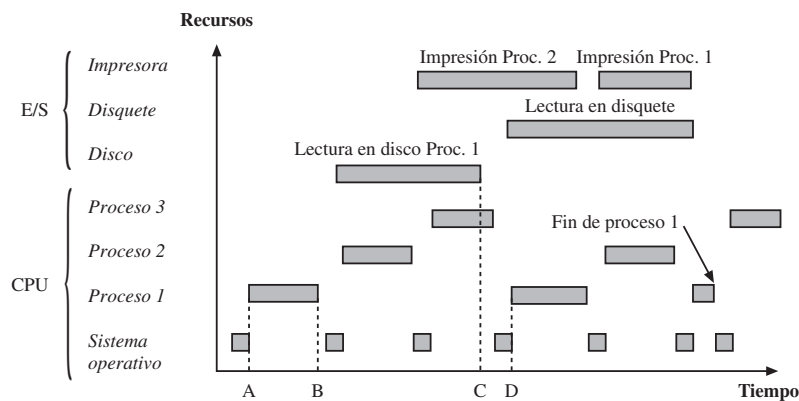
Un **sistema operativo de multiprogramación** carga en memoria principal uno o varios programas de la cola serie y todos los programas interactivos que se le presenten (Figura 9.3), y el **planificador de trabajos a corto plazo** (o **distribuidor**) del sistema operativo asigna el procesador sucesivamente a los procesos preparados de forma que el procesador se aproveche al máximo y los distintos procesos vayan avanzando en su ejecución sin necesidad de que finalice completamente uno para iniciar la ejecución de otro (Figura 9.4).

Obsérvese que con la multiprogramación la ejecución de los procesos se va solapando en el tiempo, y desde el punto de vista del usuario, como la velocidad interna del procesador es muy grande, parece que van ejecutándose simultáneamente (en paralelo). En realidad no es así, ya que al existir un





**Figura 9.3.** Ejemplo de ocupación de memoria en un sistema de multiprogramación.



**Figura 9.4.** Esquema de distribución del tiempo entre el procesador y periféricos de E/S en la ejecución de tres procesos (P1, P2 y P3) en un sistema multiprogramación no apropiativa.

único procesador, éste únicamente puede estar trabajando con un solo proceso (el proceso activo). Con rigor se dice que el procesador está ejecutando **concurrentemente** (en vez de *simultáneamente*) varios procesos: en un intervalo de tiempo determinado se han estado ejecutando alternativamente a “trozos” varios procesos ubicados en la memoria central.

### EJEMPLO 9.2

En la Figura 9.4 en el intervalo AB el Proceso 1 está en estado activo, en el intervalo BC bloqueado, y en el CD preparado.

Siempre que un proceso se bloquea o concluye, el núcleo del sistema operativo toma el control del procesador para realizar las acciones oportunas. Cuando hay multiprogramación y se detiene la ejecución de un proceso,  $P_x$ , para dar turno a otro,  $P_y$ , (que presumiblemente se había interrumpido previamente), el sistema operativo realiza un **cambio de contexto** consistente en:

1. Actualizar el bloque de control  $PCB-P_x$  del proceso  $P_x$  interrumpido; es decir, se cambia su estado (de activo a bloqueado o preparado) y se salvaguardan los contenidos de los registros del procesador, biestables indicadores, punteros a archivos de discos y punteros de las pilas, etc.
2. Restaurar los contenidos de los registros del procesador, biestables indicadores, punteros a archivos de discos, punteros de las pilas, etc., con los valores del  $PCB-P_y$  del proceso  $P_y$  que va a continuar su ejecución, y cambiar el estado del proceso a activo; de esta forma continúa sin problema la ejecución del proceso  $P_y$  previamente interrumpido.

Obsérvese que de esta forma el proceso  $P_x$ , cuando el distribuidor le dé nuevamente el turno, podrá reanudar su ejecución justo en el punto exacto en que se interrumpió.

Por lo general los sistemas operativos disponen de una técnica denominada **intercambio** (*swapping*), según la cual, cuando es necesario llevar a la memoria principal un proceso o hebra, y no cabe, se pasa temporalmente alguno de los procesos que estén inactivos (bloqueado o preparado) de memoria principal a disco, haciendo sitio así al nuevo proceso. Cuando al proceso **intercambiado en disco** le llegue su turno el **planificador de trabajos a medio plazo** lo volverá a cargar en memoria.

### 9.3.2. MEDIDAS DE PRESTACIONES DE UN SISTEMA

Para medir la eficiencia de un sistema se suelen utilizar los siguientes parámetros:

- **Rendimiento:** es el número de trabajos ejecutados durante un período de tiempo preestablecido (12 horas, por ejemplo). Representa una medida del aprovechamiento del computador, en su totalidad.
- **Tasa de utilización del procesador:** es, para un determinado tiempo, el tiempo de uso del procesador con respecto al tiempo total:

$$P = \frac{\text{Tiempo de uso de procesador}}{\text{Tiempo total}}$$

- **Tiempo de ejecución de un proceso,  $e$ :** es el tiempo transcurrido desde que se inicia su ejecución hasta que concluye. Se puede obtener como suma de los siguientes tiempos:
  1. Tiempo de procesamiento (suma de tiempos en que el proceso está activo),  $s$ .
  2. Tiempo de E/S (suma de tiempos en que el proceso está bloqueado),  $w_1$ .
  3. Tiempo de espera (suma de tiempos en que el proceso está preparado, esperando a que le llegue el turno),  $w_2$ .
- **Tiempo en máquina,  $q$ :** es el tiempo que tarda el sistema en responder a la solicitud de realización de una tarea determinada. En procesos por lotes se suele denominar **tiempo de retorno**, y es la suma de:
  1. Tiempo de espera en cola de entrada,  $w_0$ .
  2. Tiempo de ejecución,  $e$ .
  3. Tiempo en la cola de salida (*spooler* de salida),  $w_3$ .

El tiempo en máquina en procesos interactivos se suele denominar **tiempo de respuesta**, y se define como el intervalo de tiempo entre el instante en que el usuario da una orden y el instante en que se visualiza la respuesta.

- **Coefficiente de respuesta:** es el cociente

$$R = \frac{\text{Tiempo en máquina}}{\text{Tiempo de procesamiento}} = \frac{q}{s}$$

y representa una medida de la lentitud del computador en responder al usuario.

### 9.3.3. MODOS DE ASIGNACIÓN DEL PROCESADOR A LOS PROCESOS

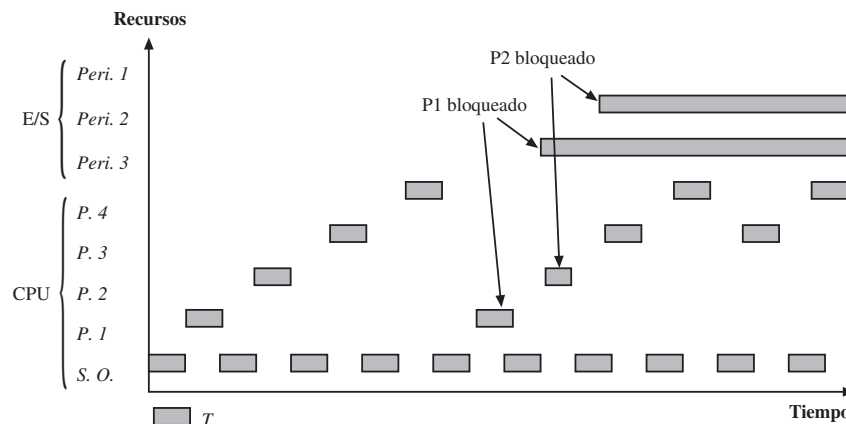
Teniendo en cuenta cómo se toma la decisión de ir alternando la ejecución de los procesos, la multiprogramación se puede realizar de dos formas básicas: no apropiativa y apropiativa.

### *Multiprogramación no apropiativa (“Nonpreemptive”)*

En este caso (Figura 9.4), una vez que un proceso está activo, continúa ejecutándose hasta que: 1) termina, o 2) se bloquea por el inicio de una operación de E/S o de cualquier otro servicio solicitado al sistema operativo, o 3) el propio proceso hace una llamada al sistema operativo para ceder el procesador a otro proceso. En este último caso se dice que el sistema es de **multiprogramación cooperativa**. En el momento de interrumpirse la ejecución del proceso el distribuidor, de acuerdo con algún **algoritmo de planificación**, da el turno a otro de los procesos pendientes que se encuentren en el estado preparado.

### *Multiprogramación apropiativa o preferente (“Preemptive”)*

En la **multiprogramación apropiativa** el sistema operativo puede interrumpir en cualquier momento el proceso activo *apropiándose* del procesador con objeto de dar paso a otro proceso que esté preparado. El sistema operativo detiene la ejecución de un proceso de acuerdo con algún criterio preestablecido, por ejemplo cuando transcurre un determinado tiempo (Figura 9.5) o cuando se presenta un proceso de mayor prioridad, como veremos a continuación. Las decisiones de cuándo se detiene la ejecución de un proceso y de cuál de los procesos preparados pasa a activo se efectúa de acuerdo con un determinado algoritmo de planificación.



**Figura 9.5.** Esquema de distribución del tiempo entre el procesador y periféricos de E/S en la ejecución de cuatro procesos (P1, P2, P3 y P4) en un sistema multiprogramación con turno rotatorio.

## 9.3.4. ALGORITMOS DE PLANIFICACIÓN

Si existen varios procesos preparados, un problema que debe resolver el distribuidor es elegir a cuál de ellos darle el turno (pasándolo a activo). El **distribuidor** se encarga de implementar un **algoritmo de planificación**. Cada proceso suele tener asociada una prioridad, existiendo **prioridades estáticas**, dadas, por ejemplo, en función de la relevancia de los procesos (o usuarios), **prioridades dinámicas**, que asigna y cambia el sistema operativo, y **prioridades mixtas**, mezcla de las dos anteriores.

Algunos de los algoritmos de planificación de mayor interés son los siguientes:

- **Turno rotatorio (round robin).** A cada uno de los procesos en memoria se le asigna un intervalo de tiempo fijo, o período T, o **quantum** (usualmente de 100 a 10 milisegundos) (Figura 9.5), y se realiza un cambio de contexto de un proceso activo a otro preparado cuando al activo o se le acaba el quantum o se bloquea. La asignación puede hacerse con ayuda de un temporizador (**reloj de tiempo real**), de forma que cuando transcurre un determinado período genera una interrupción al sistema operativo para que éste dé turno a otro proceso.

- **Planificación por prioridad.** El distribuidor da el turno al proceso preparado con mayor prioridad asignada por el propio planificador. Existen varios criterios de asignación de prioridades. A veces, para que el de mayor prioridad no monopolice el uso del procesador, a cada interrupción del reloj de tiempo real (cada 20 ms) se le baja su prioridad (**planificación con realimentación**). Se conmuta al siguiente proceso cuando alguno de los preparados en cola supere la prioridad del proceso activo.
- **Primero el de mayor coeficiente de respuesta o HRRN** (*Highest Response Ratio Next*). Se da prioridad a los procesos que tienen más tiempo de E/S en relación con su tiempo de procesador. Una forma de llevar a la práctica este criterio es dar al proceso  $P_i$  la prioridad  $1/f_i$ , siendo  $f_i$  la fracción de tiempo del último quantum asignado a  $P_i$  que realmente ha consumido el procesador.
- **Primero en llegar, primero en procesar o FCFS** (*First Come First Served*).
- **Primero el de menor tiempo de procesador o SPN** (*Shortest Process Next*).
- **Primero al que reste menos tiempo de procesador o SRT** (*Sortest Remaining Time*).

### EJEMPLO 9.3

Supóngase que los quanta son de 80 ms, y el proceso  $P_7$  sólo consume de él 20 ms (por haberse bloqueado a causa de una operación de E/S). Un planificador HRRN le asignaría una prioridad de  $p_7 = 80/20 = 4$ . Si el proceso  $P_8$  hubiese consumido sólo 10 ms de su quantum de 80 ms, obtendría una prioridad de  $p_8 = 80/10 = 8$ .

### 9.3.5. MODOS DE PROCESAMIENTO

Existen distintos tipos de sistemas operativos con distintos modos de procesamiento, algunos de los cuales se han analizado anteriormente. A continuación, para tener una visión de conjunto, se incluye un resumen de los más relevantes.

**Procesamiento serie de cola (lotes serie).** Al introducir un trabajo en el computador se incluye en una cola, no iniciándose su ejecución hasta ser seleccionado por el sistema operativo. El objetivo del sistema operativo es maximizar el uso del procesador, pasando a segundo plano los tiempos de respuesta obtenidos por los usuarios.

**Multiprogramación.** Se tienen en la memoria principal distintos procesos a los que el sistema operativo va asignando el procesador alternativamente en el tiempo, sin necesidad de que finalice completamente uno para iniciar la ejecución de otro. A veces a estos sistemas se les denomina **sistemas multitarea**, reservándose la denominación de multiprogramación para sistemas multitarea que dispongan de técnicas apropiadas de protección de memoria y de control de concurrencia para permitir el acceso compartido a dispositivos de E/S y archivos.

**Sistema multiusuario.** Sistema de multiprogramación que prevé el uso concurrente de distintos usuarios, con identificación, autenticación y control de los mismos.

**Tiempo compartido.** Sistema de multiprogramación y multiusuario que gestiona procesos interactivos, a través de los cuales el usuario interactúa directamente con el computador casi de forma inmediata, dándole la ilusión de que está trabajando él solo con el computador. El objetivo del sistema operativo es optimizar el tiempo de respuesta obtenido por los usuarios, para lo cual debe utilizar un algoritmo de planificación adecuado (apropiado con turno rotatorio, por ejemplo).

**Tiempo real.** El concepto de **tiempo real** hace referencia a que el sistema debe dar imprescindiblemente la respuesta dentro de un límite de tiempo preestablecido. Este tiempo puede ser pequeño o

grande, dependiendo de la aplicación. Frecuentemente son sistemas de multiprogramación en los que los algoritmos de planificación son del tipo no apropiativo con planificación por prioridad, con los que siempre se ejecuta la tarea de mayor prioridad que no se interrumpe hasta que finalice, a no ser que se genere otra de prioridad mayor.

**Procesamiento de transacciones.** Sistema de tiempo compartido que generalmente implica realizar cambios en una gran base de datos satisfaciendo ciertos requisitos en cuanto a tiempo de respuesta y recuperación de ciertos tipos de fallos. Las entidades bancarias utilizan estos sistemas, ya que deben asegurar que las transacciones no se pierdan, se procesen rápidamente y se minimice el coste de cada transacción.

**Sistema de multiprocesamiento.** Estos sistemas están ideados para actuar con dos o más procesadores trabajando simultáneamente. El sistema operativo distribuye los distintos procesos o hebras entre los diferentes procesadores.

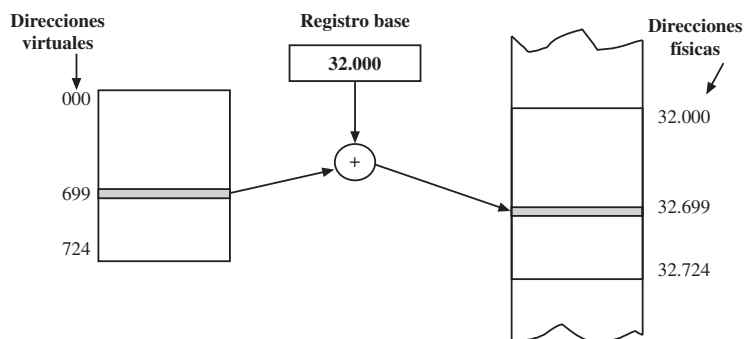
## 9.4. Gestión de la memoria

Un programa máquina es una secuencia de instrucciones en código máquina que en el momento de ejecutarse *encajan* en palabras de memoria que pueden numerarse correlativamente de la 0 a la  $n - 1$  (suponiendo que el programa ocupa  $n$  palabras de memoria). Estas direcciones se denominan **direcciones virtuales** o **direcciones lógicas**,  $dv$ .

El programa anterior, al ser cargado en memoria, ocupará  $n$  determinadas posiciones de la misma. Supongamos que las instrucciones del programa se almacenan consecutivamente; si se cargan a partir de la dirección  $dB$ , el programa quedará ubicado de la dirección  $dB$  a la  $dB + (n - 1)$  (Figura 9.6). A  $dB$  se le suele denominar **dirección base** y a las direcciones  $df$  en que realmente se almacena la instrucción de la **dirección virtual**  $dv$ , se le denomina **dirección física**. Se verifica que:

$$df = dB + dv \quad \text{para todo} \quad 0 \leq dv \leq n - 1$$

A veces el sistema operativo considera al programa dividido en segmentos. Un **segmento** es un grupo lógico de información, tal como un programa (**segmento de código**), una pila asociada al programa (**segmento de pila**) o un conjunto de datos asociados al programa (**segmento de datos**). En consecuencia, el tamaño de cada segmento no está establecido a priori y depende totalmente del programa de que se trate. Un programa ejecutable (listo para ser ejecutado por el procesador, Sección 12.4) es una colección de segmentos.



**Figura 9.6.** Transformación de direcciones virtuales en direcciones físicas.

En un sistema de multiprogramación, el sistema operativo, de acuerdo con los espacios libres de la memoria, asigna una dirección base a cada programa. Las transformaciones entre direcciones virtuales y físicas antiguamente se realizaba por software, pero en la actualidad se suele realizar con ayuda de un registro base (donde se carga previamente la dirección base) que forman parte de una **Unidad de Gestión de Memoria** (MMU, *Memory Management Unit*), que puede encontrarse en el propio chip de procesador o en uno de los chipset de la placa base.

La asignación de memoria para distintos procesos ejecutándose concurrentemente suele hacerse, dependiendo del sistema operativo, de alguna de las siguientes formas:

- Particiones estáticas.
- Particiones dinámicas.
- Segmentación.
- Paginación.
- Memoria virtual.

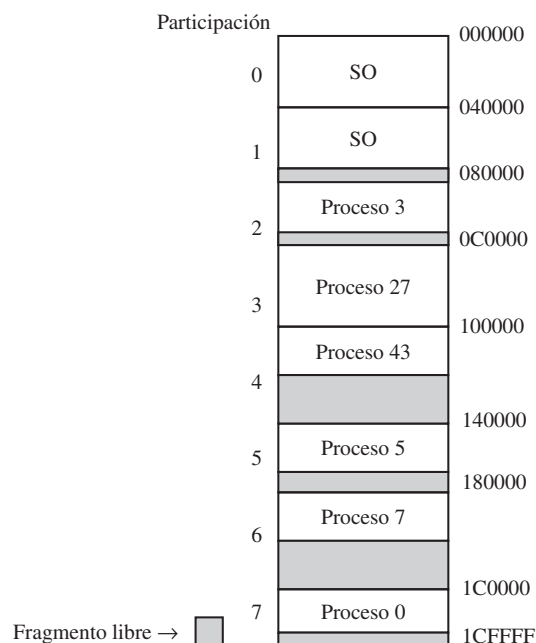
que se analizan en las siguientes secciones.

### 9.4.1. PARTICIONES ESTÁTICAS

La memoria se divide en cierto número de **particiones** o zonas, de igual o de distinto tamaño, cada una de las cuales contendrá un proceso.

#### EJEMPLO 9.4

En la Figura 9.7 se incluye un ejemplo de un sistema con una memoria principal de 2 MB y 8 particiones constantes de 256 KB. Las direcciones base son las de comienzo de cada partición. El tamaño



**Figura 9.7.** Asignación de memoria con particiones estáticas.

de la partición es determinado por el operador o por el sistema operativo. Tamaños usuales son: 8 K, 16 K, 32 K o 64 K. El sistema operativo mantiene una tabla en la que cada fila corresponde a una partición, conteniendo la posición base de la partición, su tamaño (no todas las particiones tienen por qué ser iguales) y el estado de la partición (ocupada o no ocupada). El planificador de trabajos, una vez que una partición está libre, hace que se introduzca el programa de máxima prioridad que haya en la cola de espera y que quepa en ella.

Si el espacio de una partición es  $m$  palabras y el programa ocupa  $n$  posiciones, se verificará siempre que:

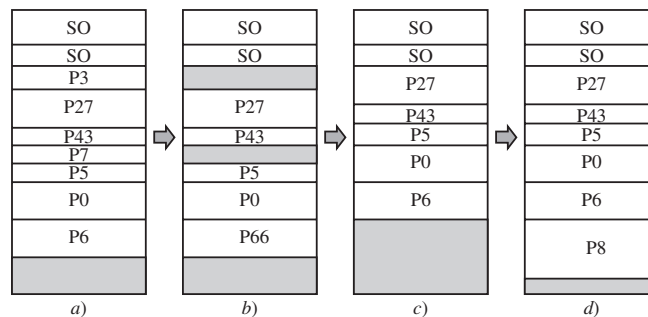
$$n \leq m$$

Cada partición tendrá unas posiciones no utilizadas si  $n < m$ . Se denomina *fragmentación interna* o **fragmentación de una partición** a (véase Ejercicio 9.2):

$$g = m - n$$

### 9.4.2. PARTICIONES DINÁMICAS

Los programas, Figura 9.8a, son introducidos por el sistema operativo inicialmente en posiciones consecutivas de memoria, no existiendo por tanto particiones predefinidas.



**Figura 9.8.** Asignación dinámica: a) los procesos se ubican correlativamente; b) P3 y P7 acaban; c) compactación; d) entra P8.

El sistema operativo puede gestionar el espacio de memoria usando una **tabla de procesos** (Tabla 9.1a), en la que cada línea contiene el número de proceso o identificativo del mismo, el espacio que ocupa, y la dirección base. Existe una tabla complementaria a la anterior con los fragmentos o huecos libres (Tabla 9.1b). El planificador de trabajos periódicamente consulta la tabla, introduciendo

Tabla de procesos		
Proceso	Dirección base	Tamaño
SOa	000000	256 Kp
SOB	040000	128 Kp
P3	060000	180 Kp
P27	08D000	256 Kp
P43	0CD000	128 Kp
P7	0ED000	128 Kp
P5	10D000	128 Kp
P66	12D000	256 Kp

a)

Tabla de fragmentos libres		
Fragmento	Dirección base	Tamaño
0	16D000	588 K

b)

**Tabla 9.1.** Tabla de procesos y fragmentos libres para el ejemplo de la Figura 9.8.

do en memoria los programas que quepan en los fragmentos. Obviamente, al iniciarse una sesión de trabajo se carga el primer programa, dejando un fragmento libre donde se pueden incluir otros programas. Al ir acabando de ejecutarse los procesos (Figura 9.8*b*), el número de fragmentos libres crecerá llegando un momento en que el porcentaje de memoria aprovechado llega a ser muy reducido. El problema puede resolverse haciendo una **compactación** (Figura 9.8*c*). Ésta consiste en reubicar periódicamente los programas de forma que todos los fragmentos queden agrupados quedando así sólo uno grande.

#### EJEMPLO 9.5

En la Figura 9.8*b* se muestra la memoria cuando acaban los procesos P3 y P7; pueden ahora introducirse tres programas: uno de 180 KB o menor, otro de 128 KB o menor y otro en el fragmento inicial libre de 588 KB. Si en la cola de espera sólo hay programas mayores de 588 KB, no entrará ninguno. En la Figura 9.8*c* se muestra cómo queda la memoria después de una compactación; ahora caben uno o varios procesos, con una capacidad total no mayor de 896 KB. En la Figura 9.8*d* entra el programa P8, que ocupa 650 KB.

### 9.4.3. SEGMENTACIÓN

El programa se considera dividido en sus segmentos. La gestión la realiza el sistema operativo, como con las particiones dinámicas, sólo que cada partición corresponde a un segmento en lugar de a un proceso. El sistema operativo mantiene una tabla-mapa de segmentos, indicando la dirección base en memoria de cada uno de ellos y su tamaño. Caso de que una dirección de un segmento sobrepase el tamaño de éste se produce una excepción.

#### EJEMPLO 9.6

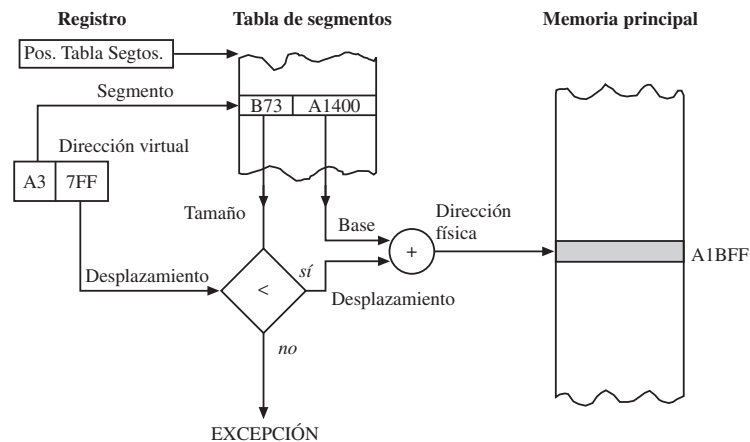
Supóngase un sistema que admite hasta 256 segmentos de 4 KBytes cada uno como máximo. Las direcciones lógicas estarán compuestas de 20 bits (5 cifras hexadecimales): los 8 primeros bits (2 primeras cifras hexadecimales) identificarán el segmento y los 12 bits restantes (3 cifras hexadecimales) identificarán el desplazamiento dentro del segmento. La dirección lógica *A37FF* se transforma en una dirección física según el esquema que muestra la Figura 9.9. Obsérvese que el tamaño del segmento es *B73*. Si por error se hubiese generado en este segmento (*A3*) una dirección virtual mayor o igual a *A3B73*, el sistema operativo generaría una excepción (interrupción interna) de “error de direccionamiento”. A partir de la tabla de segmentos se obtiene la dirección base de memoria donde se almacena el segmento *A3*, que resulta ser *A1400*, con lo que la dirección física correspondiente será:

$$A1400 + 7FF = A1BFF$$

### 9.4.4. PAGINACIÓN

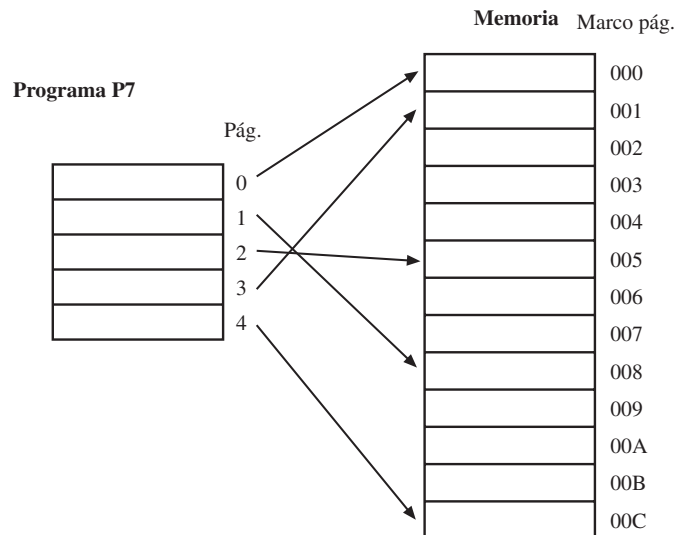
Con este procedimiento la memoria principal se estructura en **marcos de página** (también denominados **bloques de memoria**) de longitud fija. De esta forma, por ejemplo, la memoria de un computador de 64 MBytes podría quedar dividida en 16.384 marcos de página de 4 KBytes cada uno, o en 16 de 4 MBytes. Cada marco de página se identifica con un índice o número correlativo (de 0 a *FFFF*, o 0 a *F*, en los casos anteriores). Asimismo los procesos de los usuarios se dividen en módulos consecutivos de tamaño fijo, denominados **páginas lógicas o virtuales** (o, simplemente, **páginas**). Para un sistema dado, la capacidad de página y marco de página son coincidentes, de forma que cada página se almacena en un marco.





**Figura 9.9.** Esquema de transformación de una dirección virtual en una dirección física en un sistema con segmentación.

El fundamento de la paginación reside en que no es necesario que un proceso se almacene en posiciones consecutivas de memoria. Las páginas se almacenan en marcos de página libres, independientemente de que estén o no contiguos (Figura 9.10).



**Figura 9.10.** Las páginas de un proceso se almacenan en marcos de página no necesariamente consecutivos.

### EJEMPLO 9.7

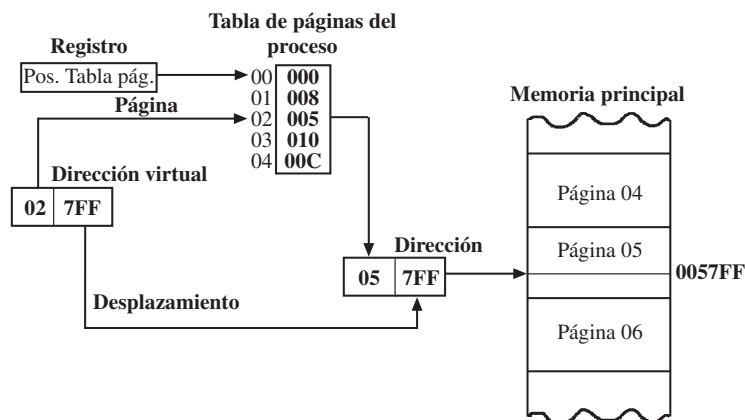
En la Figura 9.10 se muestra como podría almacenarse un proceso de 18 KBytes en un sistema con páginas de 4 KBytes y memoria principal de 16 MBytes. El proceso tendrá 5 páginas (la última de ellas ocupada sólo con 2 KBytes) y la memoria principal 4.096 (en la figura sólo se representan las 13 primeras). Cada marco de página contiene instrucciones consecutivas.

Una dirección virtual (esto es, una dirección del programa) puede considerarse formada por el **número de página** (bits más significativos) y un **desplazamiento** (o posición relativa) dentro de la página (bits menos significativos).

Cada proceso tiene asociado una Tabla de Páginas, que, sencillamente, indica el marco de página donde se encuentra almacenada la página correspondiente. La Tabla de Páginas del Proceso *P7* de la Figura 9.10 se muestra en la Tabla 9.2. La MMU hace la correspondencia entre dirección virtual y física, y puede contener la tabla de páginas o un registro indicando la posición de memoria donde se encuentra (Figura 9.11).

(página)	Marco
0	000
1	008
2	005
3	001
4	00C

**Tabla 9.2.** Tabla de páginas del proceso *P7* de la Figura 9.8.



**Figura 9.11.** Esquema de transformación de direcciones en un sistema con paginación.

En el bloque de control de cada proceso (PCB) se guarda la posición donde se encuentra almacenada su correspondiente tabla de páginas para tenerlo en cuenta en los cambios de contexto. El sistema operativo además mantiene una **tabla de marcos de página**, donde se especifica el proceso y página contenido en cada uno de los marcos y su estado (libre u ocupado). En la Tabla 9.3 se muestra un ejemplo.

### 9.4.5. MEMORIA VIRTUAL

El concepto básico de la memoria virtual se basa en que no es necesario que esté cargada en memoria principal la totalidad de un proceso cuando se encuentra en ejecución. Con la memoria virtual el proceso se mantiene en la memoria externa (disco) y en un momento dado sólo se carga en memoria principal la parte o zona del proceso que esté en ejecución. La ventaja que se obtiene es doble: *a)* el usuario dispone de una *memoria principal aparente mayor* que la memoria física real, no estando limitado el tamaño de sus programas por la capacidad de la memoria, y *b)* al ocupar menos memoria los procesos en ejecución, *podrán cargarse en la memoria más procesos* para ejecutar concurrentemente, mejorando notablemente la eficiencia de la multiprogramación.

La memoria virtual se fundamenta en que las instrucciones de un programa que se ejecutan sucesivamente (en un corto intervalo de tiempo) están en direcciones muy próximas (**principio de localidad temporal**), y en que los programas suelen estar redactados con gran linealidad; es decir, no suelen abundar los saltos entre posiciones de memoria distantes (**principio de localidad espacial**).

(marco de página)	Contenido (proceso, página)	Estado (ocupado/libre)
000	P7,0	0
001	P7,3	0
002	SO,3	0
003	SO,4	0
004	(libre)	1
005	P7,2	0
006	P5,0	0
007	P5,1	0
008	P7,1	0
009	P3,0	0
00A	(libre)	1
00B	(libre)	1
00C	P7,4	0
00D	P3,1	0
.....	...	...
.....	...	...
.....	...	...
FFF	SO,2	0

**Tabla 9.3.** Ejemplo de una tabla de marcos de página (obsérvese la ubicación de las páginas de P7, en concordancia con la Figura 9.10 y Tabla 9.2).

Para implantar la memoria virtual suele utilizarse una de las siguientes técnicas:

- Gestión de memoria por páginas.
- Gestión de memoria segmentada.
- Gestión de memoria segmentada-paginada.

Aquí vamos a considerar principalmente la primera de ellas.

Como se ha comentado anteriormente, en un sistema de memoria virtual se mantiene en disco un archivo con la **imagen del proceso** completo, que se considera troceado en páginas o segmentos, dependiendo de la técnica de gestión que se use. En cambio en la memoria principal únicamente debe estar la página (o segmento, en su caso) que en ese momento deba estar en ejecución, intercambiándose páginas entre disco y memoria principal cuando sea necesario.

La memoria virtual con paginación combina las técnicas de paginación (Sección 9.4.4) e intercambio (Sección 9.3.1) realizado a nivel de página en cuanto a que los bloques de información que se transfieren entre disco y memoria principal, y viceversa, son páginas.

El sistema operativo realiza la gestión de la memoria virtual con demanda de páginas con ayuda de dos tipos de tablas.

**Tabla de páginas del proceso.** El sistema operativo crea y mantiene una de estas tablas por cada proceso. En la Tabla 9.4 se incluye un ejemplo de tabla de páginas de un determinado proceso

(página)	Posición en disco (unidad de ubicación)	Posición en memoria (marco de página)	Ubicación (0: memoria; 1: sólo disco)
0	7ABC	—	1
1	CA73	7	0
2	4BC9	—	1
3	573C	C	0
4	A340	4	0

**Tabla 9.4.** Ejemplo de tabla de páginas de un proceso (P6) en un sistema de memoria virtual.

(Proceso P6). Cada fila corresponde a una de las páginas del proceso, y contiene los siguientes campos:

- *Dirección de disco donde comienza la página*, que puede ser el número de la unidad de ubicación (Sección 13.6) del disco donde se encuentra esa página en el proceso imagen.
- *Marco de memoria principal* en el que está (en su caso) la página.
- *Bit de ubicación*, que indica la *situación de la página*; por ejemplo, 0 si está dentro de la memoria principal, o 1 si está sólo en memoria secundaria, fuera de la memoria principal.

**Tabla de marcos de páginas** (Tabla 9.5). Cada fila de esta tabla corresponde a un marco de página de memoria principal, y contiene lo siguiente:

- (Proceso, página) que está en el marco de página.
- Estado del marco de página (0: ocupado; 1: libre, por ejemplo).
- Modificación; es un bit, *M*, para indicar si desde el instante que se ha cargado la página desde disco por última vez se ha modificado (1) o no (0). Obsérvese que si  $M = 0$  quiere decir que la página almacenada en la memoria principal coincide exactamente con la página almacenada en disco.
- Información adicional: depende del algoritmo de reemplazo que se utilice, como más adelante se verá.

(marco)	Contenido (proceso, página)	Estado (0: ocupado; 1: libre)	Modificación (M) (0: no modificado; 1: modificado)
000	SO,5	0	1
001	SO,6	0	1
002	SO,7	0	1
003	P2,5	0	0
004	P6,04	0	0
005	P2,4	0	0
006	(libre)	1	0
007	P6,1	0	0
008	P8,0	0	0
009	P2,5	0	1
00A	P4,5	0	0
00B	P4,6	0	0
00C	P6,03	0	1
00D	P4,3	0	0
....	....	...	...
....	....	...	...
....	....	...	...
FFF	SO,4	0	0

**Tabla 9.5.** Ejemplo de un trozo de tabla de marcos de página en un sistema de MV.

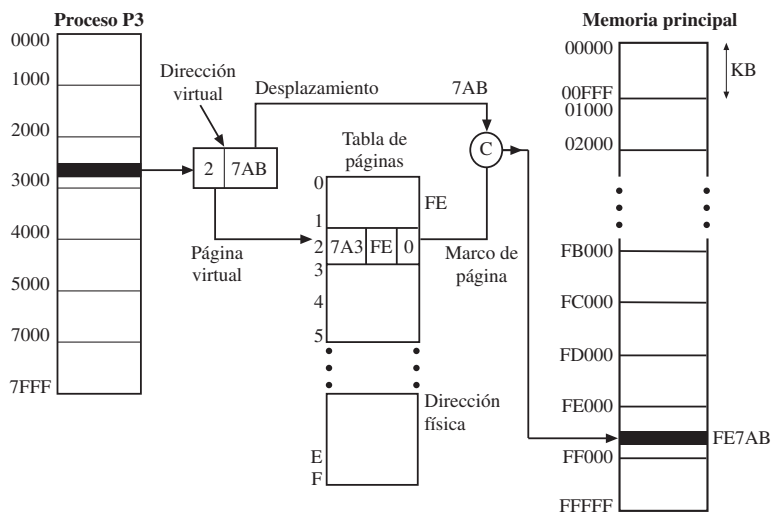
### EJEMPLO 9.8

Supongamos que el proceso *P6* hace referencia a una instrucción o dato de la *Página 0*. El sistema operativo consulta la tabla de *Páginas P6* (Tabla 9.4) y detecta que no está en memoria principal (bit ubicación = 1). En esta situación se dice que se ha producido una excepción de **fallo de página**. El sistema operativo ahora busca en la tabla de marcos de página (Tabla 9.5) en el campo “estado” si hay un marco libre, y en nuestro caso encuentra que el 6 lo está (estado = 1). Entonces con ayuda del

DMA se carga rápidamente la *Página 0* solicitada, que según la tabla de páginas se encuentra en disco a partir de la posición 7ABC. El sistema operativo actualiza las tablas de acuerdo con la nueva situación: en la tabla de páginas de P6 fila 0 hace “ubicación” igual a 0 y posición igual a 6, y en la tabla de marcos de página, en la fila 6 primer campo coloca 6,0; y estado y modificación se hacen igual a cero.

Imaginemos que posteriormente se hace una referencia a la *Página 2*, que tampoco se encuentra en memoria principal. Ahora no hay ningún marco libre, por lo que el sistema operativo (con un **algoritmo de reemplazo de página**) debe eliminar de la memoria principal una de las páginas para introducir la solicitada. Supongamos que el algoritmo indica que debe sustituirse el contenido del marco de página 004 constituido por P6,04. Como su bit modificación es 0, quiere decir que esta página no ha cambiado desde que se introdujo en la memoria principal, por lo que está exactamente igual en disco. El sistema operativo, sin más, carga en ese marco la página solicitada (02), destruyendo la página previa, lo que no causa problema por estar en disco. Si por el contrario hubiese que reemplazar la página 6,03 (en el marco de página 00C), como su bit modificación es 1, antes de cargar la nueva página debe actualizarse la copia en disco de dicha página (que está a partir de la posición 573C).

En la Figura 9.12 se incluye un esquema que indica la forma en que una dirección virtual, 27AB de un proceso P3, se transforma en su dirección física, FE7AB, en un sistema operativo de memoria virtual.



**Figura 9.12.** Esquema de transformación de una dirección virtual en una dirección física en un sistema de memoria virtual.

En relación con las páginas que deben estar en memoria hay que decidir dos cuestiones:

1. *Qué páginas deben cargarse en memoria.* Suele utilizarse una de las dos siguientes alternativas: **paginación por demanda**: una página sólo se lleva a memoria cuando se hace referencia (se *accede*) a una dirección de ella, o **asignación previa**, se cargan en memoria más páginas de las referenciadas, usualmente consecutivas (para así aprovechar la localidad espacial, y para ganar en velocidad: se transmite más rápidamente un bloque de 8 KB que dos de 4 KB).
2. *Qué página debe sustituirse en caso de producirse un fallo de página.* Usualmente se sustituye una página del propio proceso, **sustitución con alcance local**, aunque a veces se realiza una **sustitución con alcance global**; es decir, se analizan todas las páginas de memoria para realizar la sustitución. A continuación se describen algunos de los algoritmos de sustitución más difundidos:
  - **LRU (Least Recently Used)**: se sustituye la página que lleve más tiempo sin usar. Una forma de implementar este algoritmo consiste en asociar un contador, *C*, a cada marco de pá-

gina que se incremente cada vez que se haga referencia a él. Al producirse un fallo de página el sistema operativo elimina la página que esté en el marco con menor valor de  $C$ , es decir, la que se ha referenciado menos recientemente (desde que se han puesto a cero todos los contadores, cosa que puede hacerse, por ejemplo, cada vez que se carga una nueva página).

- **FIFO** (*First In First Out*): se sustituye la que lleve más tiempo en memoria. También puede implementarse como LRU, pero incluyendo en el campo  $C$  una marca temporal (fecha/hora) en el momento de cargar la página en memoria. Se elimina la página cuyo marco tenga la marca de tiempo más antigua
- **NRU** (*Not Recently Used*): se sustituye una página no utilizada recientemente. Puede implementarse añadiendo en la tabla de marcos de página, en lugar del campo  $C$ , otro campo, *referido* ( $R$ ), pero de un solo bit. En cada interrupción de reloj (usualmente cada 20 milisegundos) se hacen cero los bits  $R$  de todos los marcos, y si se hace referencia a un marco se pone su bit  $R$  a uno. De esta forma los marcos de página que en un momento dado tengan 0 no han sido referenciados desde la última interrupción de reloj, y las páginas almacenadas en ellos son candidatas a ser sustituidas caso de producirse un fallo de página. El sistema operativo para hacer la sustitución analiza los bits  $R$  (referido) y  $M$  (modificado), de forma que la prioridad de sustitución, de mayor a menor, es la siguiente:

$$\begin{array}{ll} R = 0, & M = 0 \\ R = 0, & M = 1 \\ R = 1, & M = 0 \\ R = 1, & M = 1 \end{array}$$

Si hay varios marcos de página en iguales condiciones para ser sustituidos, se elige aleatoriamente uno de ellos.

- **Reloj o de segunda oportunidad.** Se sustituye la página más antigua no utilizada recientemente. Para implementar esta técnica se utilizan los bits  $R$  (referido) y un puntero que recorre circularmente los marcos de página (del proceso, si el alcance de sustitución es local) y que se coloca inicialmente en la página que primero se cargó en memoria. En resumen, el comportamiento del algoritmo es como sigue:
  - Cuando se introduce una página, el bit de referencia del marco se pone a cero ( $R = 0$ ).
  - Si se accede a un marco, se hace  $R = 1$ , y no se avanza el puntero.
  - Si se produce un fallo de página:
    - ◆ Se analiza el valor de  $R$  del marco al que apunta el puntero:
      - Si  $R = 0$ , se reemplaza la página y se avanza una posición el puntero.
      - Si  $R = 1$ , se cambia  $R = 0$ , y así sucesivamente se avanza el puntero hasta encontrar un marco con  $R = 0$ , en donde se reemplazó la página.

Puede observarse que este algoritmo es muy similar al FIFO, pero si una página antigua ha sido referenciada ( $R = 1$ ) no se sustituye.

## 9.5. Gestión de entradas/salidas

El sistema operativo contiene módulos software que comunican los programas con los controladores físicos que generan las señales de control para los periféricos. En definitiva, el sistema de gestión de entradas/salidas traduce peticiones generales (abstractas) en operaciones concretas.

Los objetivos fundamentales del software de E/S del sistema operativo son:

- Lograr que los periféricos se utilicen con *eficiencia*.
- Que desde el punto de vista de los usuarios y de las aplicaciones las operaciones de E/S sean lo más generales posibles; es decir, se puedan programar con *independencia del dispositivo*, transparentemente a las características particulares del hardware que se utiliza.

La independencia del dispositivo implica que se pueda asociar a cada uno de ellos un nombre simbólico, siendo las reglas de denominación uniformes para todos.

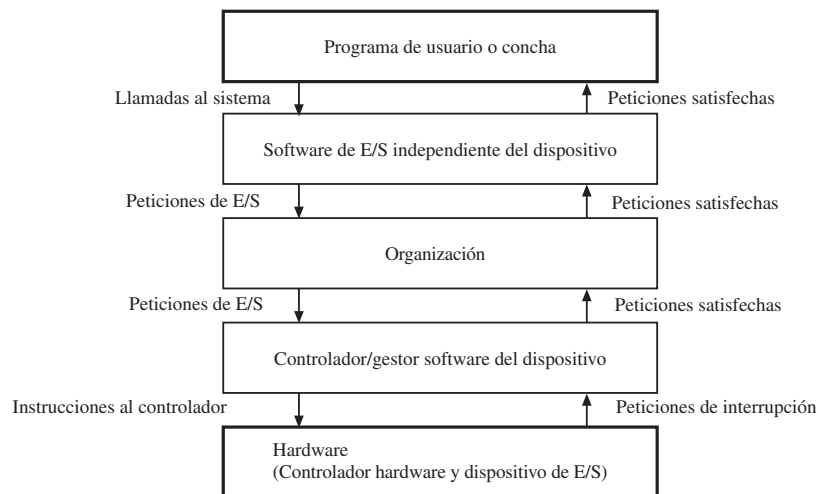
### EJEMPLO 9.9

En UNIX y en Windows NT los dispositivos se referencian como si fuesen archivos (especiales). Así en UNIX para leer de un periférico existe una llamada al sistema que expresada en lenguaje C tiene la siguiente forma:

```
numero = read (descriptorarchivo , zona , numB);
```

con la que se solicita al sistema operativo leer un número determinado de bytes (*numB*) (un **registro lógico**) de un periférico (cuyo nombre simbólico se da por medio de *descriptorarchivo*), y se memorizan en una zona específica de memoria (*zona*) en donde el programa accederá a los datos solicitados. En la variable *numero* se almacena el número de bytes leídos, de forma que si no se pueden leer los *numB* bytes, *numero* contendrá el número real de bytes leídos (si se produce algún error, el sistema operativo hace *numero* = -1). Esta sencilla instrucción permite al usuario hacer abstracción del periférico donde se encuentra el archivo y, dentro de él, el registro.

Se comprende mejor cómo actúa el software de E/S utilizando un modelo conceptual por capas (Figura 9.13). Las tres capas intermedias corresponden al sistema operativo. El nivel inferior del sistema operativo (gestor o controlador software) suele consistir en una biblioteca de rutinas de bajo nivel que trata directamente con el periférico o, más concretamente, con el **controlador (hardware) del dispositivo**, que es quien realmente ejecuta la operación de E/S (en los puertos de entrada/salida, controladores DMA, canales, procesadores de E/S). El nivel superior (software de E/S independiente del dispositivo), en contacto con los procesos de los usuarios, trata con las funciones de E/S de una forma general y desde un punto de vista lógico. De esta forma se logra que los parámetros del hardware afecten lo menos posible al software de E/S.



**Figura 9.13.** Niveles conceptuales seguidos para la gestión eficiente de las E/S.

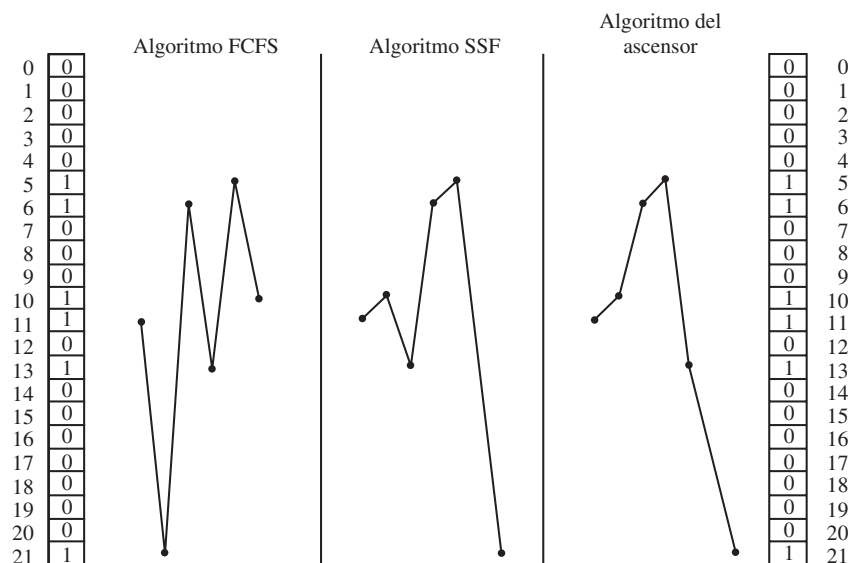
Como ejemplo de tarea de planificación y control del nivel de gestión software del dispositivo hardware, a continuación se analiza la forma de gestionar la búsqueda de un cilindro en una unidad de disco de un servidor de archivos.

Según describimos en la Sección 5.3.1, el tiempo de lectura o escritura de un sector del disco depende de tres factores: tiempo de búsqueda del cilindro, tiempo de latencia (espera al sector) y velocidad de transferencia. El tiempo que se puede optimizar desde el programa gestor del disco es el tiempo de búsqueda del cilindro. El problema es establecer el orden de acceso a los cilindros cuando se acumulan diversas peticiones de acceso. Los algoritmos utilizables son los siguientes:

- **Algoritmo FCFS** (*First-Come First-Served*, primero en llegar, primero en ser atendido). Según van llegando las peticiones de acceso se acumulan en una memoria FIFO de tal manera que la primera solicitud que llega es la primera que se atiende.
- **Algoritmo SSF** (*Sortest Seek First*, el más cercano se atiende primero). Este algoritmo mantiene una tabla de peticiones pendientes, con tantos bits como cilindros (Figura 9.14). Todos los bits inicialmente están a cero, y cuando hay una solicitud de un cilindro se pone el bit correspondiente a 1; es decir, en un momento dado habrá tantos unos como peticiones pendientes. Una vez realizado el acceso a un cilindro se pone a cero su bit. El algoritmo SSF va seleccionando los cilindros en función de la distancia más corta a donde se encuentre posicionada la cabeza.
- **Algoritmo SCAN o del ascensor.** Utiliza también una tabla de peticiones pendientes, y para atenderlas las recorre en secuencia, de abajo a arriba (del 21 a 0) accediendo a los cilindros con peticiones pendientes (es decir, los que en la tabla tienen un 1) y posteriormente recorre la tabla hacia abajo: del cilindro superior que haya tenido petición, en dirección al cilindro inferior con petición, y así sucesivamente.

### EJEMPLO 9.10

Suponiendo, para simplificar, que un disco contiene tan sólo 22 pistas, y que sus cabezas se encuentra inicialmente posicionado en el cilindro 11 y se acumulan sucesivamente las solicitudes de acceso a los siguientes cilindros: 21, 6, 13, 5, 10, los accesos se efectuarían en el siguiente orden (Figura 9.14):



**Figura 9.14.** Representación de los movimientos que se producen en la cabeza de una unidad de disco para atender la secuencia de peticiones del Ejemplo 9.8.



Algoritmo FCFS:

11 → 21 → 6 → 13 → 5 → 10

Algoritmo SSF:

11 → 10 → 13 → 6 → 5 → 21

Algoritmo ascensor:

11 → 10 → 6 → 5 → 13 → 21

## 9.6. Gestión de archivos

El sistema operativo realiza la gestión de archivos coordinando el uso de los dispositivos de memoria masiva, manteniendo información de los nombres simbólicos de todos los archivos almacenados, del lugar físico donde se encuentran ubicados, de los usuarios que pueden acceder a cada uno de ellos, de la localización de las zonas libres, etc.

Un archivo, dependiendo principalmente del uso o capacidad que vaya a tener, puede estructurarse en un soporte (disco o cinta) de distintas formas (Capítulo 13). Desde el punto de vista hardware, para almacenar datos o programas sólo existen direcciones físicas. En un disco toda información se graba o lee físicamente en bloques (clusters o **unidades de ubicación**, Sección 13.6) identificados por (número de unidad)/(superficie)/(pista)/(unidad de ubicación). El sistema operativo posibilita que el usuario no tenga que utilizar direcciones físicas: podemos utilizar un archivo y almacenar o recuperar información sin más que indicar su nombre y efectuando llamadas al sistema o utilizando ciertas órdenes del lenguaje de control.

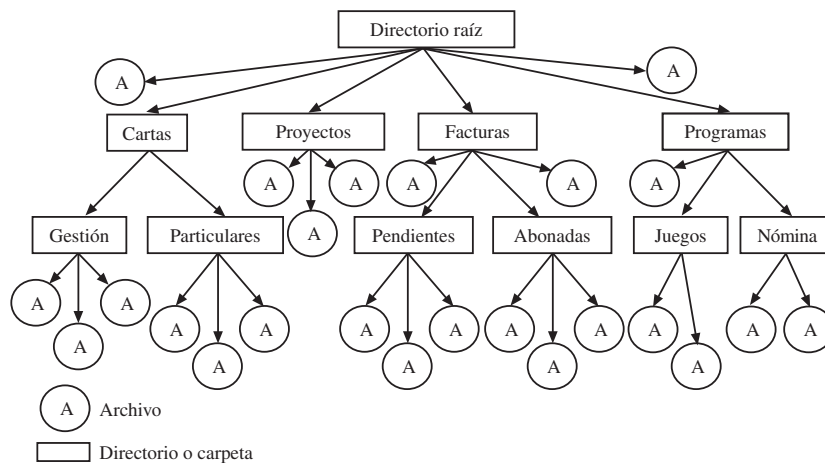
El sistema operativo hace uso de los conceptos de **archivo** y de **carpeta** (o **directorio**) para organizar el almacenamiento de la información. El conjunto de módulos del sistema operativo que se encarga de la gestión de archivos y directorios se suele denominar **sistema de archivos**.

El concepto de **archivo** posibilita aislar al usuario de los problemas físicos de almacenamiento. En efecto, cuando el usuario desee referirse a un conjunto de información del mismo tipo (ya sean datos o programas) como una unidad de almacenamiento, no tiene nada más que crear un archivo dándole el nombre que considere oportuno. Cada archivo tiene asociado su *nombre*, *atributos*, *descriptor de seguridad* (especifica los grupos de personas que pueden acceder al archivo y sus privilegios), *direcciones* donde se encuentran los datos y los *datos*. Los atributos pueden incluir cuestiones tales como fecha y hora de creación, fecha y hora de la última actualización, bits de protección (sólo lectura, o lectura y escritura), contraseña de acceso, número de bytes por registro, capacidad del archivo, etc. Los archivos se almacenan en disco en la forma que se detalla en la Sección 13.6.

El segundo concepto o abstracción que utiliza el sistema operativo para gestionar volúmenes de información es el de **carpeta** o **directorio**. Las carpetas son conjuntos de archivos agrupados siguiendo algún criterio arbitrariamente elegido por el usuario que lo crea: carpeta de cartas, carpeta de facturas, etc. La estructura global del sistema de archivos suele organizarse en forma de árbol en el que los nodos interiores son carpetas o archivos y los nodos exteriores son archivos. De una carpeta pueden depender, Figura 9.15, archivos u otras carpetas.

Una carpeta se puede gestionar con una tabla-índice que contiene un elemento por cada archivo o carpeta dependiente de él. Cada elemento está formado por el nombre del archivo dado por el usuario e información adicional; puede estar constituida por los atributos del archivo y la dirección del bloque donde comienza el archivo. También la información adicional puede ser un puntero a otra estructura con información sobre el archivo.

Cuando se abre un archivo, el sistema operativo utiliza la **ruta** (*path-name*), que contiene una lista de todas las carpetas atravesadas desde la carpeta raíz al archivo en cuestión (al que identifica por su nombre), y extrae a partir de la información del elemento la tabla de direcciones en disco y la ubica en memoria principal. Con ayuda de esta tabla rápidamente pueden realizarse todas las referencias al archivo.



**Figura 9.15.** Ejemplo de organización de archivos y directorios en forma de árbol.

## 9.7. Conclusiones

En este capítulo se han presentado los conceptos básicos sobre sistemas operativos. Después de definir lo que es un sistema operativo (Sección 9.2) y de describir la labor que realiza para facilitar el uso del computador, controlar su funcionamiento e incrementar su eficiencia, se han analizado los distintos servicios que presta para gestionar los distintos recursos. En efecto, en distintos apartados se estudia la gestión del procesador (Sección 9.3), la gestión de la memoria (Sección 9.4), la gestión de entradas/salidas (Sección 9.5) y la gestión de archivos (Sección 9.6). En cierta medida puede afirmarse que el sistema operativo actúa como una interfaz entre la máquina y los usuarios o programas de aplicación.

### Test



**T9.1.** El *software de control* de un sistema operativo está constituido por:

- Los traductores de lenguajes (intérpretes y compiladores), antivirus y el conjunto de paquetes de aplicación general (procesador de textos, hoja electrónica, etc.).
- El sistema operativo, los traductores de lenguajes (intérpretes y compiladores) y gestores de bases de datos.
- El sistema operativo, el intérprete de órdenes y los programas de diagnóstico y mantenimiento.
- Todos los programas del computador menos los programas de los usuarios.

**T9.2.** Los *programas de utilidad* (“utilidades”) de un computador son:

- Los programas para el diagnóstico del funcionamiento del computador.
- Los programas que controlan el funcionamiento del computador.
- Programas adicionales al sistema operativo, tales como antivirus, traductores de lenguajes de alto nivel, compresores de datos, etc.

- Los programas de aplicación tales como procesadores de texto, hoja electrónica, aplicaciones estadísticas, etc.

**T9.3.** Los módulos del sistema operativo se lanzan directamente a ejecución por medio de:

- Señales de control.
- Llamadas al sistema.
- Instrucciones de cualquier lenguaje de programación.
- Programas de biblioteca.

**T9.4.** Un *archivo de impresión* es:

- Un archivo que contiene el programa de control (gestor) de una impresora.
- Un archivo que contiene toda la información necesaria para imprimir un documento por la impresora.
- Un archivo extraordinariamente grande.
- Un archivo que contiene el conjunto de juegos de los tipos de impresión utilizables por una impresora.

**T9.5.** La modalidad de *cola de trabajos* (o lotes) de funcionamiento de un sistema operativo hace referencia a que:

- a) El sistema operativo mantiene una lista (o cola) de todos los trabajos que se van ejecutando a lo largo del tiempo, manteniendo una contabilidad del uso que hacen los usuarios de los distintos recursos.
- b) Los trabajos pendientes de ejecución se almacenan en la memoria masiva, y el monitor del sistema operativo los va lanzando a ejecución según una prioridad.
- c) A la lista de reservas de uso del computador, realizada por los usuarios.
- d) El conjunto de trabajos (en la década de los cincuenta, paquetes de tarjetas perforadas) pendientes de ser introducidos en un computador para su ejecución.

**T9.6.** Un *sistema operativo distribuido* se caracteriza por:

- a) Tener distribuidos sus módulos en distintos computadores.
- b) Utilizar datos de distintos computadores.
- c) Poder transferir archivos de un computador a otro de la red, acceder a un computador remoto para ejecutar programas y acceder a Internet y a páginas web.
- d) Permitir a un usuario ejecutar un mismo programa o distintos programas concurrentemente en diversos computadores, dándole la ilusión de que trabaja en un único computador.

**T9.7.** Un *proceso* es:

- a) Un programa, una vez que ha sido almacenado en el computador.
- b) Un programa que ha iniciado su ejecución.
- c) Un programa (módulo) del sistema operativo que controla el funcionamiento del procesador.
- d) Cualquier programa del sistema operativo que controle a un dispositivo periférico.

**T9.8.** El *planificador de trabajos* (planificador a largo plazo) se encarga de:

- a) Seleccionar los procesos que deben intercambiarse a disco.
- b) Seleccionar el trabajo de espera en cola que debe iniciar su ejecución.
- c) Seleccionar el proceso en estado preparado que debe iniciar o continuar su ejecución.
- d) Asignar las prioridades de uso de los dispositivos de E/S por los distintos procesos.

**T9.9.** El *planificador a medio plazo* se encarga de:

- a) Seleccionar los procesos que deben intercambiarse a disco.
- b) Seleccionar el trabajo de espera en cola que debe iniciar su ejecución.
- c) Seleccionar el proceso en estado preparado que debe iniciar o continuar su ejecución.
- d) Asignar las prioridades de uso de los dispositivos de E/S por los distintos procesos.

**T9.10.** El *planificador a corto plazo* (distribuidor) se encarga de:

- a) Seleccionar los procesos que deben intercambiarse a disco.
- b) Seleccionar el trabajo de espera en cola que debe iniciar su ejecución.
- c) Seleccionar el proceso en estado preparado que debe iniciar o continuar su ejecución.

- d) Asignar las prioridades de uso de los dispositivos de E/S por los distintos procesos.

**T9.11.** Un computador monoprocesador con sistema operativo de monoprogramación:

- a) Sólo puede ser utilizado por un único programador.
- b) Puede ser utilizado por varios programadores, pero hasta que no acaba de ejecutarse un programa no se inicia la ejecución de otro.
- c) En un instante dado se puede ejecutar más de un proceso, siempre que sean del mismo programador.
- d) Se ejecutan concurrentemente en el tiempo distintos procesos del mismo programador, de forma que cuando se bloquea o interrumpe uno de ellos, continúa o se inicia la ejecución de otro.

**T9.12.** Un computador monoprocesador con sistema operativo de multiprogramación:

- a) Sólo puede ser utilizado por un único programador.
- b) Puede ser utilizado por varios programadores, pero hasta que no acaba de ejecutarse un programa no se inicia la ejecución de otro.
- c) En un instante dado se puede ejecutar más de un proceso.
- d) Se ejecutan concurrentemente (alternándose) en el tiempo distintos procesos, de forma que cuando se bloquea o interrumpe uno de ellos, continúa o se inicia la ejecución de otro.

**T9.13.** Un *proceso bloqueado* se caracteriza por:

- a) Estar interrumpida temporalmente su ejecución, sea cual sea el motivo.
- b) Estar interrumpida temporalmente su ejecución, debido a estar realizando una operación de E/S.
- c) Estar en un ciclo indefinido de ejecución, por un fallo del programa. Para salir de este estado es necesario provocar una interrupción del programa.
- d) Estar intercambiado en disco.

**T9.14.** La *multiprogramación* tiene por objeto:

- a) Que varios usuarios programen el mismo computador.
- b) Ejecutar sucesivamente y poco a poco varios trabajos ubicados en la memoria principal.
- c) Que un usuario pueda programar un mismo computador con distintos lenguajes de programación (esto es, con distintos compiladores o intérpretes).
- d) Que el procesador pueda interpretar directamente (sin traducción) distintos lenguajes (C, Pascal, Java, FORTRAN, etc.).

**T9.15.** Un *cambio de contexto* en multiprogramación son las acciones que tiene que realizar el sistema operativo:

- a) Para cambiar la ejecución de un programa de un tipo (de gestión, por ejemplo) a otro programa de otro tipo (científico, por ejemplo).
- b) Para salvar en la memoria principal los contenidos de los registros del procesador asociados a un proceso, y restituir los asociados a otro proceso previamente interrumpido.
- c) Para cambiar o actualizar los periféricos de un computador (por ejemplo, cambiar un CD por un DVD, o una impresora por otra más rápida).

d) Cuando acaba un trabajo de la cola serie y tiene que cargar uno nuevo.

**T9.16.** ¿Cuál es el coeficiente de respuesta de un programa, que consume 3,6 segundos de procesador, y que se introduce en un computador a las 11:36:43,5 (*horas:minutos:segundos*) y se obtienen sus resultados a las 13:15:27,8?:

- a) 1.646.
- b) 0,0006.
- c) Faltan datos para obtener el resultado.
- d) Ninguna de las contestaciones anteriores es correcta.

**T9.17.** Un computador en un minuto ejecuta tres procesos (P1, P2 y P3), cada uno de los cuales consume 9, 8 y 12 segundos, respectivamente, de procesador. Si en ese intervalo de tiempo el SO utiliza el procesador durante 1 segundo en total, la tasa de utilización del procesador resulta ser:

- a) 0,5.
- b) 2,0.
- c) 0,483.
- d) 2,07.

**T9.18.** Un usuario lanza a ejecución un programa que consume 10 segundos de procesador, y tarda medio minuto en producir el resultado. El tiempo de respuesta del computador para ese programa es:

- a) 30 segundos.
- b) 20 segundos.
- c) 40 segundos.
- d) 0,33.

**T9.19.** Un usuario lanza a ejecución un programa que consume 10 segundos de procesador, y tarda medio minuto en producir el resultado. El coeficiente de respuesta del computador para ese programa es:

- a) 0,33.
- b) 3.
- c) 20.
- d) 40.

**T9.20.** Un programa lanzado a ejecución permanece en espera en la cola de entrada durante 15 minutos, y una vez que inicia su ejecución consume 45 segundos en operaciones de entrada/salida, 35 segundos en estado de preparado/listo y 20 segundos de CPU. Un minuto después de finalizado el proceso, el usuario obtiene los resultados. El tiempo de respuesta es:

- a) 1m 40s.
- b) 16m 40s.
- c) 2m 40s.
- d) 17m 40s.

**T9.21.** Un programa lanzado a ejecución permanece en espera en la cola de entrada durante 15 minutos, y una vez que inicia su ejecución consume 45 segundos en operaciones de entrada/salida, 35 segundos en estado de preparado/listo y 20 segundos de CPU. Un minuto después de finalizado el proceso, el usuario obtiene los resultados. El coeficiente de respuesta es:

- a) 5.
- b) 53.
- c) 0,0196.
- d) 0,2.

**T9.22.** En la multiprogramación no apropiativa (*nonpre-emptive*), el sistema operativo:

- a) Puede interrumpir la ejecución de un proceso en cualquier momento, para asignar el procesador a otro proceso.
- b) En ningún caso puede interrumpir la ejecución de un proceso.
- c) Puede interrumpir la ejecución del proceso, para asignar el procesador a otro proceso, pero sólo cuando el primer proceso lo solicita o está en estado de bloqueado.
- d) No permite a ningún proceso apropiarse del uso procesador (monopolizar su uso).

**T9.23.** En la multiprogramación apropiativa (*preemptive*), el sistema operativo:

- a) Puede interrumpir la ejecución de un proceso en cualquier momento para asignar el procesador a otro proceso.
- b) En ningún caso puede interrumpir la ejecución de un proceso.
- c) Puede interrumpir la ejecución del proceso, para asignar el procesador a otro proceso, pero sólo cuando el primer proceso lo solicita o está en estado de bloqueado.
- d) No controla el tiempo de ejecución de los procesos, por lo que alguno de ellos podría apropiarse (monopolizar) el uso del procesador.

**T9.24.** El algoritmo de planificación a corto plazo (distribuidor) que da prioridad al proceso más corto (menor tiempo de CPU) es el:

- a) HRRN.
- b) FCFS.
- c) SPN.
- d) SRT.

**T9.25.** El algoritmo de planificación a corto plazo (distribuidor) que da prioridad al proceso que primero llega es el:

- a) HRRN.
- b) FCFS.
- c) SPN.
- d) SRT.

**T9.26.** El algoritmo de planificación a corto plazo (distribuidor) que da prioridad al proceso que menos tiempo de procesador le queda es el:

- a) HRRN.
- b) FCFS.
- c) SPN.
- d) SRT.

**T9.27.** El algoritmo de planificación a corto plazo (distribuidor) que da prioridad al proceso que en ese instante tiene el mayor coeficiente de respuesta es el:

- a) HRRN.
- b) FCFS.
- c) SPN.
- d) SRT.

**T9.28.** Un sistema operativo de *tiempo real* debe:

- a) Proporcionar a los procesos tiempos de respuesta lo más pequeños posibles.
- b) Proporcionar a los procesos tiempos de respuesta menores de un límite preestablecido.
- c) Actuar con distintos usuarios de forma interactiva, casi inmediata, dando a cada uno de ellos la sensación de que en la máquina está trabajando sólo él.
- d) Permitir realizar accesos o cambios en una gran base de datos con ciertos requisitos en cuanto a tiempo de respuesta y a la recuperación de fallos.

**T9.29.** Un sistema operativo de *tiempo compartido* debe:

- a) Proporcionar a los procesos tiempos de respuesta lo más pequeños posibles.
- b) Proporcionar a los procesos tiempos de respuesta menores de un límite preestablecido.
- c) Actuar con distintos usuarios de forma interactiva, casi inmediata, dando a cada uno de ellos la sensación de que en la máquina está trabajando sólo él.
- d) Permitir realizar accesos o cambios en una gran base de datos con ciertos requisitos en cuanto a tiempo de respuesta y a la recuperación de fallos.

**T9.30.** Un sistema operativo orientado a *transacciones* debe:

- a) Proporcionar a los procesos tiempos de respuesta lo más pequeños posibles.
- b) Proporcionar a los procesos tiempos de respuesta menores de un límite preestablecido.
- c) Actuar con distintos usuarios de forma interactiva, casi inmediata, dando a cada uno de ellos la sensación de que en la máquina está trabajando sólo él.
- d) Permitir realizar accesos o cambios en una gran base de datos con ciertos requisitos en cuanto a tiempo de respuesta y a la recuperación de fallos.

**T9.31.** Suponiendo un computador de longitud de palabra de 32 bits, con una memoria principal de 128 MBytes y que los procesos que ejecuta ocupan por término medio 64 Kpalabras, el número medio de procesos a partir del cual es necesario hacer intercambios (*swapping*) entre memoria principal y disco es:

- a) 2.
- b) 2.048.
- c) 215.
- d) 512.

**T9.32.** Un proceso intercambiado es aquel proceso que:

- a) Está pendiente de una operación de entrada/salida con una unidad de disco.
- b) Se ejecuta con datos de otro proceso.
- c) Está preparado o bloqueado, y ha sido traspasado a disco.
- d) Intercambia datos e instrucciones con otros procesos.

**T9.33.** Una palabra del segmento de código de un proceso se encuentra almacenada en la dirección física de memoria 1FFF 7C3B, teniendo asociado dicho segmento la dirección base 1FFF 6B2A. La dirección virtual de dicha palabra dentro del segmento es:

- a) H'E765.
- b) H'1111.
- c) H'8C3A.
- d) H'9D4B.

**T9.34.** En un determinado sistema la dirección virtual de una instrucción de un programa es A3BC 74CD; suponiendo que el direccionamiento se hace por Bytes, la capacidad máxima prevista para los programas es de:

- a) 4 GBytes.
- b) 1 GByte.
- c) 16 GBytes.
- d) No se dan datos suficientes para obtener el resultado.

**T9.35.** En un determinado sistema operativo que utiliza segmentación, la dirección virtual de una instrucción es A3B C74CD; suponiendo que el direccionamiento se hace por Bytes, la capacidad máxima prevista para los programas es de:

- a) 4 GBytes.
- b) 1 GByte.
- c) 16 GBytes.
- d) No se dan datos suficientes para obtener el resultado.

**T9.36.** En un determinado sistema operativo que utiliza segmentación, la dirección virtual de una instrucción es A3B C74CD; suponiendo que el tamaño máximo de cada segmento es de 16 Mpalabras, el número máximo de segmentos por programa es:

- a) 256.
- b) 16.
- c) 4.096.
- d) No se dan datos suficientes para obtener el resultado.

**T9.37.** Un sistema operativo con memoria segmentada utiliza tablas de segmentos de 16 elementos. Cada elemento contiene la capacidad del segmento (20 bits) y la dirección base correspondiente a ese segmento. Sabiendo que, para un determinado programa, el elemento que ocupa el tercer lugar de la tabla de segmentos contiene el valor 3B7 77B9 0000, la unidad de gestión de memoria para la dirección virtual 21A56A generará:

- a) La dirección física B9A56A.
- b) La dirección física BAA56A.
- c) Una excepción.
- d) DAA56A.

**T9.38.** Un sistema operativo con memoria segmentada utiliza tablas de segmentos de 16 elementos. Cada elemento contiene la capacidad del segmento (20 bits) y la dirección base correspondiente a ese segmento. Sabiendo que, para un determinado programa, el elemento que ocupa el tercer lugar de la tabla de segmentos contiene el valor 3B7 77B9 0000, la uni-



dad de gestión de memoria para la dirección virtual 24A56A generará:

- a) La dirección física BBA56A.
- b) La dirección física FBA56A.
- c) Una excepción.
- d) No se dan datos suficientes para obtener el resultado.

**T9.39.** Suponiendo que las direcciones virtuales de un programa se componen de 32 bits, y que cada programa puede tener como máximo 256 páginas, el tamaño de las páginas es:

- a) 16 Mpalabras.
- b) 64 Kpalabras.
- c) 256 Mpalabras.
- d) No se dan datos suficientes para obtener el resultado.

**T9.40.** Suponiendo que las longitudes de palabra y de las instrucciones virtuales de un programa son de 32 bits, y que cada programa puede tener como máximo 1.024 páginas, el tamaño de las páginas es:

- a) 16 MBytes.
- b) 1 MByte.
- c) 4 MBytes.
- d) No se dan datos suficientes para obtener el resultado.

**T9.41.** El tamaño máximo de las tablas de páginas en un determinado sistema operativo con memoria paginada es de 256 elementos, y en cada uno de estos se incluye el marco de página asociado. Sabiendo que, para un determinado programa, el elemento que ocupa el cuarto lugar de la tabla de páginas contiene el valor FA3 y que una dirección virtual de ese programa es 03C0 6ACD, puede deducirse que la capacidad máxima de memoria principal que admite ese sistema operativo es:

- a) 4 Gpalabras.
- b) 16 Mpalabras.
- c) 64 Gpalabras.
- d) 1 Mpalabra.

**T9.42.** El tamaño máximo de las tablas de páginas en un determinado sistema operativo con memoria paginada es de 256 elementos, y en cada uno de éstos se incluye el marco de página asociado. Sabiendo que, para un determinado programa, el elemento que ocupa el cuarto lugar de la tabla de páginas contiene el valor FA3, la unidad de gestión de memoria para la dirección virtual 03C0 6ACD generará:

- a) La dirección física F A3C0 6ACD.
- b) La dirección física FA30 3C0 6ACD.
- c) Una excepción.
- d) 03C0 7A70.

**T9.43.** La memoria virtual:

- a) Permite ejecutar un programa aunque el computador no tenga memoria principal.
- b) Se ideó para poder ejecutar programas desde disco (sin necesidad de cargar el código máquina en la memoria principal).
- c) Permite ejecutar programas cuyo tamaño sea mayor que la capacidad de la memoria principal.

- d) Hace posible que desde un computador se pueda utilizar (a través de la red) la memoria principal de otro.

**T9.44.** Un sistema con memoria virtual paginada utiliza 32 bits para las direcciones virtuales de los programas, y cada uno de éstos puede llegar a tener hasta  $65.536 = 2^{16}$  páginas. Por otra parte la memoria principal tiene una capacidad de 16 Mpalabras. Si como máximo el sistema operativo mantiene 4 páginas de cada proceso en la memoria principal, ¿cuál es el número mínimo de procesos que se pueden mantener en la memoria principal? Solución:

- a) 64 procesos.
- b) 256 procesos.
- c) 128 procesos.
- d) 512 procesos.

**T9.45.** Un sistema con memoria virtual paginada utiliza 32 bits para las direcciones virtuales de los programas, y cada uno de éstos puede llegar a tener hasta  $65.536 = 2^{16}$  páginas. Por otra parte la memoria principal tiene una capacidad de 16 Mpalabras. Si como máximo el sistema operativo mantiene 2 páginas de cada proceso en la memoria principal, ¿a partir de qué número de procesos tendrá que hacer el sistema operativo intercambios (*swapping*) con disco? Solución:

- a) 64 procesos.
- b) 256 procesos.
- c) 128 procesos.
- d) 512 procesos.

**T9.46.** El objetivos fundamental de la gestión de las E/S por el sistema operativo es que:

- a) A un computador se le pueda conectar cualquier periférico.
- b) Los periféricos conectables se puedan utilizar eficientemente, y de forma lo más transparente para el usuario.
- c) El controlador hardware sea lo menos complejo posible.
- d) El controlador (gestor) software sea lo más sencillo posible.

**T9.47.** Suponiendo que una unidad de disco magnético contiene sólo 32 pistas, sus cabezas se encuentran situadas inicialmente en la 14, y que se encuentran acumuladas las siguientes peticiones de acceso (que han ido llegando en el orden indicado): 30, 12, 17, 0, 23, 15; con el algoritmo FCFS el número total de pistas que tendrían que atravesar radialmente las cabezas de lectura/escritura sería:

- a) 46.
- b) 56.
- c) 87.
- d) Ninguna de las contestaciones anteriores es correcta.

**T9.48.** Suponiendo que una unidad de disco magnético contiene sólo 32 pistas, sus cabezas se encuentran situadas inicialmente en la 14, y que se encuentran acumuladas las siguientes peticiones de acceso (que han ido llegando en el orden indicado): 30, 12, 17, 0, 23, 15; con el algoritmo SSF el número total de pistas que tendrían que atravesar radialmente las cabezas de lectura/escritura sería:

- a) 46.
- b) 56.

- c) 87.  
d) Ninguna de las contestaciones anteriores es correcta.

**T9.49.** Suponiendo que una unidad de disco magnético contiene sólo 32 pistas, sus cabezas se encuentran situadas inicialmente en la 14, y que se encuentran acumuladas las siguientes peticiones de acceso (que han ido llegando en el orden indicado): 30, 12, 17, 0, 23, 15; con el algoritmo del ascensor el número total de pistas que tendrían que atravesar radialmente las cabezas de lectura/escritura sería:

- a) 46.  
b) 56.

- c) 87.  
d) Ninguna de las contestaciones anteriores es correcta.

**T9.50.** En la gestión de entradas/salidas por parte del sistema operativo, el nivel que directamente interactúa (da las instrucciones) al controlador de un periférico es:

- a) El gestor (software) del dispositivo.  
b) El nivel de software independiente del dispositivo.  
c) El nivel de organización física.  
d) El interprete del lenguaje de control del sistema operativo.



## Problemas resueltos

### GESTIÓN DEL PROCESADOR

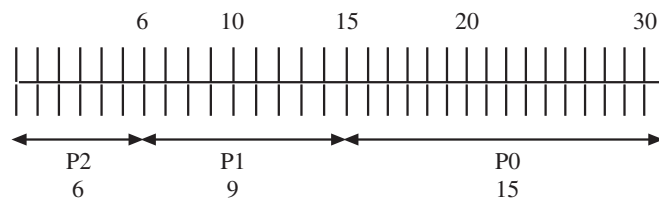
**P9.1.** En un instante dado entran en una cola serie de un sistema de multiprogramación no apropiativa tres programas, con la estimación de tiempos de ejecución que se da a continuación:

P0 (15 minutos)      P1 (9 minutos)      P2 (6 minutos)

Calcular el tiempo medio de respuesta utilizando el algoritmo de planificación SPN (*Shortest Process Next*).

#### SOLUCIÓN

Con el algoritmo SPN se ejecuta en primer lugar el de tiempo de procesamiento menor; es decir, el orden de ejecución será: P2, P1, P0. Un diagrama temporal de la ejecución de los procesos es el siguiente:



Con lo que los tiempos de respuesta serán:

$$R_2 = 6; R_1 = 15; R_0 = 30; \bar{R} = \frac{30 + 15 + 6}{3} \text{ minutos}$$

**P9.2.** En un instante dado entran en una cola serie de un sistema de multiprogramación tres programas, con la estimación de tiempos de ejecución que se da a continuación:

P0 (15 minutos)      P1 (9 minutos)      P2 (10 minutos)      P3 (2 minutos)

¿Con cuál de los siguientes técnicas de planificación obtendría P1 mejor tiempo de respuesta: FCFS (*First Come First Served*), SPN (*Shortest Process Next*) o SRT (*Shortest Remaining Time*)?

#### SOLUCIÓN

- a) FSFS; los procesos se ejecutan en el orden que llegan. Entonces, antes de P1 se ejecuta P0, con lo que el tiempo de respuesta de P1 es:

$$t_{P1} = 15 + 9 = 24 \text{ minutos}$$

- b) SPN; primero se ejecuta el más corto; es decir, el orden de ejecución sería P3, P1, P2 y P0. Con lo que el tiempo de respuesta de P1 sería:

$$t_{P1} = 2 + 9 = 11 \text{ minutos}$$

- c) SRT; en este caso, como los cuatro procesos se han presentado simultáneamente, el orden de ejecución es el mismo que en el caso de SPN, con lo que:

$$t_{P1} = 11 \text{ minutos}$$

En consecuencia, el menor tiempo de respuesta se obtiene con SPN y SRT.

## GESTIÓN DE MEMORIA

**P9.3.** Si la tabla de páginas de un proceso de 15 Kpalabras es la que se indica a continuación, y suponiendo que los bloques de memoria son de 4 Kpalabras, indicar las direcciones físicas de memoria que ocuparán las instrucciones cuyas direcciones virtuales son las siguientes:

- a) 0000.  
b) 3A10.  
c) 2FFF

(página)	Bloque n.º
0	5
1	8
2	7
3	6

### SOLUCIÓN

Las páginas son de 4 Kpalabras  $\times 2^{10}$  palabras/Kpalabra =  $2^{12}$  palabras.

Para direccionar una palabra se necesitan 12 bits  $\rightarrow 12/4 = 3$  cifras HEX.

Dirección virtual = n.º de página + desplazamiento (3 cifras HEX).

- a) 0000  $\rightarrow$  página 0 desplazamiento 000  $\rightarrow$  la página 0 está en el marco 5  $\rightarrow$  dirección física: 5000.  
b) 3A10  $\rightarrow$  página 3 desplazamiento A10  $\rightarrow$  la página 3 está en el marco 6  $\rightarrow$  dirección física: 6A10.  
c) 2FFF  $\rightarrow$  página 2 desplazamiento FFF  $\rightarrow$  la página 2 está en el marco 7  $\rightarrow$  dirección física: 7FFF.

**P9.4.** En un determinado sistema operativo que utiliza segmentación, la dirección virtual de una instrucción es CD7F A7354; suponiendo que el direccionamiento se hace por palabras de 32 bits, ¿cuál es la capacidad máxima prevista para los programas?

### SOLUCIÓN

La dirección virtual tiene 9 cifras HEX =  $9 \times 4 = 36$  bits, luego la capacidad máxima de los programas será:

$$2^{36} = 2^6 \times 2^{30} = 64 \text{ Gpalabras} = 64 \times 4 \text{ GBytes} = 256 \text{ GBytes}$$

**P9.5.** Un sistema operativo con memoria segmentada utiliza tablas de segmentos de 256 elementos. Cada elemento contiene la capacidad del segmento (16 bits) y la dirección base correspondiente. Sabiendo que, para un determinado programa, el elemento que ocupa el decimotercer lugar de la tabla de segmentos contiene el valor A357 B735 0000, ¿qué dirección o información generaría la unidad de gestión de memoria (MMU) para la dirección lógica B735F?

### SOLUCIÓN

1. Según el enunciado del problema cada elemento de la tabla de segmentos contiene 12 cifras hexade-

cimales. Por otra parte, la capacidad del segmento se da con  $16 \text{ bits} = \frac{16}{4} = 4 \text{ cifras HEX}$ , con lo

que las 4 primeras cifras de cada elemento de la tabla de segmentos corresponden a la capacidad del segmento y las 8 restantes a la dirección base.



La dirección lógica B735 corresponde al elemento B = 12 de la tabla, que como empieza a partir del elemento 0, será el decimotercero. Es decir, la dirección base de este segmento es:  $d_B = B735\ 0000$ , con lo que la dirección física será:

$$d_F = B735\ 0000 + 735F = B735\ 735F$$

**P9.6.** Si la tabla de páginas de un determinado proceso es:

0	5
1	A
2	B
3	7

¿Cuál es la dirección física de la dirección virtual 2FAC?

**SOLUCIÓN**

Según la tabla el número de páginas es menor de 16, con lo que la página se dará con una única cifra hexadecimal que será la primera cifra de la dirección virtual. Es decir, la dirección virtual 2FAC se encuentra en la página 2 y tiene un desplazamiento dentro de dicha página de FCA. Según la tabla, la página 2 se encuentra en el marco de página B, con lo que la dirección física será: BFCA

0	5
1	A
2	B
3	7

 $\Rightarrow$  **BFAC**

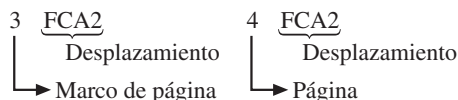
**P9.7.** Si la tabla de páginas de un determinado proceso es:

4
A
D
F
3

¿A qué dirección virtual corresponde la dirección física 3FCA2?

**SOLUCIÓN**

La dirección física 3FCA2 se encuentra, según la tabla, en el marco de página 4, con lo que la dirección virtual será:



**P9.8.** Un sistema con memoria virtual paginada utiliza 32 bits para las direcciones virtuales de los programas, y cada uno de éstos puede llegar a tener hasta 4 Kpáginas. Por otra parte, la memoria principal tiene una capacidad de 1 Gp. Si como máximo el sistema operativo mantiene 2 páginas de cada proceso en la memoria principal, ¿cuál es el número mínimo de procesos a partir del cual es necesario hacer intercambios (*swapping*) entre memoria principal y disco?

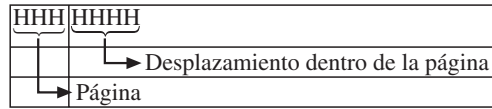
**SOLUCIÓN**

Direcciones virtuales ( $dv$ )  $\rightarrow$  32 bits HHHH HHHH (8 cifras HEX).

Para direccionar la página se necesitan:

$$4\text{ Kpáginas} = 2^2 \times 2^{10} = 2^{12}\text{ páginas} \rightarrow 12\text{ bits} = 3\text{ cifras HEX}$$

Esto nos indica que:



Es decir, la dirección dentro de la página se da con:

$$5 \text{ cifras HEX} = 4 \times 4 = 20 \text{ bits}$$

con lo que la capacidad de una página es:

$$2^{20} \text{ palabras} = 1 \text{ Mp}$$

Por otra parte, como la memoria principal tiene una capacidad de 16 p, en ella caben simultáneamente:

$$\frac{1 \text{ Gp}}{1 \text{ Mp}} = \frac{1.024 \text{ Mp}}{1 \text{ Mp}} = 1.024 \text{ páginas}$$

Con lo que el número de procesos que caben en la Mp es:

$$\frac{1.024 \text{ páginas}}{2 \text{ páginas/proceso}} = 512 \text{ procesos}$$

por tanto, a partir de 512 procesos puede que se tengan que realizar intercambios con disco.

- P9.9.** Suponiendo un computador de longitud de palabra de 32 bits, con una memoria principal de 256 MB y que los procesos que ejecuta ocupan como máximo 64 Kpalabras, ¿cuál es el número mínimo de procesos a partir del cual es necesario hacer intercambios (*swapping*) entre memoria principal y disco?

SOLUCIÓN

$$C_{\text{proceso}} = 64 \text{ Kp} \times \frac{4 \text{ B}}{p} = 256 \text{ KB}$$

$$N_{\text{procesos}} = \frac{256 \text{ MB}}{256 \text{ KB}} = 1.024$$

- P9.10.** Un computador en el que se direccionan palabras de 32 bits es gestionado por un SO con paginación, dándose las direcciones virtuales con 32 bits y pudiendo llegar a tener un proceso hasta 4 Kpáginas. El SO utiliza un disco con un tiempo medio de acceso de 10 ms y con interfaz SCSI-III (16 bits, 40 MB/s). En un momento dado se produce un fallo de página de forma tal que el bit de modificación de la página a desalojar indica que ha sido actualizada. Estimar el tiempo que se tardará en efectuar el cambio de página.

SOLUCIÓN

Si las direcciones virtuales se dan con 32 bits = 8 cifras HEX y un proceso puede llegar a tener hasta 4 Kpáginas =  $2^{12}$  páginas, quiere decir que de los 32 bits los 12 primeros se dedican a identificar la página y los  $32 - 12 = 20$  bits restantes al desplazamiento dentro de la página. Por consiguiente, el tamaño de las páginas será de:

$$C_{\text{página}} = 2^{20} = 1 \text{ Mpalabra} = 1 \text{ Mpalabra} \times \frac{4 \text{ B}}{\text{palabra}} = 4 \text{ MB}$$

suponiendo que toda la información de una página se lee y graba contiguamente en el disco, el tiempo total de transferencia de una página será:

$$t_p = 10 \text{ ms} + \frac{4 \text{ MB}}{4 \text{ MB/s}} = 10 \text{ ms} + 0,1 \text{ s} = 110 \text{ ms}$$

Como en el cambio de página hay que salvar previamente la página que está en la memoria (ya que el bit de modificación es 1), el tiempo total de cambio de página sería:

$$t_{cp} = 2 \times t_p = 110 \text{ ms} \times 2 = 220 \text{ ms}$$

**P9.11.** En un sistema operativo con memoria virtual paginada, para especificar las direcciones se yuxtaponen 4 cifras hexadecimales: 1 para referenciar la página y 3 para indicar el desplazamiento dentro de la página.

- Calcular el tamaño máximo de los programas.
- Suponiendo que la memoria está inicialmente totalmente vacía, que es de 16 Kpalabras y que se referencian sucesivamente las siguientes direcciones:

B7A0, C328, B547, 3466, C5A7, 7F42, AC3A, B567, 973C

¿qué referencias causan fallo de página?

- Suponiendo que la última página referenciada (973C) ha sido almacenada en el marco de página 1, indicar la dirección física a la que se accede.

#### SOLUCIÓN

- Como las direcciones virtuales se forman con 4 cifras HEX, y cada cifra HEX corresponde a 4 bits, cada dirección se formará con  $4 \cdot 4 = 16$  bits; es decir, la capacidad máxima de los programas será:

$$C_{\text{programa}} = 2^{16} \text{ palabras} = 2^6 \cdot 2^{10} \text{ palabras} = 64 \text{ Kpalabras}$$

Por otra parte, como en las direcciones virtuales se utiliza una cifra HEX = 4 bits, para especificar la página, el número de páginas por programa será:  $N_{\text{páginas/programa}} = 2^4 = 16 \text{ páginas/programa}$ .

Como el desplazamiento dentro de cada página se da con 3 cifras HEX =  $3 \cdot 4 = 12$  bits, la capacidad de cada página será:  $C_{\text{página}} = 2^{12} \text{ palabras} = 2^2 \cdot 2^{10} \text{ palabras} = 4 \text{ Kpalabras}$ .

- La memoria principal (MP) tendrá los siguientes marcos de página (mp):

$$N_{\text{mp}} = \frac{16 \text{ Kpalabras/MP}}{4 \text{ Kpalabras/mp}} = 4 \text{ mp/MP}$$

Las sucesivas páginas que se van referenciando podrían introducirse en los 4 marcos de página de la siguiente forma:

Referencia →	B 7A0	C 328	B 547	3 466	C 5A7	7 F42	A C3A	B 567	9 73C
0	B	B	B	B	B	B	A <sup>(1)</sup>	A	A
1		C	C	C	C	C	C	C	9 <sup>(3)</sup>
2				3	3	3	3	B <sup>(2)</sup>	B
3						7	7	7	7
↑ marco de página	fallo de página	fallo de página		fallo de página		fallo de página	fallo de página	fallo de página	fallo de página

<sup>(1)</sup> Se saca la página B (la que lleva más tiempo sin usarse) y se mete la A.

<sup>(2)</sup> Se saca la página 3 (la que lleva más tiempo sin usarse) y se mete la B.

<sup>(3)</sup> Se saca la página C (la que lleva más tiempo sin usarse) y se mete la 9.

- Suponiendo que la página 9 esté en el marco de página 1, la dirección física correspondiente a la dirección virtual 973C será 173C (sustituimos el identificador de página, 9, por el identificador de marco de página, 1). En efecto, el marco de página 1 empieza en la dirección  $1000)_H$ , y como el desplazamiento es 73C, tenemos que la dirección física es:

$$\begin{array}{rcl} \text{Dirección base} & \rightarrow & 1000)_H \\ \text{Desplazamiento} & \rightarrow & 73C)_H \\ \hline \text{Dirección física} & \rightarrow & 173C)_H \end{array}$$

**P9.12.** La memoria de un computador es gestionada por el sistema operativo en la forma de memoria virtual con paginación. En un momento dado se encuentran ejecutándose concurrentemente los procesos que se indican en la siguiente tabla de marcos de página (se indica Proceso/Página):

P0/01, P0/00, P3/02, P2/A6, P4/C5, P1/00, P1/01, P3/00, libre, P2/A5  
P1/02, P3/01, P2/A4, P2/A3, P5/01, P3/03, P1/03, P4/C4, P2/A1, P4/C6

- a) Caso de que sucesivamente se generan las siguientes direcciones virtuales, indique las direcciones físicas correspondientes (*en hexadecimal*).

P2-A6CD73; P3-01A325, P1-0473B5

- b) De acuerdo con los esquemas de direccionamiento utilizados, cuáles son las capacidades máximas posibles de los programas y de la memoria principal.
- c) Cuáles son las direcciones físicas (*en hexadecimal*) correspondientes al último marco de página.

#### SOLUCIÓN

- a) La tabla que da el enunciado del problema nos indica que las páginas se especifican con dos cifras HEX, con lo que la estructura de las direcciones virtuales será la siguiente:

$$\begin{array}{c} \text{página} \downarrow \\ \text{Px- HH HHHH} \\ \text{Proceso} \uparrow \\ \text{desplazamiento} \uparrow \end{array}$$

También de la tabla del enunciado se deducen las páginas que ocupa cada marco de página, según se resume en la siguiente tabla:

Marco de página	Contenido Proceso/pág.
00	P0/01
01	P0/00
02	P3/02
03	P2/A6
04	P4/C5
05	P1/00
06	P1/01
07	P3/00
08	Libre
09	P2/A5
0A	P1/02
0B	P3/01
0C	P2/A4
0D	P2/A3
0E	P5/01
0F	P3/03
10	P1/03
11	P4/C4
12	P2/A1
13	P4/C6

Teniendo en cuenta la estructura de cada dirección virtual, y a partir de la tabla, podemos obtener el marco de página que corresponde a cada dirección virtual, según se muestra en el cuadro de la página siguiente.

- b) Tamaño de los programas:

Como las direcciones virtuales se dan con 6 cifras HEX =  $6 \cdot 4 = 24$  bits, la capacidad máxima de los programas será:

$$C_{\text{programas}}(\text{máxima}) = 2^{24} \text{ Bytes} = 2^4 \cdot 2^{20} \text{ Bytes} = 16 \text{ MBytes}$$

Tamaño de la memoria principal:

Como el desplazamiento se da con 4 HEX =  $4 \cdot 4 = 16$  bits, cada página o marco de página tiene en total:  $2^{16} = 64 \text{ KBytes}$ .

Dirección virtual	Desplazamiento	Página	Consultando la tabla de marcos de página	Marco de página (mp)	Dirección física mp/despla.
<b>P2-A6CD73</b>	CD73	A6		03	03CD73
<b>P3-01A325</b>	A352	01		0B	0BA352
<b>P1-047385</b>	73B5	04		<i>fallo de pág.*</i>	0873B5

\* Como el marco de página 8 está libre, la página 04 del proceso P1 se ubicará allí.

La capacidad que tiene en la actualidad la memoria principal es:

$$C_{\text{memoria}} = 20 \text{ mp} \cdot 64 \text{ KBytes/mp} = 1.280 \text{ KBytes} = 1,25 \text{ MBytes}$$

c) Como el último marco de página es el  $19_{10} = 13_H$ , las direcciones que comprende serán:

$$130000_H \text{ a } 13FFFF_H$$

**P9.13.** En un computador la memoria principal se estructura en páginas. La tabla de páginas de un proceso es la que se indica en la siguiente tabla:

(página)	Marco
0	A6
1	E4
2	F3
3	C8

- Obtener la dirección virtual de un dato que se encuentra en la posición de memoria F37C44.
- Obtener la dirección física de memoria que ocuparía la instrucción cuya dirección virtual es: 35B54.
- Si el tamaño de palabra es de 32 bits, ¿cuál es la capacidad máxima posible de la memoria?

#### SOLUCIÓN

- La dirección física F37C44 está compuesta por el desplazamiento 7C44 dentro del marco de memoria F3, que como indica la tabla de páginas del proceso se corresponde con su página 2, así que la dirección virtual del dato es 27C44.
- En este caso, mirando la tabla de páginas podemos obtener que la página 3 del proceso está alojada en el marco número C8, por lo que la dirección física a la que se desea acceder es C85B54.
- Como los marcos de página se referencian con 1 Byte, podemos concluir que la memoria ha sido dividida en  $2^8 = 256$  marcos de página. También hemos descubierto en los apartados anteriores que los desplazamientos dentro de cada página se realizan mediante cantidades de 16 bits, por lo que cada página debe tener un tamaño de  $2^{16} = 65536$  palabras, así que si cada palabra de memoria es de 32 bits  $= 4 = 2^2$  Bytes, el tamaño de la memoria es de

$$2^8_{\text{marcos}} \times 2^{16}_{\text{palabras/marco}} \times 2^2_{\text{Bytes/palabra}} = 2^{26} \text{ Bytes} = 64 \text{ MBytes}$$

**P9.14.** Un sistema operativo gestiona la memoria principal de un computador organizado en palabras de 32 bits utilizando la técnica de memoria virtual bajo la demanda de páginas de segmentos. El sistema operativo reserva los marcos de página que se indican en la tabla para un determinado proceso PP. El reemplazo de páginas se realiza localmente (es decir, cuando se produce un fallo de página en el proceso PP, la página nueva se introduce en uno de los 4 marcos de página indicados en la tabla). La política de reemplazo de páginas es FIFO y el sistema operativo actúa lo más eficientemente posible para reducir al máximo las transferencias de páginas entre memoria principal y disco.

- Si se hace referencia ordenadamente a las siguientes direcciones de programa:

$$51324837, 742B3248, 742C7CD4, 742C3728$$

determinar las direcciones físicas (en hexadecimal) a las que se accede en cada caso.

Tabla de marcos de página

Marco	Contenido (proceso/segmento/página)	Estado	Instante de carga	Cambio (modificación pág.)
D1	PP,A,A13	0	25	1
D2	PP,3,017	0	25	0
D3	PP,5,137	0	127	0
D4	(libre)	1	0	0

- b) Indicar los tamaños máximos de los programas, de los segmentos y de la memoria principal.  
c) Suponiendo que la unidad de disco donde se encuentra el proceso virtual gira a una velocidad de 7.200 rpm, el tiempo de búsqueda de pista es de 5 ms y que el ancho de banda entre el disco y la memoria es de 20 MB/s, estimar el tiempo necesario para gestionar cada una de las 4 referencias.

## SOLUCIÓN

- a) Del enunciado y de las tablas se deduce que las direcciones virtuales tienen la siguiente estructura:

$$\begin{array}{c}
 \text{página} \downarrow \\
 \text{H HHH HHHH} \\
 \text{segmento} \uparrow \\
 \text{desplazamiento} \uparrow
 \end{array}$$

Consultando la tabla, en la columna de proceso/segmento/página podemos obtener los marcos de página donde se encuentran las direcciones respectivas. Cambiando la identificación proceso/segmento/página, por la del marco de página, obtenemos directamente la dirección física:

Dirección virtual	Segmento	Página	Marco de página	Dirección física
51374837	5	132	D4 <sup>1</sup>	D4 4837
742B3248	7	42B	D2 <sup>2</sup>	D2 3248
742C7CD4	7	42C	D1 <sup>3</sup>	D1 7CD4
742C3728	7	42D	D1 <sup>4</sup>	D1 3728

<sup>1</sup> Fallo de página; como el mp D4 está libre, allí se carga la página 132 del segmento 5.

<sup>2</sup> Fallo de página; como las páginas D1 y D2 son las que llevan más tiempo (instante menor: 25) se elimina una de estas páginas. Como la D2 no se ha modificado, es la que se sustituye, por la 42B del segmento 7.

<sup>3</sup> Fallo de página; ahora se sustituye la página que está en D1 por la página 7/42C.

<sup>4</sup> La página 7/42C está en memoria (en el mp D1) por lo que no hay fallo de página.

- b) Tamaños máximos de:

Programa:

$$\text{direcciones con 8 HEX} = 32 \text{ bits} \Rightarrow C_{\text{programa}} = 2^{32} \text{ palabras} = 4 \cdot 2^{32} \text{ Bytes} = 16 \text{ GBytes.}$$

Segmento:

$$\text{direcciones con 7 HEX} = 28 \text{ bits} \Rightarrow C_{\text{segmento}} = 2^{28} \text{ palabras} = 4 \cdot 2^{28} \text{ Bytes} = 1 \text{ GBytes.}$$

Página:

$$\text{direcciones con 4 HEX} = 16 \text{ bits} \Rightarrow C_{\text{página}} = 2^{16} \text{ palabras} = 4 \cdot 2^{16} \text{ Bytes} = 256 \text{ KBytes.}$$

Memoria principal:

Según la tabla del enunciado, cada marco de página se identifica con 2 cifras HEX = 8 bits  $\Rightarrow$  hay en total  $N_{mp} = 2^8 = 256$  marcos de página. Como en cada marco de página cabe una página:

$$C_{MP} = N_{mp} \cdot C_{página} = 256 \cdot 256 \text{ Kbytes} = 2^8 \cdot 2^{18} = 2^{26} \text{ Bytes} = 64 \text{ Mbytes}$$

- c) Primera referencia: carga de una página en MP:  $t_1 = \frac{256 \text{ KBytes}}{1 \text{ Mbytes/seg.}} = 0,25 \text{ seg.}$

*Segunda referencia:* carga de una página en MP:  $t_2 = 0,25$  seg.

*Tercera referencia:* primero hay que salvar en disco la página PP/A/A13, y después cargar la 7/42C; es decir, hay que hacer la transferencia de 2 páginas (una de MP a disco y otra de disco a MP):  
 $t_3 = 2 \cdot 0,25$  seg. = 0,5 seg.

*Cuarta referencia:* Como no hay fallo de página, no hay que hacer ninguna transferencia con el disco, con lo que:  $t_4 = 0$  seg.

- P9.15.** Un computador con un sistema operativo que gestiona memoria virtual con paginación, en un determinado momento tiene la siguiente tabla de marcos de página.

Proceso	Página
SO	01
SO	A3
P1	B4
P3	C3
P2	F3
P3	05
SO	04
	(Libre)
P1	B1
P5	C7
P1	D1
P4	A3
P4	04
P2	A2
P1	D5
P5	CC

- a) ¿A qué procesos pertenecen y cuáles son las direcciones lógicas correspondientes a las siguientes direcciones físicas: B427; 056B; D118?
- b) ¿Cuál es la capacidad de memoria principal, la capacidad máxima de los programas y el tamaño de página?
- c) Calcular la dirección física a la que se accede cuando se hace referencia a la dirección A312C del proceso P2.

#### SOLUCIÓN

- a) En la tabla de marcos de página (mp) hay 16 elementos (0 a F) de forma que (ver tabla en página siguiente):

En el marco de página 0 se encuentra la página SO/01

En el marco de página 1 se encuentra la página SO/A3

En el marco de página 2 se encuentra la página P1/B4

.....

En el marco de página F se encuentra la página P5/CC

- La dirección física B42F está en el mp:B y, según la tabla, en B se encuentra P4/A3. El desplazamiento es 42F. Luego la dirección física corresponde al proceso P4 y a la dirección lógica A342F.
- 046B se encuentra en el mp:0 que corresponde a la página SO/01; es decir, corresponde al SO y la dirección lógica es: 0146B.
- D118 se encuentra en el mp:D; es decir, corresponde a la página P2/A2, proceso P2, dirección lógica A2118.

- b) Según el apartado anterior y el enunciado del problema:

*Capacidad de MP:* como las direcciones físicas se dan con 4 HEX,  $4 \text{ HEX} \cdot 4 \text{ bits/HEX} = 18 \text{ bits}$ .

$$C_{MP} = 2^{16} = 64 \text{ Kpalabras}$$

Marco de página	Proceso	Página
0	SO	01
1	SO	A3
2	P1	B4
3	P3	C3
4	P2	F3
5	P3	05
6	SO	04
7		(Libre)
8	P1	B1
9	P5	C7
A	P1	D1
B	P4	A3
C	P4	04
D	P2	A2
E	P1	D5
F	P5	CC

*Capacidad del programa:* la página, dentro del programa, se da con 2 cifras HEX (ver tabla de marcos de página); es decir, un programa tiene en total  $2^{2 \times 4} = 2^8 = 256$  páginas/programa. Como el desplazamiento dentro de página se da con 3 cifras HEX =  $4 \times 3 = 12$  bits la capacidad de una página es:  $2^{12} = 4$  Kpalabras, con lo que:

$$C_{\text{programa}} = 256 \text{ páginas/programa} \times 4 \text{ Kpalabras/página} = 2^8 \times 2^{12} = 2^{20} = 1 \text{ Mpalabra}$$

También se podría haber razonado de la siguiente forma: como las direcciones virtuales se dan con 5 cifras HEX =  $5 \times 4 = 20$  bits:

$$C_{\text{programa}} = 2^{20} = 1 \text{ Mpalabra}$$

*Tamaño de página:* según se ha obtenido anteriormente:

$$C_{\text{página}} = 4 \text{ Kpalabras}$$

- c) La dirección virtual A312C del proceso P2 se encuentra en la página A3 de dicho proceso (el desplazamiento es 12C). Observamos en la tabla de marcos de página que la página P2/A3 no se encuentra en la MP, con lo que se produce un fallo de página. Como el marco de página 7 está libre, suponemos que el SO ubicará allí la página P2/A3, con lo que la dirección física será 712C.

**P9.16.** Un sistema operativo gestiona la memoria principal de un computador organizado en palabras de 32 bits: utiliza la técnica de memoria virtual bajo demanda de páginas. Es capaz de gestionar hasta 1 GB de memoria virtual. La memoria principal tiene 4 MB organizados en 16 marcos de página y está inicialmente vacía.

- Indicar en la segunda columna de la tabla cuándo se producen fallos de página al hacer referencia a las direcciones que se indican en la primera columna.

Dirección	Fallo de página	Marco de página	Dirección física
0014569 AF01182 AB01100 0101100 AF01182			
0104400 00AB000 00B0010 0010010			

- Indicar, en la cuarta columna de la tabla, la dirección física a la que se accede en cada referencia, suponiendo que la memoria principal se va llenando ocupando primero el marco de página de menor dirección que se encuentre libre.



## SOLUCIÓN

Cada marco de página ocupa:

$$C_{\text{marco}} = \frac{4 \text{ MB}}{16} = 256 \text{ KB} = \frac{256}{4} \text{ Kpalabras} = 64 \text{ Kpalabras} = 2^{16} \text{ palabras}$$

es decir, para el desplazamiento se utilizan 16 bits = 16/8 cifras HEX = 4 HEX.

Esto quiere decir que las tres primeras cifras HEX de las direcciones virtuales (primera columna de la tabla) corresponden a la página y las cuatro últimas al desplazamiento. De acuerdo con lo anterior podemos escribir la siguiente tabla:

Dirección virtual		Fallo de página	Dirección física	
pág.	desplaza.		m. de p.	desplaza.
001	4569	SI	0	4569
AF0	1182	SI	1	1182
AB0	1100	SI	2	1100
010	1100	SI	3	1100
AF0	1182	NO	1	1182
010	4400	NO	3	4400
00A	B000	SI	4	B000
00B	0010	SI	5	0010
001	0010	NO	0	0010

**P9.17.** Suponga un computador cuya capacidad máxima de memoria es de 16 Mpalabras y que un proceso tiene asignados sólo tres marcos de página: 19, 1A y 1B, e inicialmente están libres. Suponga que se hace referencia a las direcciones virtuales que se indican en la primera columna de la tabla.

- Indicar el tamaño de las páginas y el tamaño máximo de los programas.
- Obtener (y escribir en la tabla) las direcciones físicas a las que se accede, haciendo uso de las siguientes políticas de reemplazo:
  - LRU (usada hace más tiempo).
  - FIFO (primera en entrar primera en salir).
  - Reloj (NRU).

Cadena de referencias	Direcciones físicas (en hexadecimal)		
	LRU	FIFO	NRU
A32C754 B43A65C A3275F2 5414576 7B58798 A3279DC 474678C 7B556FD B43667C A32776A 7B54567 A3267CD			

- c) Indicar (como resultado del ejercicio) cuál de las políticas es más eficiente, y compararlas desde el punto de vista de recursos que necesitan y el tiempo añadido que necesita el sistema operativo para gestionarlas.

## SOLUCIÓN

- a) Capacidad de las páginas:

$$\text{Capacidad de página} = \frac{\text{Capacidad total memoria principal}}{\text{Número de marco de página}} = \frac{C_{MP}}{N_{mp}}$$

$$C_{MP} = 16 \text{ Mpalabras.}$$

Los marcos de página (mp) se identifican con 2 cifras HEX = 8 bits  $\Rightarrow N_{mp} = 2^8$

Luego:

$$C_{página} = \frac{C_{MP}}{N_{mp}} = \frac{16 \text{ Mp}}{2^8} = \frac{2^4 \cdot 2^{20}}{2^8} \text{ palabras} = 64 \text{ Kp}$$

Como  $64 \text{ Kp} = 2^{16} \text{ p}$ , cada posición dentro de la página (esto es, el desplazamiento) se identificará con 16 bits =  $16/4 = 4$  cifras HEX.

*Capacidad de los programas:*

Las direcciones virtuales se dan con 7 HEX =  $7 \cdot 4 = 28$  bits; luego la capacidad máxima de los programas será:

$$C_{programas} = 2^{28} = 2^8 \cdot 2^{20} = 256 \text{ Mp}$$

- b) En las siguientes tablas se muestra el comportamiento de los tres algoritmos.

“mp” significa marco de página.  
 “u” significa marco de página usado.  
 “ ” continúa la página en el marco.  
 “-” marco con una página anterior (de otro proceso).

Referencias pág./despl.	LRU				
	Pág. en mp			Fallo de página	Dirección física mp-despl.
	19	1A	1B		
A32C754	A32	—	—	Sí	19-C754
B43A65C	“	B43	—	Sí	1A-A65C
A3275F2	u	“	—	No	19-4576
5414576	“	“	541	Sí	1B-4576
7B58798	“	7B5	“	Sí	1A-8798
A3279DC	u	“	“	No	19-79DC
474678C	“	“	474	Sí	1B-678C
7B556FD	“	u	“	No	1A-56FD
B43667C	B43	“	“	Sí	19-667C
A32776A	“	“	A32	Sí	1B-776A
7B54567	“	u	“	No	1A-4567
A3267CD	“	“	u	No	1B-67CD

Referencias pág./desp.	FIFO				
	Pág. en mp			Fallo de página	Dirección física mp-despl.
	19	1A	1B		
A32C754	A32	—	—	Sí	19-C754
B43A65C	“	B43	—	Sí	1A-A65C
A3275F2	u	“	—	No	19-4576
5414576	“	“	541	Sí	1B-4576
7B58798	7A5	“	“	Sí	19-8798
A3279DC	“	A32	“	Sí	1A-79DC
474678C	“	“	474	Sí	1B-678C
7B556FD	u	“	“	No	19-56FD
B43667C	B43	“	“	Sí	19-667C
A32776A	“	u	“	No	1A-776A
7B54567	“	7B5	u	Sí	1B-4567
A3267CD	“	“	A32	Sí	1B-67CD

Referencias pág./desp.	NRU							
	Pág. en mp			R(19)	R(1A)	R(1B)	Contador	Dirección física mp-despl.
	19	1A	1B					
A32C754	A32	—	—	0	<b>0</b>	0	1	19-C754
B43A65C	“	B43	—	0	0	<b>0</b>	2	1A-A65C
A3275F2	u	“	—	0	0	<b>0</b>	2	19-4576
5414576	“	“	541	<b>1</b>	0	0	0	1B-4576
7B58798	“	7B5	“	<b>0</b>	0	0	2	1A-8798
A3279DC	u	“	“	<b>1</b>	0	0	2	19-79DC
474678C	“	“	474	1	<b>0</b>	0	0	1B-678C
7B556FD	“	u	“	<b>0</b>	0	0	0	1A-56FD
B43667C	“	“	B43	0	<b>0</b>	0	0	1B-667C
A32776A	u	“	“	1	<b>0</b>	0	0	19-776A
7B54567	“	u	“	1	1	0	0	1A-4567
A3267CD	u	“	“	1	1	0	0	19-67CD

- c) Según se observa en las tablas, con LRU se necesitan efectuar 4 reemplazos de página (sin considerar las 3 cargas de página iniciales). Con FIFO se efectúan 6 reemplazos de página y con NRU tan sólo 3 reemplazos, por lo que resulta más eficiente, en cuanto a menor número de intercambios con disco, el algoritmo NRU.

En cuanto a los recursos:

- LRU necesita un contador por cada marco de página.
- NRU un biestable (1 bit) por marco de página y un contador por proceso.
- FIFO una cola FIFO por proceso.

Por lo que respecta a los algoritmos:

- El más fácil de implementar es el FIFO: se reemplaza el primero de la cola.
- Con LRU es necesario, cada vez que se hace una referencia a una página, actualizar los contadores de los marcos de página, y, cuando hay fallo de página, analizar todos, para, en función de sus valores, determinar el marco de página donde se introducirá la página referenciada.
- El NRU es más sencillo que el LRU ya que, cuando se produce una referencia (sin fallo de página) sólo hay que poner a 1 el bit de referencia del marco de página correspondiente, y, si hay fallo de página, se analizan los bits de los marcos de página sucesivos (cambiándolos de 1 a 0 e incrementando el contador) hasta encontrar un marco de página con  $R = 0$ .

## GESTIÓN DE E/S

**P9.18.** Suponiendo que una unidad de disco magnético contiene sólo 32 pistas, sus cabezas se encuentran situadas inicialmente en la 13 y que se encuentran acumuladas las siguientes peticiones de acceso (que han ido llegando en el orden indicado): 27, 1, 15, 0, 13, 29, 17:

- ¿Cuál es el número total de pistas que tendrían que atravesar radialmente las cabezas de lectura/escritura con el algoritmo SSF (*Shortest Seek First*)?
- Si la densidad radial del disco es de 2.490 pistas/pulgada y la velocidad media de desplazamiento del brazo de la cabeza lectora es de 317,5 cm/s, hacer una estimación del tiempo que se tardaría en realizar los accesos a las pistas indicadas.

## SOLUCIÓN

- El número total de pistas que tendrían que atravesar radialmente las cabezas de lectura/escritura sería:

$$\begin{array}{cccccccc}
 13 & \rightarrow & 15 & \rightarrow & 17 & \rightarrow & 27 & \rightarrow & 29 & \rightarrow & 1 & \rightarrow & 0 \\
 \downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow & & \\
 2 & + & 2 & + & 10 & + & 2 & + & 28 & + & 1 & & = 45
 \end{array}$$

- De acuerdo con el resultado anterior, la longitud que tendría que recorrer el brazo para realizar todos los accesos sería:

$$L = \frac{45 \text{ pistas}}{2.490 \frac{\text{pistas}}{\text{cm}}} = 0,0181'' = 0,0181'' \cdot 2,54 \frac{\text{cm}}{\text{''}} = 0,046 \text{ cm}$$

Con lo que el tiempo en recorrer esa longitud sería:

$$t = \frac{L}{v} = \frac{0,046 \text{ cm}}{317,5 \frac{\text{cm}}{\text{s}}} = 1,446 \cdot 10^{-4} \text{ s} = 0,14 \text{ ms}$$

Conviene hacer notar que únicamente se ha considerado el tiempo de desplazamiento de la cabeza lectora. Para estimar el tiempo total de obtención de la información de las pistas habría que considerar, entre otros factores, los tiempos de espera al sector (latencia) y el ancho de banda (tiempo de transferencia) de la unidad de disco.

## Problemas propuestos



## GESTIÓN DEL PROCESADOR

**P9.19.** Se introduce a las 8:15 horas un trabajo en la cola de espera de un sistema operativo que admite procesamiento por lotes. Una vez iniciada su ejecución a las 9:10 horas, consume 6 segundos de procesador, 7 minutos en E/S y, estando preparado, pierde en esperas 40 segundos. Obtener el coeficiente de respuesta del proceso.

**P9.20.** Suponga que tenemos un computador con multiprogramación en el que cada trabajo tiene características idénticas, y que se utiliza una planificación por turno rotatorio (*round-robin*). En un período de computación,  $t$ , para un trabajo, la mitad del tiempo corresponde a E/S y la otra mitad a actividad del procesador. Cada trabajo se ejecuta durante un total de  $N$  períodos. Se consideran los siguientes parámetros:

- Tiempo de Respuesta (*Turnaround*) = tiempo para finalizar un trabajo.
- Rendimiento (*Throughput*) = número medio de trabajos terminados por período de tiempo,  $t$ .
- Utilización de procesador = porcentaje de tiempo que está activo el procesador (sin estar esperando).
- Coeficiente de respuesta.

Calcule estas cantidades para uno, dos y cuatro trabajos simultáneos, asumiendo que el período  $t$  se distribuye de cada una de las siguientes formas:

- E/S la primera mitad, procesador la segunda mitad.
- E/S el primer y el último cuarto de tiempo, procesador el segundo y el tercer cuarto.

**P9.21.** En una cola serie se encuentran cinco programas, con la estimación de tiempos de ejecución que se indica a continuación:

P0 (10 minutos); P1 (6 minutos); P2 (4 minutos);  
P3 (4 minutos); P4 (4 minutos)

Suponiendo que los programas anteriores entran en el computador en el mismo instante, comprobar que el tiempo medio de retorno para los cinco procesos es menor si se ejecutan primero los de menor duración y después los de mayor duración.

**P9.22.** En SO de multiprogramación utiliza planificación por *turno rotatorio*, y en un momento dado el planificador a corto plazo selecciona para ejecutar los cuatro procesos que se indican en la tabla, ninguno de los cuales tiene operaciones de entrada salida.

Proceso o hilo	<i>n</i> (miles de ciclos de procesador)	Prioridad (1 la mayor)	Orden en que han llegado
A	8	2	1.º
B	5	4	2.º
C	2	2	3.º
D	7	3	4.º

Calcular la tasa de utilización del procesador y el coeficiente de respuesta medio, suponiendo las siguientes planificaciones:

- No apropiativa, con derecho preferencial.
- Turno rotatorio con quantum de 2 ciclos.
- FIFO.
- Discutir los resultados.

Segmento	Base	Límite	Atributos de acceso
Código (CS)	07AC 321B	05210	—
Datos (DS)	547C B312	B7325	—
Pila (SS)	C432 7C30	00FFF	—

- Cuál es el tamaño del programa.
- Cuál es el tamaño máximo posible de la memoria principal.
- Cuál es la dirección física correspondiente al dato de la dirección virtual A7325.
- Cuál es la dirección física correspondiente a la instrucción de la dirección virtual 7355.

**P9.25.** Un computador tiene una capacidad de memoria principal de 640 KB y está organizada en bloques de 32 KB y gestionada por el sistema operativo por paginación. Suponiendo que en un momento dado se ejecutan concurrentemente los programas siguientes:

S.O. (64 KB), P1 (130 KB), P2 (82 KB), P3 (96 KB)

- ¿Qué espacio de memoria se encuentra desaprovechado?
- Escriba la tabla de procesos supuesto que las tablas-mapas de páginas están en el bloque 1 de la memoria principal.
- Indique cómo calcularía las direcciones físicas que ocupa cada bloque de memoria.

Proceso	C (tiempo CPU) (ms)
A	30
B	50
C	20
D	40

Suponiendo que los quantum de tiempo son de  $T = 10$  ms, y que el sistema operativo consume un quantum para realizar los cambios de contexto, obtener el coeficiente de respuesta para el proceso A.

**P9.23.** Se tienen los siguientes trabajos a ejecutar con un procesador cuya frecuencia de reloj es de  $F = 500$  MHz.

## GESTIÓN DE MEMORIA

**P9.24.** Suponga un sistema operativo que actúa con segmentación pura, y en el que cada proceso se organiza en como máximo 4 segmentos. Imagine la siguiente tabla de segmentos.

- Suponiendo que las tablas-mapas de páginas de la figura corresponden a las de los procesos P1 y P2, obtenga las direcciones físicas correspondientes a las siguientes direcciones relativas de cada proceso (el primer dígito corresponde al número de página):

P1  $\rightarrow$  32507

P2  $\rightarrow$  10100

Tablas-mapas de páginas	
Proceso P1	Proceso P2
3	4
7	A
8	B
9	
C	

**P9.26.** Un sistema de multiprogramación con memoria virtual paginada utiliza 32 bits para las direcciones virtuales de los programas, y cada uno de éstos puede llegar a tener hasta 4 Kpáginas. Se sabe que el sistema operativo mantiene como máximo 8 páginas de cada proceso en la memoria principal, y se observa que el mínimo número de procesos con el que llegan a producirse intercambios (*swapping*) con disco es de 512. ¿Cuál es la capacidad de memoria principal del sistema?

**P9.27.** Un computador organizado en palabras de 32 bits dispone de un procesador con un tiempo de ciclo de 50 ns y de una unidad de disco que gira a 10.800 rpm, y realiza las transferencias de información a 40 MBytes/s. Por otra parte el sistema operativo es de memoria virtual, con páginas de 4 Kpalabras. Estadísticamente se observa lo siguiente:

- a) El 99,5 por 100 de las instrucciones que se ejecutan acceden a la página en curso de ejecución.
- b) De las instrucciones que acceden a otra página, el 75 por 100 de las veces lo hacen a páginas ya en memoria.
- c) La página a reemplazar, tras un fallo de página, el 1,5 por 100 de las veces ha sido modificada.

Para el sistema anteriormente descrito, realizar una estimación del incremento medio del tiempo de ciclo, a consecuencia de los transvases de páginas asociados a la memoria virtual.

**P9.28.** En un sistema operativo con memoria virtual paginada, suponiendo que la memoria está inicialmente totalmente vacía, que es de 16 Kpalabras y que se referencian sucesivamente las siguientes direcciones:

B7A0, C328, B547, 3466, C5A7, 7F42, AC3A, B567, 973C

- a) ¿Qué referencias causan fallo de página (suponga que la técnica de reemplazo es de tipo FIFO; es decir, se sustituye la página que lleve más tiempo en memoria)?
- b) Indicar la dirección física a la que se accede en la última referencia (973C).
- c) Indicar el tiempo que tarda en transferirse una página suponiendo que el ancho de banda entre disco y memoria es de 16 KB/seg.

**P9.29.** En un sistema operativo con memoria virtual paginada, para especificar las direcciones virtuales se yuxtaponen 4 cifras hexadecimales: 1 para referenciar la página y 3 para indicar el desplazamiento dentro de la página. Las palabras de memoria son de 32 bits. Suponiendo que la memoria está inicialmente totalmente vacía, que es de 16 Kpalabras y que se referencian sucesivamente las siguientes direcciones:

B7A0 (l), C328 (l), B547 (e), 3466 (l),  
C5A7 (e), 7F42 (l), AC3A (l), B567 (l), 973C (l)

(La letra entre paréntesis indica si la referencia es de lectura, l, o de escritura, e.)

- a) Indicar la dirección física a que corresponde la última dirección virtual (973C) de acuerdo con las distintas políticas de reemplazo estudiadas (FIFO, LRU y NRU).
- b) Estimar, para las 9 referencias indicadas y en el caso FIFO, los tiempo de transferencia con el disco debidos a fallo de página, suponiendo que el ancho de banda entre disco y memoria es de 16 KB/seg.

Referencia →	B7A0	C328	B547	3466	C5A7	7F42	AC3A	B567	973C
Tiempo →									

## GESTIÓN DE E/S

**P9.30.** Suponiendo que una unidad de disco magnético contiene sólo 32 pistas, sus cabezas se encuentran situadas inicialmente en la 14 y que se encuentran acumuladas las siguientes peticiones de acceso (que han ido llegando en el

orden indicado): 30, 12, 17, 0, 23, 15, obtener el orden en que se realizarían los distintos accesos con:

- a) El algoritmo SSF.
- b) El algoritmo FCFS.
- c) El algoritmo del ascensor.



# Tipos y estructuras de datos

Debido a la muy diversa naturaleza (textos, números, imágenes, etc., símbolos, en general) de la información que deben procesar los programas de un computador, aquélla debería planificarse y estructurarse de acuerdo con unos formatos y reglas preestablecidos, que se denominan **tipos de datos** o **estructuras de datos**. Un tipo o una estructura de datos determina cómo se codifican los datos y las operaciones que se pueden realizar con ellos. En este capítulo se van a analizar conceptos básicos de los tipos y estructuras de datos más utilizados. Entre los primeros se encuentran los tipos de datos entero, real, lógico, carácter, enumerado y subrango; y entre los segundos las estructuras arrays, cadenas, listas, árboles y grafos. Es importante indicar que no todos los lenguajes de programación disponen de la posibilidad de operar con los mismos tipos de datos, y por otra parte este capítulo no pretende cubrir todos los existentes.

El hardware del computador es capaz de operar directamente con los tipos de datos más sencillos (enteros, lógicos y carácter) de forma que los repertorios de instrucciones máquina suelen contener instrucciones específicas para ello. Si el computador dispone de un procesador aritmético (FPU) también es capaz de operar directamente con datos reales. Con frecuencia, a las estructuras más complejas (listas lineales, pilas, colas, árboles y grafos) se les denomina **tipos abstractos de datos**, ya que están definidas haciendo abstracción de las operaciones que realmente realiza el hardware y se idearon en función de las necesidades de las aplicaciones. El programador puede utilizar estas estructuras quedándole oculto cómo realmente se implementan, cuestión que es problema del compilador o del procesador de lenguaje que se esté utilizando.

Siempre que en un programa se utilice un dato o una estructura, debe estar claramente definido de qué tipo es con objeto de que se pueda almacenar y tratar correctamente. Esta definición está a veces implícita en las reglas del lenguaje y otras veces el programador debe explicitarlo por medio de lo que se denomina una **declaración**. Un lenguaje en el que todas las variables deben ser declaradas por el programador suele denominarse **lenguaje de programación fuertemente tipificado**, tal es el caso, por ejemplo, del lenguaje Ada.

Es interesante comentar que no se han llegado a introducir en los lenguajes de programación tipos o estructuras de datos de gran relevancia en la actualidad, como son imágenes, audio, vídeo e hipertexto<sup>1</sup>.

---

<sup>1</sup> El lenguaje Java sí incluye algunas herramientas para operar con estos tipos de datos.



## 10.1. Tipos de datos

Un **dato** es cualquier objeto que pueda tratar un computador. Los datos son las **constantes** que se definen dentro de los programas, las **variables** que éstos utilizan, la información externa a dichos programas, etc. Recordemos que físicamente sólo podemos acceder a los datos contenidos en la memoria de un computador indicando su dirección; el concepto de variable nos permite utilizar esos datos por medio de nombres simbólicos. El valor de una variable en un momento dado viene determinado por el contenido de las posiciones de memoria asociadas a ella.

Externamente al computador utilizamos valores, que podemos denominar **magnitudes**. Por lo general, cualquier magnitud no se puede representar exactamente dentro del computador como dato, de forma que los dominios de las magnitudes y de los datos que las representan son distintos. Se denomina **tipo de datos** al conjunto de transformaciones entre magnitudes y datos que las representan así como a las operaciones y funciones internas y externas definidas con dichos datos. En consecuencia, un tipo de datos debe establecer la forma de codificación de los datos y las operaciones que pueden realizarse.

En esta sección se analizan los tipos de datos más sencillos y básicos, algunos de los cuales ya se estudiaron en el Capítulo 2; no obstante, los volvemos a incluir aquí para dar una visión de los mismos desde el punto de vista del programador. Otros tipos de datos, como enumerado y subrango, son implementados por software incluyendo en los traductores de lenguajes las funciones de biblioteca adecuadas para su uso por el programador.

### 10.1.1. ENTEROS

Los **datos de tipo entero** se utilizan para representar números enteros. Si queremos representar los números enteros con un conjunto finito y fijo para cada dato, se puede almacenar el número en binario con un número,  $n$ , fijo de bits. Con  $n$  bits se pueden representar enteros aproximadamente<sup>2</sup> en el rango  $2^{n-1}$  y  $-2^{n-1}$ , siendo  $2^n$  la **cardinalidad del tipo** (número de datos exactamente representables dentro del tipo). Es decir, aumentando o disminuyendo el número  $n$  de bits se podrán representar más o menos enteros, necesitando más o menos memoria, respectivamente. Habitualmente:  $n = 8, 16, 32$  o  $64$ . Según vimos en el Capítulo 2, con este tipo de datos puede ser que al realizar una operación con dos números enteros dé como resultado un valor no representable, produciéndose por consiguiente un error. A este tipo de error se le suele denominar **desbordamiento** (*overflow*), y el programa puede o no avisar al usuario cuándo se produce.

### 10.1.2. REALES

Los **datos de tipo real** se utilizan para representar números reales. Se suele hacer expresando el número como producto de una mantisa por una base elevada a un exponente, es decir:

$$N = M \cdot B^E$$

donde  $N$  es el número a representar,  $M$  es la mantisa,  $E$  es el exponente y  $B$  es la base del exponente. Los números de bits que se utilicen para la mantisa y el exponente van a determinar la precisión y el número de reales que se puedan representar. Habitualmente se utiliza la normalización IEEE754 para representar internamente este tipo de datos (Sección 2.4.2).

---

<sup>2</sup> Según vimos en el Capítulo 2, los datos enteros se pueden representar internamente según distintos formatos: sin signo, signo y magnitud, complemento a uno, complemento a dos, sesgado, etc. La cardinalidad y el rango concreto de representación dependen del formato (Sección 2.4).

Al igual que los datos de tipo entero, los reales utilizan en su representación un número,  $n$ , limitado de bits, que suele ser 32 (**simple precisión**), 64 (**doble precisión**) o 128 (**cuádruple precisión**). Al ser  $n$  un número finito, surgen los siguientes problemas:

- Se pueden producir errores si el resultado de alguna operación es excesivamente pequeño (**agotamiento** o *underflow*) o grande (**desbordamiento**).
- Es necesario extremar las precauciones con los errores de redondeo, especialmente cuando directa o indirectamente aparecen cancelaciones (ver Problema 10.2).
- En la suma y producto no siempre se cumplen las propiedades asociativa y distributiva (ver Problema 10.20).

Más detalles pueden verse en la Sección 2.4.2.

### 10.1.3. LÓGICOS

Los **datos de tipo lógico** se utilizan para representar valores lógicos o booleanos. Los datos lógicos son aquellos que sólo pueden tomar dos valores (*verdadero* o *falso*, 1 o 0). Con este tipo de datos se pueden definir **operaciones lógicas**. Los **operadores lógicos** básicos son: AND (Y), OR (O) y NOT (NO). También se utilizan mucho las operaciones NAND (NO-Y), NOR (NO-O) y XOR (NO-exclusivo). En la Tabla 10.1 se muestran dichas operaciones básicas. Más detalles sobre estas operaciones pueden verse en la Sección 3.1.1.

a	b	a AND b	a OR b	NOT a	a NAND b	a NOR b	a XOR b
0	0	0	0	1	1	1	0
0	1	0	1	1	1	0	1
1	0	0	1	0	1	0	1
1	1	1	1	0	0	0	0

Tabla 10.1. Operaciones lógicas.

El programador puede definir y utilizar directamente variables lógicas y constantes lógicas (*verdadero* o *falso*; *TRUE* o *FALSE*); pero también se generan como consecuencia de la evaluación de **operadores de relación**, entre los que se encuentran:

- |                     |   |                                 |
|---------------------|---|---------------------------------|
| • Igual             | =   | Ejemplos: $A = 32.5$ ; $7 = 42$ |
| • Distinto          | $\neq$                                      | Ejemplos: $X - Y \neq C - 27.2$ |
| • Menor que         | <   | Ejemplo: "Pepe" < "Juan"        |
| • Mayor que         | >   | Ejemplo: $7 > 5.6$              |
| • Menor o igual que | $\leq$ (a veces se representa como $\leq$ ) | Ejemplo: $6 \leq 6$             |
| • Mayor o igual que | $\geq$ (a veces se representa como $\geq$ ) | Ejemplo: $C \geq D$             |

Obviamente el resultado de evaluar un operador de relación es *verdadero* o *falso*, dependiendo de que se cumpla o no, respectivamente, la relación.

**EJEMPLO 10.1**

¿Cuál es el resultado de la evaluación de las expresiones de relación que se indican en la primera columna de la tabla que se da a continuación?

$7 = 42$	<i>Falso.</i>
$X - Y \neq C - 27.2$	<i>Verdad</i> si $C$ es distinto de $X - Y + 27.2$ , <i>falso</i> en caso contrario.
"Pepe" < "Juan"	<i>Falso</i> , ya que los caracteres se consideran en orden alfabético creciente.
$X = \text{"Carlos"}$	<i>Verdad</i> si la variable $X$ contiene la palabra <i>Carlos</i> , <i>falso</i> en caso contrario.
"Antonio" < $X$	<i>Verdad</i> si $X$ contiene una cadena de 7 caracteres que alfabéticamente va después de <i>Antonio</i> .
$6 \leq 6$	<i>Verdad.</i>
$C \geq 48$	<i>Verdad</i> siempre que $C$ almacene un valor mayor o igual a 48, <i>falso</i> en caso contrario.

Los procesadores de lenguajes asocian a cada variable lógica una palabra de memoria o un registro del procesador, y dentro de este almacenamiento el valor de la variable queda almacenado en un único bit, habitualmente el de la posición menos significativa. Así, en  $C$  un valor lógico se representa como un entero con valor 1 (cierto) o 0 (falso).

**10.1.4. CARACTERES**

Los **datos de tipo carácter** representan elementos del conjunto de los caracteres. A cada carácter se le asocia un código y ese código es el que se almacena. El conjunto de códigos que se utilice para cada carácter depende del lenguaje de programación o de la aplicación que se esté utilizando. Los códigos más habituales son el EBCDIC (8 bits), ASCII (7 u 8 bits) y Unicode (16 bits). Por ejemplo, utilizando Unicode, al teclear el carácter "A" se almacena H'0041; o al teclear "a" se almacena H'0061. En el Apéndice 2 se incluyen tablas de los códigos anteriores, y más detalles pueden verse en la Sección 2.1.

Cuando se utiliza un carácter como un valor constante se escribe entre comillas. Así, "C" representa al carácter  $C$ ; de esta forma se puede distinguir entre nombres de variables y caracteres.

**EJEMPLO 10.2**

Indicar los valores que almacenarán las variables que se indican al evaluarse la siguiente sucesión de expresiones:

```

r = 4
Pepe = "r";
Jose = r;
z = "4"

```

**SOLUCIÓN**

$r = 4$	$r$ almacena el valor 4.
Pepe = "r"	La variable <i>Pepe</i> almacenará el carácter "r".
Jose = r	La variable <i>Jose</i> almacenará el valor que en ese momento tenga la variable $r$ ; es decir, 4.
$z = \text{"4"}$	La variable $z$ almacenará el código del carácter 4 (H'0034, en Unicode), no el valor numérico 4 (H'0004, suponiendo que se almacena en $n = 16$ bits).

Como cada símbolo del código tiene asociado una combinación de bits, ésta puede interpretarse como un valor numérico, de forma que a cada símbolo se le puede hacer corresponder un número en-

tero, y al revés, a cada número entero de un rango determinado<sup>3</sup> se le puede hacer corresponder un código de un carácter. De esta forma el orden alfabético de los caracteres coincide con el de su código, considerado como número entero. La mayoría de lenguajes de programación disponen de funciones para pasar el símbolo a su código numérico decimal, y viceversa. Estas funciones, por ejemplo, podría ser *ORD(carácter)* y *CHR(entero)*, respectivamente.

### EJEMPLO 10.3

Obtener el código numérico decimal de los siguientes caracteres Unicode (o ASCII-Latín 1): “6”; “b”; “Z” y “+”.

#### SOLUCIÓN

```
ORD("6") = 54
ORD("b") = 98
ORD("Z") = 90
ORD("+") = 43
```

### EJEMPLO 10.4

Obtener el carácter Unicode al que corresponden los siguientes valores decimales:

#### SOLUCIÓN

```
CHR(50) = "2"
CHR(114) = "r"
CHR(47) = "/"
CHR(75) = "K"
```

## 10.1.5. ENUMERADOS

Los **datos de tipo enumerado** pueden ser de muchos tipos. Se definen por el programador especificando de forma ordenada el conjunto finito de elementos o valores que integran el tipo. Los enumerados se almacenan como valores enteros.

Por ejemplo, se puede definir el tipo enumerado *mes* de la siguiente forma:

```
mes = {enero, febrero, marzo, abril, mayo, junio, julio, agosto, septiembre,
       octubre, noviembre, diciembre}
```

Los lenguajes de programación que aceptan este tipo de datos incluyen funciones especiales para operar con ellos; así, el Pascal contiene las funciones predecesor (*PRED*) y sucesor (*SUCC*), que proporcionan el anterior o siguiente del dato enumerado seleccionado, según se pone de manifiesto en el siguiente ejemplo.

---

<sup>3</sup> En ASCII-Latín 1 de 0 a 256, y en Unicode de 0 a 65.535.

**EJEMPLO 10.5**

Indicar los valores que toma la variable *V* como resultado de evaluar las expresiones que se indican para datos del tipo meses.

**SOLUCIÓN**

<i>V</i> = SUCC(noviembre);	<i>V</i> toma el valor diciembre.
<i>V</i> = SUCC(diciembre);	<i>V</i> toma el valor enero.
<i>V</i> = PRED(mayo);	<i>V</i> toma el valor abril.
<i>V</i> = PRED(enero);	<i>V</i> toma el valor diciembre.
<i>V</i> = PRED(SUCC(enero))	<i>V</i> toma el valor ENERO.

**10.1.6. SUBRANGO**

Los datos de tipo **subrango** se forman a partir de datos de tipo entero, carácter o enumerado. Son subconjuntos de los datos del tipo original, pudiéndose realizar las mismas operaciones que con éstos.

**EJEMPLO 10.6**

TYPE;	<i>definición de variables o tipos en Pascal</i>
vocales = "a", "e", "i", "o", "u";	<i>vocales es un subrango del tipo carácter</i>
verano = junio..agosto;	<i>verano es un subrango del tipo mes</i>
dígitos_pares = 0, 2, 4, 6, 8;	<i>dígitos_pares es un subrango del tipo entero</i>

**10.2. Estructuras de datos**

Una **estructura de datos** se forma a partir de una serie de variables relacionadas, a las que se puede acceder individualmente o como un conjunto. Es decir, una estructura de datos representa un conjunto de datos relacionados entre sí. Las estructuras se construyen a partir de otros tipos de datos.

Hay dos tipos de estructuras de datos:

- **Homogéneas:** todos los datos elementales que la forman son del mismo tipo.
- **Heterogéneas:** los datos elementales que la forman son de distintos tipos.

Hay estructuras en las que se conoce a priori su tamaño y por tanto al definir las en el programa se puede reservar la memoria necesaria para almacenarlas, éstas se denominan **estructuras estáticas**. Sin embargo, hay otras en las que no se conoce el espacio que van a ocupar, en este caso se denominan **estructuras dinámicas**.

**10.2.1. ARRAYS**

Un **array** o **matriz**<sup>4</sup> es un conjunto de elementos del mismo tipo y de tamaño fijo. Cada dato se encuentra en una posición fija que se conoce gracias a uno o varios índices. Para cada combinación de índices de un array determinado existe uno y sólo un elemento. Un array es una estructura estática,

<sup>4</sup> A veces a estas estructuras se las denomina también **formaciones** o **arreglos** o **tablas**.

siendo necesario indicar cuándo se declara el tamaño máximo de cada una de sus dimensiones. El tamaño de un array será el producto del número de elementos en cada dimensión. Para determinar lo que ocupa un array en memoria hay que multiplicar el tamaño del array por lo que ocupa cada elemento, que, a su vez, depende del tipo de dato de los elementos del array.

En Pascal el programador puede establecer arbitrariamente los nombres de los índices; así la declaración:

```
Temperatura: array [2..10, 5:30] of float
```

define un array en el que  $i = 2, 3, \dots, 10$  y  $j = 5, 6, 7, \dots, 30$ .

Por otra parte, en C, C++, C# y Java los índices comienzan por 0; así, el elemento **Tabla**[7][5] se refiere al dato ubicado en la fila octava, columna sexta.

### EJEMPLO 10.7

Obtener la ocupación de memoria (en bytes) de un array definido como **B**(1:100,1:100), suponiendo que es de tipo entero, con  $n = 32$  bits.

#### SOLUCIÓN

El tamaño del array será:

$$t_{VM} = 100 \cdot 100 = 1.000 \text{ elementos}$$

Como cada elemento ocupa 32 bits = 4 Bytes, la ocupación total de memoria del array será:

$$C_{VM} = 10.000 \cdot 4 = 40.000 \text{ Bytes} \approx 40 \text{ KBytes}$$

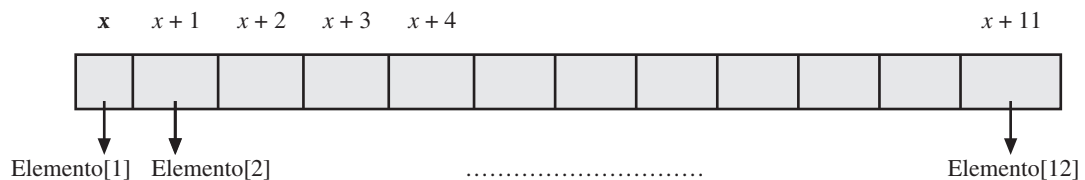
El concepto de array es similar al de matriz en matemáticas, aunque estas últimas estructuras lo son sólo de números, mientras que en los arrays los elementos pueden ser caracteres o pertenecer a otro tipo de dato cualquiera, numérico o no. Recuérdese que en una matriz, al igual que en un array, podemos hacer referencia a sus elementos por medio de los índices que indican su posición; así, el elemento  $a_{ij}$  especifica al dato que se encuentra en la fila  $i$ , columna  $j$ . Un array de una sola dimensión (**array lineal**) es similar a un vector: se puede hacer referencia a cada elemento indicando su posición relativa dentro del mismo. Usualmente se hace referencia a los elementos de los arrays indicando su nombre y a continuación los índices entre paréntesis y separados por comas (caso de FORTRAN y Pascal) o con su nombre y a continuación los índices, cada uno de ellos entre corchetes (caso de C, C++, C# o Java). Por ejemplo, para hacer referencia al elemento de la matriz **A**, cuyos índices son:  $i = 4, j = 4, k = 3$  se utilizaría **A**(4,4,3) en Pascal y **A**[4][4][3] en los otros lenguajes citados.

		Columnas		
		1	2	3
Filas	1	Julio	Luis	Pepe
	2	a	b	c
	3	María	Ana	Laura
	4	11	12	13

Elemento  
(3,2)

**Figura 10.1.** Ejemplo de array de dos dimensiones.

Si se quieren procesar una serie de, por ejemplo, 12 datos de la misma naturaleza, en vez de tener que crear 12 variables se puede crear un array de dimensión 12 que almacene esos valores. El acceso a un elemento del array se hace a través de su índice. En memoria los elementos de un array se almacenan en posiciones consecutivas; de forma que si  $x$  es la dirección del primer elemento, el cuarto elemento estará en la posición  $x + 3$ .



**Figura 10.2.** Representación de un array de dimensión 12.

Dentro de un programa se puede hacer referencia a un array completo, o a cada elemento que puede ser utilizado como una variable, según se indica en el siguiente ejemplo.

### EJEMPLO 10.8

Suponiendo que un array lineal,  $E(6)$ , contiene los siguientes elementos:

$$E = \{32, 48, 24, 2, 14, 1\}$$

indicar el valor de la variable  $Z$  como resultado de evaluar las siguientes expresiones:

$$I = 4;$$

$$J = 6;$$

$$Z = \frac{45 - E(I) + 5 \cdot E(2) + E(J)}{2}$$

### SOLUCIÓN

Después de evaluar  $I$  y  $J$ , tenemos que:

$$Z = \frac{45 - E(I) + 5 \cdot E(2) + E(J)}{2} = \frac{45 - E(4) + 5 \cdot E(2) + E(6)}{2} = \frac{45 - 2 + 5 \cdot 48 + 1}{2} = 142$$

Es decir,  $Z$  contendrá el valor 142.

Al declarar un array, el traductor del lenguaje hace que se reserve una zona específica de memoria para él, y se almacena, o bien “por filas” o bien “por columnas”. Si se hace **almacenamiento por filas**, se memoriza primero la primera fila, después la segunda, y así sucesivamente, siempre en posiciones consecutivas. En el caso de **almacenamiento por columnas**, primero se almacena la primera columna, después la segunda, etc. La expresión que obtiene la dirección de memoria en función de los índices de un elemento se denomina **dirección polinomial**.

### 10.2.2. CADENAS DE CARACTERES

Una **cadena** es una estructura formada por un conjunto sucesivo de caracteres. Por ejemplo, una cadena puede ser una palabra.

Se pueden hacer diversas operaciones con cadenas, como por ejemplo:

- **Concatenar**, es decir, se puede formar una cadena nueva a partir de dos existentes. Por ejemplo, dadas las cadenas “123” y “456”, se pueden concatenar dando lugar a “123456”.
- **Extraer** una subcadena de una cadena. Por ejemplo, dada la cadena “123456” se puede extraer la subcadena “135”.
- **Comparar** dos cadenas según su orden. Como se indicó en la Sección 10.1.4, a cada carácter se le puede asociar el valor numérico correspondiente a su código, de esta forma es posible decir que un carácter es mayor o menor que otro. Así la *B* es mayor que la *A*, la *B* que la *C*, etc. Al comparar dos cadenas se considera mayor aquella en el que el primer carácter en que difieren es mayor. De esta forma se pueden, por ejemplo, ordenar alfabéticamente cadenas de caracteres.
- **Obtener la longitud** de una cadena, es decir, el número de caracteres que contiene.
- **Identificar subcadena**.

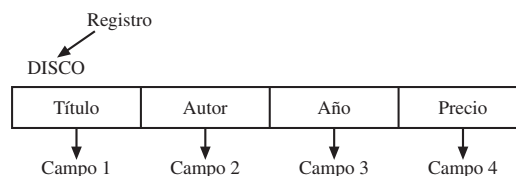
### 10.2.3. REGISTROS

Un registro está formado por un conjunto de elementos relacionados, que pueden ser de distinto tipo, relativos a un mismo ente, y al que se le da un único nombre. Un registro está formado por campos que tienen un determinado orden y se identifica cada uno de ellos con un nombre. Para definir un registro hay que especificar el nombre y tipo de cada campo que lo constituye. Todos los campos de un registro están relacionados, es decir, contienen información sobre un mismo tema o ente. Por ejemplo, un registro para una discoteca que tuviese 4 campos: título, autor, año y precio (Figura 10.3), en Pascal se podría declarar así:

```
Disco: record
    Titulo: packed array [1..8] of char;
    Nombre: packed array[1..10] of char;
    Autor: packed array[1..8] of char;
    Año: integer;
    Precio: real
end
```

El mismo registro en lenguaje C se declara de la siguiente forma:

```
struct
{char Titulo [10];
  char Nombre [10];
  char Autor [8];
  int Año;
  float Precio
} Disco
```



**Figura 10.3.** Ejemplo de registro formado por 4 campos.



La diferencia con un array es que en éste todos los elementos tienen que ser del mismo tipo, mientras que los campos de un registro suelen ser de distinto tipo; de hecho, a veces se denominan a los registros **arrays heterogéneos**.

Desde los lenguajes de programación se puede acceder a un registro completo (por su nombre) o a campos individualmente. Así, *Disco1.Precio* hace referencia al campo Precio del registro *Disco1*.

### EJEMPLO 10.9

Sumar el precio de tres los 3 registros de discos siguientes: *Disco1*, *Disco2* y *Disco3*.

#### SOLUCIÓN

```
precio_total = Disco1.Precio + Disco2.Precio + Disco3.Precio
```

Puede observarse que los campos de un registro pueden tratarse como cualquier variable.

## 10.2.4. LISTAS

Una **lista** es una estructura de datos formada por un conjunto de datos de un mismo tipo ordenados de acuerdo con una secuencia lineal. Existen dos tipos básicos de listas: contiguas y con enlaces. Los elementos de una lista se suelen denominar **nodos**.

Los elementos de una **lista contigua** se almacenan en una zona de memoria de forma consecutiva. Cada elemento constituye un subbloque y se almacenan en orden, uno detrás de otro. Si la dirección inicial de almacenamiento de la lista es *db* y cada subbloque ocupa *cbb* palabras de memoria, la **dirección polinomial** del elemento *i*-ésimo de la lista será:

$$de_i = db + cbb \cdot (i - 1)$$

donde *de<sub>i</sub>* representa la dirección a partir de donde se encuentra almacenado el elemento *i*.

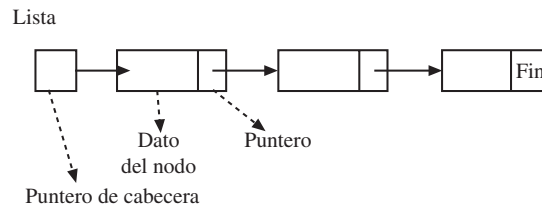
Un problema importante que se presenta con las listas contiguas es que si queremos insertar (o eliminar) un elemento en una determinada posición *i*, hay que desplazar todos los elementos restantes que estén a continuación de *i*. Piénsese, por ejemplo, en una lista de alumnos de una clase que está ordenada alfabéticamente; al insertar un nuevo alumno habría que desplazar *cbb* posiciones de memoria los nombres de todos los alumnos que, dentro del orden alfabético, estuviesen después del nuevo.

Con las **listas enlazadas** se resuelve el problema de inserción y borrado de las listas contiguas, obteniéndose una estructura dinámica. En efecto, cada elemento de la lista tiene dos partes, los datos del usuario y un identificativo, enlace o **puntero** para localizar el elemento siguiente de la lista (Figura 10.4). El nombre de una lista tiene asociado el **puntero de cabecera**, para acceder a su primer elemento; a partir de él es posible localizar cualquier elemento de la lista, por lectura sucesiva de los punteros de los elementos anteriores. Obviamente, el último nodo no tiene puntero, ya que no tiene ningún nodo sucesor; en su lugar se suele colocar un patrón de bits que no correspondan a ninguna dirección, y que sea identificable por las funciones que operen con la lista, usualmente se pone todo a ceros; nosotros utilizaremos la palabra *FIN* como identificador.

Las operaciones que se pueden llevar a cabo con una lista son las siguientes:

- **Acceder al primer elemento**, lo que, como se ha comentado anteriormente, se puede acceder directamente.
- **Acceder a un nodo**. Según hemos indicado para acceder a un nodo es necesario acceder previamente a los que le anteceden. A través de accesos a los nodos se realizan las operaciones de **búsqueda** y de **recuperación** (lectura) de los datos de un nodo.

- **Insertar o añadir un nodo** al principio, al final o en medio de la lista. Esta operación se hace almacenando en algún sitio de memoria libre en nodo nuevo, y copiando el puntero del nodo anterior como puntero del nodo nuevo a su siguiente, y definiendo como puntero del nodo anterior la dirección nodo nuevo (Problema 10.11).
- **Eliminar un nodo.** Sencillamente se cambia el puntero del nodo anterior por el del nodo eliminado (Problema 10.11).
- **Recorrido** de la lista. Consiste en atravesar todos los nodos de la lista, por ejemplo para imprimirlos.

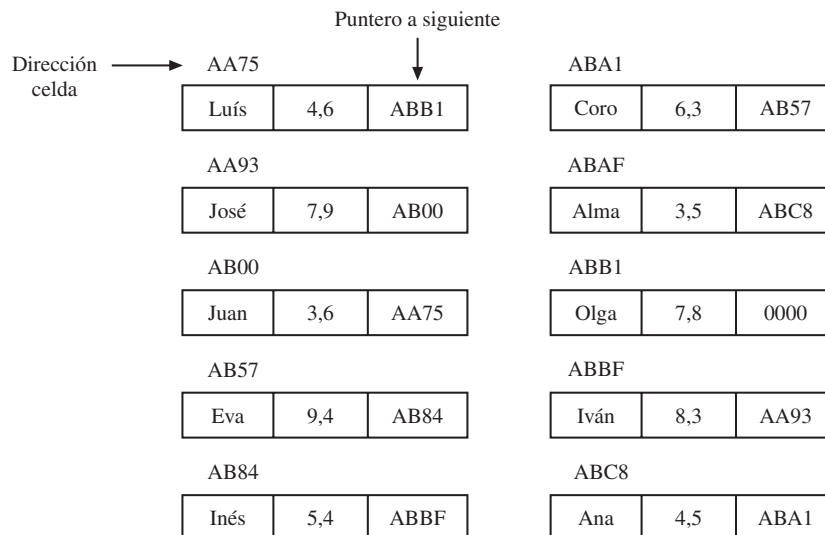


**Figura 10.4.** Lista con tres nodos.

Puede concluirse que una lista con enlaces, además de ser una estructura dinámica, no necesita ser almacenada en posiciones contiguas de memoria, pero por el contrario el acceso a cada uno de los nodos es más complejo y lento.

### EJEMPLO 10.10

Suponiendo que se dispone de la lista enlazada de la Figura 10.5, cuyo puntero de cabecera es H'ABAF, indicar los pasos que habría que realizar para acceder a la nota de Eva.



**Figura 10.5.** Ejemplo de lista enlazada.

### SOLUCIÓN

- Se accede al nodo de cabecera, dirección ABAF, se lee, y como Eva  $\neq$  Alma, se pasa al nodo siguiente, dirección ABC8.

- Se accede al nodo de dirección ABC8, se lee, y como Eva  $\neq$  Ana, se pasa al nodo siguiente, dirección ABA1.
- Se accede al nodo de dirección ABA1, se lee, y como Eva  $\neq$  Coro, se pasa al nodo siguiente, dirección AB57.
- Se accede al nodo de dirección AB57, se lee, y como Eva = Eva, la nota es 9,4.

### EJEMPLO 10.11

Acerca de la lista enlazada de la Figura 10.5, cuyo puntero de cabecera es H'ABAF, determinar:

- El último nodo.
- Su ocupación en memoria, sabiendo que el campo nombre ocupa 8 Bytes, la nota 4 Bytes y el puntero 2 Bytes.

### SOLUCIÓN

- El último nodo es el que tiene como puntero la dirección H'0000; es decir, Olga.
- Cada nodo ocupa:

$$C_{nodo} = 8 + 4 + 2 = 14 \text{ Bytes/nodo}$$

Como hay un total de 10 nodos, la capacidad total de memoria para almacenar la lista será:

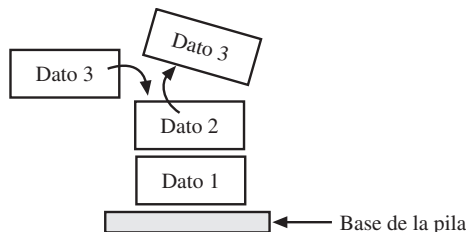
$$C_{TOTAL} = 10 \cdot 14 = 140 \text{ Bytes}$$

Las estructuras analizadas en esta sección se denominan **listas generales**, y se caracterizan porque se pueden insertar y borrar directamente elementos que estén en cualquier parte de ellas. En las dos secciones siguientes se estudian otros dos tipos de listas que tienen restringido su acceso a uno de sus extremos.

## 10.2.5. PILAS

Una **pila** es una lista en la que sólo se pueden hacer inserciones y eliminaciones en el extremo inicial de la estructura. De esta forma, el último elemento insertado tiene que ser el primer elemento en salir, y de ahí que a estas estructuras se las denomine también listas **LIFO** (*last-in, first-out*). El primer elemento de la lista se denomina **cabecera**. El proceso de **insertar** (o “apilar”) es una operación de colocar un nuevo elemento como cabecera, introduciéndose en un nivel inferior la cabecera anterior, y la de **extracción** (o “desapilar”) supone la lectura y eliminación de la cabecera actual, pasando el elemento inmediato del nivel inferior a cabecera (Figura 10.6).

En la Sección 4.2.2 se indicó cómo pueden implementarse las pilas.



**Figura 10.6.** Pila “el dato 3 es el último en entrar y el primero en salir”.

Las operaciones básicas que se pueden hacer con una pila son:

- Insertar elementos.
- Extraer elementos.
- Consultar si está vacía o no.

### EJEMPLO 10.12

Describir las operaciones que se realizan en el interior de una pila cuando se realizan las siguientes acciones:

- Insertar Ana.
- Insertar José.
- Insertar Eva.
- Extraer.
- Insertar Juan.

### SOLUCIÓN

En la Figura 10.7 pueden verse las acciones que tienen lugar.

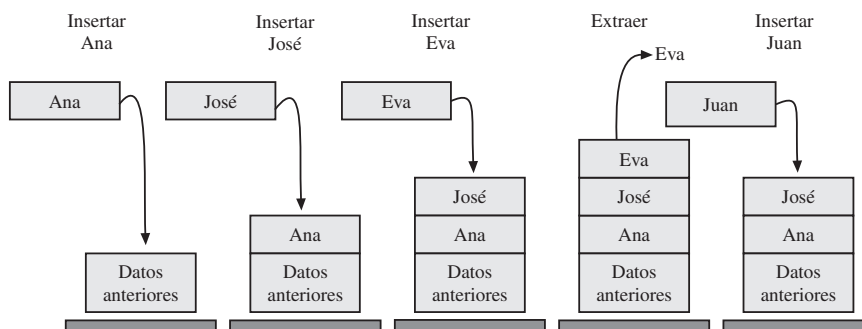


Figura 10.7. Ejemplo de operaciones en una pila.

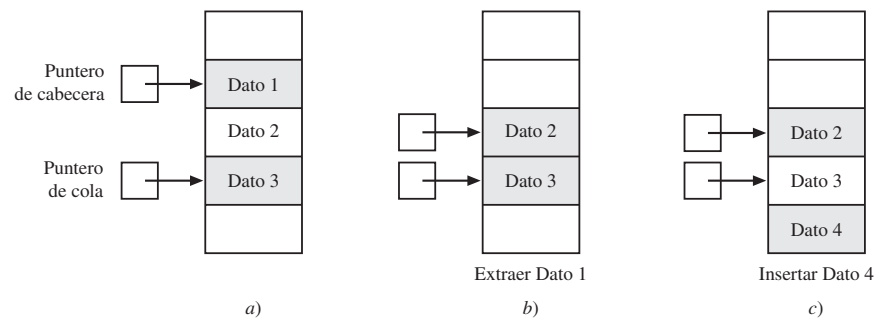
## 10.2.6. COLAS

Una **cola** es una lista en la que las inserciones se hacen por un extremo y las eliminaciones por el otro. En este caso, el primer elemento en entrar es el primero en salir, de ahí que se las denomine listas **FIFO** (*first-in, first-out*). El extremo por el que se sacan elementos se denomina **cabecera** y el extremo por el que se insertan elementos se denomina **cola**. Para implementar estas estructuras se necesitan dos **punteros**, uno para almacenar la dirección de la cabeza y otro para almacenar la dirección de la cola (Figura 10.8).

Las operaciones básicas que se pueden hacer con una pila son:

- Insertar elementos.
- Extraer elementos.
- Consultar si está vacía o no.

Suponiendo que la cola se implementa según el esquema de la Figura 10.8, que *PCabe* y *PCola* son los punteros (direcciones de memoria), donde se encuentran la cabecera y la cola de la lista, res-



**Figura 10.8.** a) cola con tres elementos, b) eliminación de un elemento y c) inserción de un elemento.

pectivamente, y que cada dato de la pila ocupa una posición de memoria, las operaciones de inserción y extracción en la cola se efectuarían de la siguiente forma:

*Inserción del dato D:*

$p\_cola \leftarrow p\_cola + 1$   
 $M(p\_cola) \leftarrow D$

*Extracción de un dato de la cola:*

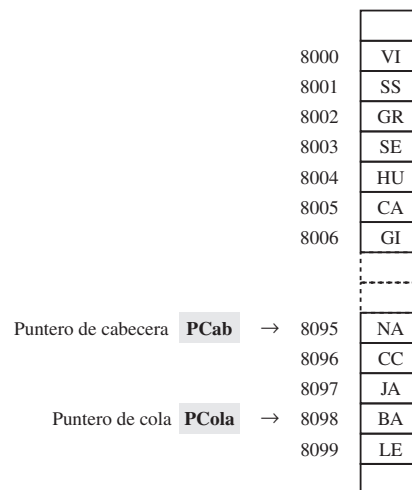
$D \leftarrow M(p\_cabeza)$   
 $p\_cabeza \leftarrow p\_cabeza + 1$

La estructura que acabamos de analizar se suele denominar **cola lineal**, y un problema que presenta en su almacenamiento es que la zona de memoria que va ocupando va desplazándose sucesivamente a las posiciones altas de memoria. Por ejemplo, si inicialmente el puntero de cabecera,  $PCab$ , estaba en la posición 8.000, después de que hayan entrado en la cola 124 elementos, la nueva posición del puntero sería  $PCab = 8.000 + 124 = 8.124$  (suponemos que cada uno de ellos ocupa una posición de memoria) aunque la cola estuviese vacía. Este problema se soluciona utilizando una **cola circular**. Para diseñar una cola circular hay que prever el máximo número de elementos que pueda haber en ella, y se reserva un bloque fijo de memoria para almacenarla. Cuando un puntero llega al límite de la zona reservada se desplaza al otro extremo de la misma, continuando el almacenamiento o extracción de los nuevos elementos que lleguen allí.

### EJEMPLO 10.13

Suponga que a partir de la posición de memoria 8.000 se almacena una cola circular de 100 elementos. Suponer que el puntero de cabecera está en la posición  $PCab = 8.095$  y el puntero de cola en  $PCola = 8.098$ . Indicar la evolución del contenido de la cola suponiendo los contenidos de memoria que se indican en la Figura 10.9 y las siguientes actividades:

Insertar MA  
 Extraer  
 Insertar GE  
 Insertar SA  
 Extraer  
 Extraer  
 Insertar LE  
 Insertar GI



**Figura 10.9.** Ejemplo de implementación de una cola en la memoria de un computador.

## SOLUCIÓN

Las operaciones que se realizarán serán las siguientes:

1. Insertar MA: Se añade un nuevo elemento en la cola; es decir:

$$PCola \leftarrow [PCola + 1]_{\text{rango } 8000 \text{ a } 8099} = 8899; M(8899) \leftarrow MA$$

2. Extraer: Se elimina un elemento de la cabecera de la cola; es decir:

$$Salida\_cola \leftarrow MP(8095) = NA; PCab \leftarrow [PCab + 1]_{\text{rango } 8000 \text{ a } 8099} = 8096$$

3. Insertar GE:

$$PCola \leftarrow [PCola + 1]_{\text{rango } 8000 \text{ a } 8099} = 8000; M(8000) \leftarrow GE$$

4. Insertar SA: Se añade un nuevo elemento en la cola; es decir:

$$PCola \leftarrow [PCola + 1]_{\text{rango } 8000 \text{ a } 8099} = 8001; M(8001) \leftarrow SA$$

5. Extraer: Se elimina un elemento de la cabecera de la cola; es decir:

$$Salida\_cola \leftarrow MP(8096) = CC; PCab \leftarrow [PCab + 1]_{\text{rango } 8000 \text{ a } 8099} = 8097$$

6. Extraer: Se elimina un elemento de la cabecera de la cola; es decir:

$$Salida\_cola \leftarrow MP(8097) = JA; PCab \leftarrow [PCab + 1]_{\text{rango } 8000 \text{ a } 8099} = 8098$$

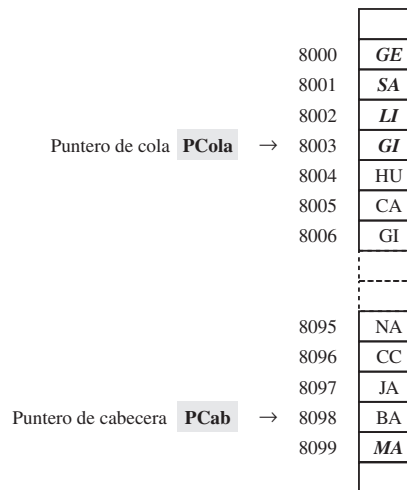
7. Insertar LE: Se añade un nuevo elemento en la cola; es decir:

$$PCola \leftarrow [PCola + 1]_{\text{rango } 8000 \text{ a } 8099} = 8002; M(8002) \leftarrow LE$$

8. Insertar GI: Se añade un nuevo elemento en la cola; es decir:

$$PCola \leftarrow [PCola + 1]_{\text{rango } 8000 \text{ a } 8099} = 8003; M(8003) \leftarrow GI$$

En consecuencia, después de las 5 inserciones y las 3 extracciones, el contenido de la memoria será el que se indica en la Figura 10.10.



**Figura 10.10.** Ejemplo de implementación de una cola en la memoria de un computador. Situación después de haber realizado 5 inserciones y 3 extracciones.

Observamos que las direcciones sucesivas seguidas por los punteros, dentro de la zona reservada a la lista, son de la siguiente forma:

8000, 8001, 8002, ..., 8098, 8099, 8000, 8001, 8002, ..., 8098, 8099, 8000, 8001, 8002, ..., 8098, 8099

Todo ocurre como si la última posición de memoria reservada para la zona de la lista (8099) fuese adjunta a la primera (8000); es decir, como si tuviésemos una cola circular. La implementación es muy sencilla porque basta con hacer los incrementos de los punteros *módulo 100*. Si *dbc* es la dirección inicial de la zona reservada a la cola y *tc* es el tamaño de la cola, los algoritmos que implementan la inserción y extracción son los siguientes:

*Inserción del dato D:*

$des1 \leftarrow (des1 + 1) \bmod tc$   
 $Pcola \leftarrow dbc + des1$   
 $M(Pcola) \leftarrow D$

*Extracción de un dato de la cola:*

$D \leftarrow M(Pcab)$   
 $des2 \leftarrow (des2 + 1) \bmod tc$   
 $Pcab \leftarrow dbc + des2$

### 10.2.7. ÁRBOLES

Un **árbol** es una estructura dinámica de datos formada por elementos del mismo tipo llamados **nodos** y un conjunto de líneas que conectan los nodos, llamadas **ramas**. El nodo superior se denomina **nodo raíz**. Los nodos de los extremos se denominan **nodos terminales** u **hojas**. Cada nodo tiene la estructura de un árbol y se denomina **subárbol**. A los nodos descendientes se les llama **hijos** y a los ascendentes **padres**. Se denomina **grado** de un nodo al número de subárboles que sustenta, y **orden** al mayor de los grados. El **nivel** de un nodo es la distancia desde él hasta el nodo raíz. La **altura** de un árbol es el nivel de la hoja más lejana del nodo raíz más 1.

En la Figura 10.11 se muestran los distintos elementos que forman un árbol. El grado del nodo raíz es 3, del N1 es 2 y del N3 es 1. El orden del árbol es 3.

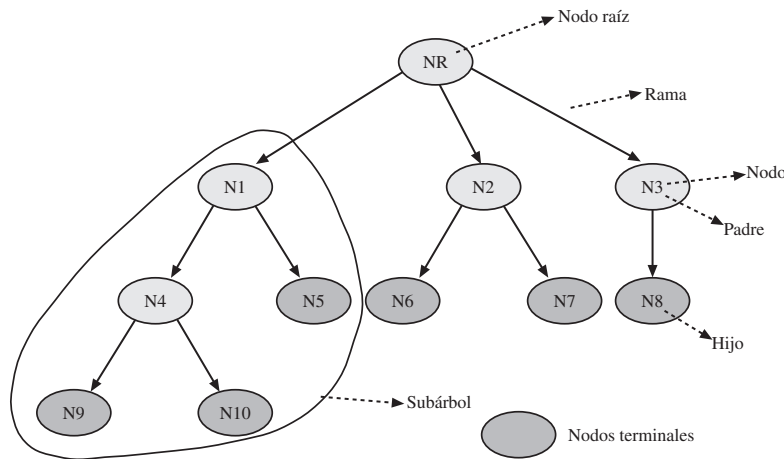


Figura 10.11. Elementos de un árbol.

Un caso particular de árboles son los árboles binarios. Un **árbol binario** es un árbol cuyos nodos no pueden tener más de dos subárboles. La altura de un árbol binario se puede predecir:

- La altura máxima ( $H_{max}$ ) de un árbol binario será igual al número de nodos ( $N$ ):

$$H_{max} = N$$

- La altura mínima ( $H_{min}$ ) viene dada por:

$$H_{max} = (\log_2 N) + 1 = (3,32 \cdot \log N) + 1$$

Al igual que una lista, un árbol se suele implementar en el interior de un computador utilizando enlaces (punteros). Habitualmente, para facilitar el desplazamiento a través del árbol para acceder a cualquier nodo, se suele representar internamente por medio de una tabla en la que cada fila corresponde a un nodo distinto. En la fila se incluye un puntero para dirigirse al padre, y los punteros hacia sus distintos hijos. Si el árbol es muy complejo la tabla resulta excesivamente grande, y entonces se implementan como una lista encadenada pero con tres punteros por nodo: al padre, al primer hijo y al siguiente hermano (Problema 10.18). Al árbol se accede directamente a través de su nombre y del puntero a la raíz.

Los **árboles binarios** se almacenan como una lista encadenada, teniendo cada elemento dos punteros, uno a cada hijo. Caso de que un nodo no tenga algún o ningún hijo se utiliza un patrón de bits preestablecido, que nosotros denominaremos *FIN*. Los árboles binarios también se pueden almacenar de forma estática, como los arrays, siguiendo la siguiente ordenación: primero los datos del nodo raíz (primer nivel del árbol), a continuación los datos de los 2 hijos del nodo raíz (segundo nivel) empezando de izquierda a derecha, a continuación los 4 nietos del nodo raíz (tercer nivel), luego los 8 biznietos del nodo raíz (cuarto nivel), y por último los  $2^{n-1}$  elementos del  $n$ -ésimo nivel. Caso de que falte algún nodo en el árbol a representar, se incluye un patrón de bits que identifique esa situación (*FIN*).

#### EJEMPLO 10.14

Indicar cómo se almacenaría el árbol de la Figura 10.12 de forma estática (sin enlaces). Suponga que se almacena a partir de la posición H'AA00 de memoria (contenido del **puntero raíz**), y que los datos de cada nodo ocupan cuatro palabras de memoria.

#### SOLUCIÓN

La solución se muestra en la Figura 10.13. Obsérvese que se ha tenido en cuenta que los datos de



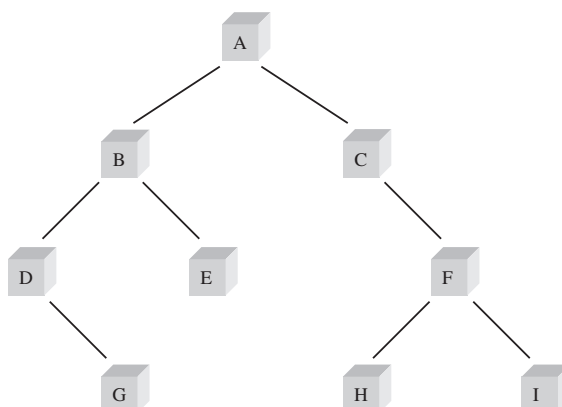


Figura 10.12. Ejemplo de árbol binario.

cada nodo ocupan cuatro palabras de memoria. Por otra parte, en las posiciones correspondientes a nodos que no existen, incluimos un patrón de bits que codifiquen este hecho (FIN en la figura).

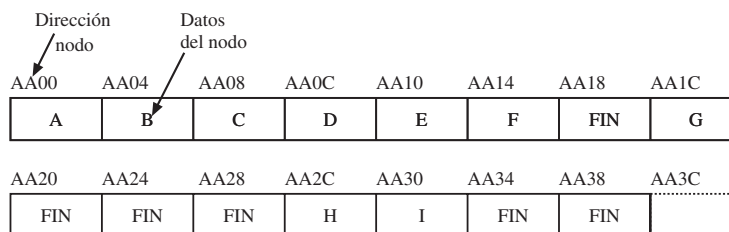


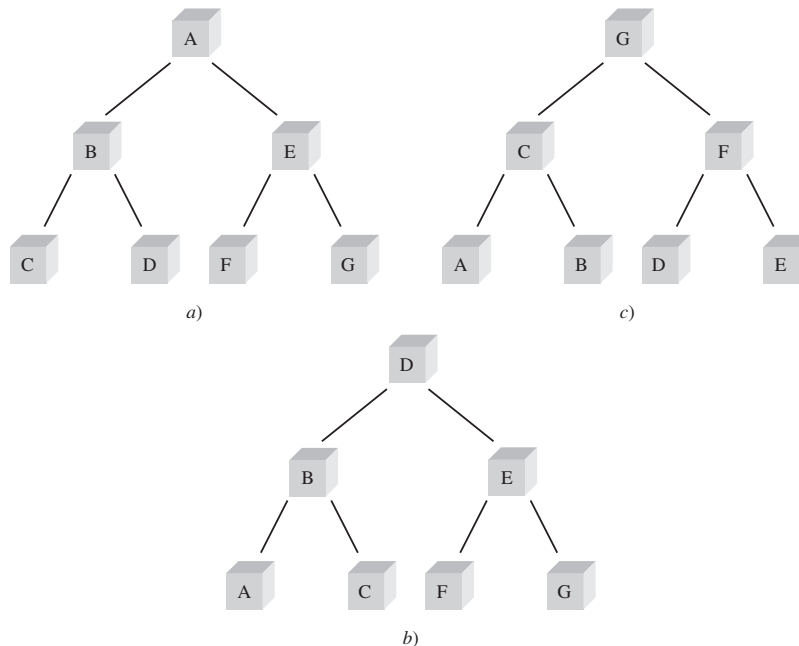
Figura 10.13. Almacenamiento estático del árbol binario de la figura anterior (Figura 10.12).

Las operaciones básicas que se pueden realizar con un árbol son:

- Buscar un elemento.
- Insertar elementos.
- Borrar elementos.
- Consultar si está vacío o no.

Las tres primeras operaciones suelen implicar realizar un recorrido por el árbol. El **recorrido de un árbol** supone realizar las operaciones necesarias para analizar o procesar la información de cada nodo, pasando por cada uno de ellos una sola vez, y siguiendo una secuencia predeterminada. Para un árbol binario, los procedimientos más habituales de recorrido son los siguientes:

- **Recorrido primero en anchura.** Se accede a los nodos por orden de nivel; así, primero se analiza el nodo raíz (nivel 1), luego los hijos del raíz (nivel 2), y así sucesivamente hasta llegar al nivel  $n$ .
- **Recorrido primero en profundidad.** Cualquier nodo de un árbol binario ( $N$ ) sustenta como máximo dos árboles: uno derecho (*right*,  $R$ ) y otro izquierdo (*left*,  $L$ ). Los recorridos tradicionales hacen referencia al orden en que se procesan el nodo ( $N$ ), el árbol izquierdo ( $L$ ) y el árbol derecho ( $R$ ); la Figura 10.14 muestra estos recorridos, que son los que se obtienen siguiendo las letras por orden alfabético:
  - **Recorrido en pre-orden** o **NLR**. Se procesa primero el nodo raíz, luego el árbol izquierdo y luego el árbol derecho (las siglas NLR indican este orden). (Figura 10.14a.)
  - **Recorrido en orden** o **LNR**. Se procesa primero el árbol izquierdo, luego nodo raíz y por último el árbol derecho (Figura 10.14b).
  - **Recorrido en post-orden** o **LRN**. Se procesa primero el árbol derecho, luego nodo raíz y por último el árbol izquierdo (Figura 10.14c).



**Figura 10.14.** Recorridos en profundidad de un árbol binario: a) pre-orden (NLR), b) orden (LNR) y c) post-orden (LRN).

La utilización de árboles tiene múltiples aplicaciones, como pueden ser facilitar las **búsquedas binarias**, para acceder rápidamente a un determinado registro asociado a un archivo o tabla ordenada (ver Capítulo 11, Problema 11.12), o en los procesadores de lenguajes para, por ejemplo, analizar las expresiones de un programa escrito en un lenguaje de alto nivel (Sección 12.3).

### 10.2.8. ESTRUCTURAS DEFINIDAS POR EL USUARIO. CLASES

En las Secciones 10.1.5 y 10.1.6 estudiamos los tipos de datos enumerado y subrango, que podían ser definidos por el usuario. Los lenguajes de programación modernos dan mayores posibilidades y versatilidad a los usuarios para crear tanto tipos como estructuras de datos. En la Sección 10.2.3 vimos cómo con la declaración:

```
struct
{char Nombre [10];
 char Autor;
 int Año;
 float Precio
} Disco
```

creábamos una nueva estructura heterogénea denominada **Disco**. Más interés aún tiene el poder crear un nuevo tipo de datos o tipo de estructuras. Esto, para el ejemplo anterior y en C, se hace de la siguiente forma:

```
typedef struct
{char Nombre [10];
 char Autor;
 int Año;
 float Precio
} TipoDisco
```

Obsérvese que ahora la declaración empieza por la palabra *typedef* (de “definición de tipo”). Una vez definido el tipo es posible definir variables (o estructuras) de dicho tipo. Así, podríamos definir las variables `discoOpera`, `discoLigera`, `discoJazz`, como variables del tipo `TipoDisco` de la siguiente forma:

```
TipoDisco discoOpera, discoLigera, discoJazz
```

Dentro del programa podremos utilizar sin problema las nuevas variables, pero las operaciones que se pueden hacer con ellas corresponden a las de sus tipos originales: el nombre y autor de tipo carácter, el año de tipo entero y el precio de tipo real.

Algunos lenguajes (como los lenguajes orientados a objetos C++, C# y Java) van más lejos, y por medio del concepto de **clase** permiten crear auténticos tipos y estructuras en los que se define tanto el método de almacenar los datos como el conjunto de funciones y operaciones que pueden realizarse con los elementos del tipo definido.

## 10.3. Conclusiones

Para facilitar la utilización de los computadores, y dada la diversa naturaleza de los datos que se procesan, se definen diversos tipos de datos y estructuras que permiten al programador hacer abstracción de la máquina que ejecutará sus programas. Son los **procesadores de lenguajes** (compiladores, intérpretes, etc.) quienes se encargarán de generar el código máquina adecuado para almacenar los datos y de hacer las llamadas a las funciones de biblioteca correspondientes para operar con ellos, todo de acuerdo con lo establecido en la definición del tipo de que se trate.

En la Figura 10.15 se enumeran los distintos tipos de datos y estructuras analizados en este capítulo. No todos los lenguajes de programación admiten los mismos tipos y estructuras, ni todos los posibles se han analizado en el presente capítulo, aunque sí los más relevantes.

- **TIPOS DE DATOS:**
  - Enteros.
  - Reales.
  - Lógicos.
  - Caracteres.
  - Definidos por el usuario:
    - Enumerados.
    - Subrangos.
- **ESTRUCTURAS DE DATOS:**
  - Arrays.
  - Cadenas de caracteres.
  - Registros.
  - Listas:
    - Generales:
      - ◆ Contiguas.
      - ◆ Enlazadas.
    - Restringidas:
      - ◆ Pilas.
      - ◆ Colas.
  - Árboles.
  - Estructuras de datos definidas por el usuario:
    - Definición de tipos en C (*typedef struct*).
    - Clases.

**Figura 10.15.** Tipos de datos y estructuras analizadas en el presente capítulo.

## Test



**T10.1.** En informática se denomina dato a cualquier:

- a) Variable o constante de tipo numérico.
- b) Variable o constante de tipo numérico o carácter.
- c) Cadena de variables o constantes de tipo numérico o carácter.
- d) Objeto manipulable por un computador.

**T10.2.** Si el número de bits con la que se representan 328 datos de tipo entero fuese  $n = 8$  bits, su cardinalidad sería:

- a) 328.
- b)  $2^{328}$ .
- c) 256.
- d) 8.

**T10.3.** La suma y el producto de datos de tipo real cumplen:

- a) Las mismas propiedades que los números reales.
- b) Unas u otras propiedades, dependiendo del computador (longitud de palabra).
- c) Todas las propiedades, pero no siempre la distributiva.
- d) La propiedad conmutativa, pero no siempre la asociativa ni la distributiva.

**T10.4.** Se pueden producir agotamientos y desbordamientos cuando se opera con datos de tipo:

- a) Entero.
- b) Real.
- c) Lógico.
- d) Entero, real o lógico.

**T10.5.** El resultado de una determinada operación lógica entre dos variables con valores:  $a = 0$  y  $b = 1$ , es  $z = 0$ ; dicha operación es:

- a) Y (AND).
- b) O (OR).
- c) XOR.
- d) NOY (NAND).

**T10.6.** El resultado de una determinada operación lógica entre dos variables con valores:  $a = 0$  y  $b = 1$ , es  $z = 1$ ; dicha operación:

- a) Es la Y (AND).
- b) Es la O (OR).
- c) Puede ser la Y o la O.
- d) Es la No O (NOR).

**T10.7.** El resultado de una determinada operación lógica entre dos variables con valores:  $a = 1$  y  $b = 1$ , es  $z = 0$ :

- a) Dicha operación es la Y (AND).
- b) Dicha operación es la O (OR).
- c) Dicha operación es la XOR.
- d) Dicha operación puede ser la Y o la O.

**T10.8.** El resultado de una determinada operación lógica entre dos variables con valores:  $a = 1$  y  $b = 1$ , es  $z = 1$ ; dicha operación:

- a) Es la Y (AND).
- b) Es la O (OR).
- c) Es la XOR.
- d) Puede ser la Y o la O.

**T10.9.** El siguiente tipo de datos no está normalizado:

- a) Tipo carácter.
- b) Tipo entero.
- c) Tipo lógico.
- d) Tipo enumerado.

**T10.10.** ORD es una función que se utiliza con datos de tipo carácter y que proporciona:

- a) El código numérico correspondiente a un carácter.
- b) El carácter correspondiente a un código numérico.
- c) El número de bits del carácter.
- d) La denominación del conjunto de caracteres que se está utilizando (ASCII, EBCDIC, Unicode, Fieldata, etc.).

**T10.11.** CHR es una función que se utiliza con datos de tipo carácter y que proporciona:

- a) El código numérico correspondiente a un carácter.
- b) El carácter correspondiente a un código numérico.
- c) El número de bits del carácter.
- d) La denominación del conjunto de caracteres que se está utilizando (ASCII, EBCDIC, Unicode, Fieldata, etc.).

**T10.12.** Por lo general el hardware del computador puede procesar directamente con datos de:

- a) Enteros.
- b) Enumerado y subrango.
- c) Tipo entero, lógico y carácter.
- d) Cualquier tipo.

**T10.13.** El tipo de datos real es:

- a) Un tipo de datos elemental.
- b) Un tipo de datos estructurado (una estructura de datos).
- c) Una estructura de datos heterogénea.
- d) Una estructura dinámica.

**T10.14.** Suponga definido el tipo de datos enumerado siguiente: **color** = {rojo, naranja, amarillo, verde, morado, azul}; el **PRED(SUC(morado))** es:

- a) Morado.
- b) Azul.
- c) Verde.
- d) Amarillo.

**T10.15.** Los elementos de un array homogéneo:

- a) Deben ser de tipo numérico (entero o real).
- b) Deben ser de tipo entero.
- c) Pueden ser de distinto tipo.
- d) Pueden ser de cualquier tipo, pero todos del mismo.

**T10.16.** La dimensión de un array viene dada por:

- a) El número de índices necesario para especificar cada elemento.
- b) Los valores máximos de los índices.
- c) El número total de elementos.
- d) El número de palabras que ocupa en la memoria principal.

**T10.17.** Los elementos de un registro se denominan:

- a) Punteros.
- b) Nodos.
- c) Campos.
- d) Índices.

**T10.18.** Los elementos de un registro:

- a) Deben ser de tipo numérico (entero o real).
- b) Deben ser de tipo carácter.
- c) Pueden ser de distinto tipo.
- d) Pueden ser de cualquier tipo, pero todos del mismo.

**T10.19.** Una estructura de datos en la que el primer elemento que entra es el último que se extrae se denomina:

- a) Cola.
- b) Árbol.
- c) Cadena.
- d) Pila.

**T10.20.** Una operación de inserción en una lista general:

- a) Añade un elemento delante del primero que ha llegado.
- b) Añade un elemento delante del último que ha llegado.
- c) Añade un elemento detrás del último que ha llegado.
- d) Puede añadir un elemento en cualquier lugar de la lista.

**T10.21.** Una operación de eliminación en una lista general:

- a) Elimina al primero que ha llegado.
- b) Elimina al último que ha llegado.
- c) Elimina un elemento, en el lugar que se especifique, de la lista.
- d) Elimina el elemento situado después de la cabecera.

**T10.22.** Una operación de extracción en una cola:

- a) Elimina al primero que ha llegado.
- b) Elimina al último que ha llegado.
- c) Elimina un elemento, en el lugar que se especifique, de la cola.
- d) Elimina el elemento situado después de la cabecera de la cola.

**T10.23.** Una operación de inserción en una cola:

- a) Añade un elemento delante del primero que ha llegado.
- b) Añade un elemento delante de su cabecera.
- c) Añade un elemento a continuación del último que ha llegado.
- d) Añade un elemento, en el lugar que se especifique, de la cola.

**T10.24.** Una operación de inserción en una pila:

- a) Añade un elemento delante del primero que ha llegado.
- b) Añade un elemento delante de su cabecera.

- c) Añade un elemento a continuación del último que ha llegado.

- d) Añade un elemento, en el lugar que se especifique, de la pila.

**T10.25.** Una operación de extracción en una pila:

- a) Elimina al primero que ha llegado.
- b) Elimina al último que ha llegado.
- c) Elimina un elemento, en el lugar que se especifique, de la pila.
- d) Elimina el elemento situado después de la cabecera.

**T10.26.** Las siguiente estructura de datos pueden ser heterogénea:

- a) Array.
- b) Cadena de caracteres.
- c) Lista.
- d) Registro.

**T10.27.** La siguiente estructura de datos es dinámica:

- a) Array.
- b) Registro.
- c) Lista.
- d) Cadena.

**T10.28.** Una lista FIFO es:

- a) Una cola.
- b) Una pila.
- c) Un árbol.
- d) Un puntero.

**T10.29.** Una lista LIFO es:

- a) Una cola.
- b) Una pila.
- c) Un árbol.
- d) Un puntero.

**T10.30.** El primer elemento que llegó a una pila es Pera, seguido de Uva, Fresa y Manzana. ¿Cuál es el primer elemento en salir?

- a) Pera.
- b) Uva.
- c) Fresa.
- d) Manzana.

**T10.31.** El primer elemento que llegó a una cola es Pera, seguido de Uva, Fresa y Manzana. ¿Cuál es el primer elemento en salir?

- a) Pera.
- b) Uva.
- c) Fresa.
- d) Manzana.

**T10.32.** Los elementos de un árbol se denominan:

- a) Punteros.
- b) Nodos.
- c) Campos.
- d) Índices.

**T10.33.** Un árbol es:

- a) Un tipo de datos elemental.
- b) Una estructura con datos de distinto tipo.
- c) Una estructura de datos heterogénea.
- d) Una estructura dinámica.

**T10.34.** Un árbol binario con 12 nodos puede tener como máximo una altura de:

- a) 12.
- b) 4.
- c) 7.
- d) 6.

**T10.35.** Un árbol binario con 12 nodos puede tener como mínimo una altura de:

- a) 4.
- b) 12.
- c) 3.
- d) 7.

**T10.36.** Un nodo de un árbol binario puede tener los siguientes grados:

- a) 2.
- b) 3.
- c) 0, 1 o 2.
- d) Cualquiera.

**T10.37.** El orden de un árbol binario:

- a) Es 2.
- b) Puede ser cualquiera, siempre que sea menor del número de nudos.
- c) 0, 1 o 2.
- d) Depende del número de nodos que tenga.

**T10.38.** En el recorrido pre-orden se accede primero a:

- a) El nodo raíz.
- b) El subárbol derecho.
- c) El subárbol izquierdo.
- d) Depende del número de nodos que tenga.

**T10.39.** En el recorrido post-orden se accede el último a:

- a) El nodo raíz.
- b) El subárbol derecho.
- c) El subárbol izquierdo.
- d) Depende del número de nodos que tenga.

**T10.40.** Las siguientes estructuras suelen implementarse con punteros:

- a) Array.
- b) Cadena.
- c) Listas y árboles.
- d) Registros.

## Problemas resueltos



### TIPOS DE DATOS

**P10.1.** Poner de manifiesto que cuando se opera con un número predeterminado de dígitos no se cumple la ley asociativa de la suma, supóngase que se dispone de un sistema que opera con datos reales con una mantisa de 4 dígitos (uno de ellos en la parte entera y los tres restantes de la parte decimal), la base del exponente es  $B = 10$ .

$$a = 2,325 \cdot 10^3; \quad b = 5,238 \cdot 10^6; \quad c = -5,231 \cdot 10^6$$

Comprobar que no se verifica

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

### SOLUCIÓN

Las operaciones las debemos realizar teniendo siempre en cuenta que el número se representa con 4 dígitos, uno en la parte entera y tres en la fraccionaria.

$$(a + b) = 2,325 \cdot 10^3 + 5,238 \cdot 10^6 = (0,002 + 5,238) \cdot 10^6 = 5,240 \cdot 10^6$$

$$(a + b) + c = 5,240 \cdot 10^6 - 5,231 \cdot 10^6 = 0,009 \cdot 10^6 = 9,000 \cdot 10^3$$

$$(b + c) = 5,238 \cdot 10^6 - 5,231 \cdot 10^6 = 0,007 \cdot 10^6 = 7,000 \cdot 10^3$$

$$a + (b + c) = 2,325 \cdot 10^3 + 7,000 \cdot 10^3 = 9,325 \cdot 10^3$$

luego:  $(a + b) + c \neq a + (b + c)$ , con lo que no se verifica la propiedad asociativa de la suma.

- P10.2.** Suponiendo que se dispone de un sistema que opera con datos reales con una mantisa de 4 dígitos (uno de ellos en la parte entera y los tres restantes de la parte decimal), la base del exponente es  $B = 10$ , probar que, debido a problemas de la limitación de número de bits que obligan a redondear a 3 cifras fraccionarias, no se verifica que:

$$(a + b) - b = a$$

siendo:

$$a = 2,325 \cdot 10^3; \quad b = 5,238 \cdot 10^6$$

**SOLUCIÓN**

En efecto:

$$(a + b) = 2,325 \cdot 10^3 + 5,238 \cdot 10^6 = (0,002 + 5,238) \cdot 10^6 = 5,240 \cdot 10^6$$

$$(a + b) - b = 5,240 \cdot 10^6 - 5,238 \cdot 10^6 = 0,002 \cdot 10^6 = 2,000 \cdot 10^3$$

con lo que  $(a + b) - b \neq a$ .

- P10.3.** Indicar el valor lógico resultado de evaluar las siguientes expresiones de relación:

**a)**  $A = 27; B = 32$

**b)**  $A = \text{"J"}; B = \text{"K"}$

**c)**  $A = \text{"D"}; B = \text{"d"}, C = \text{"!"}$

$$A + B - 59 \neq 0$$

$$B \leq A$$

$$C = B - A$$

**SOLUCIÓN**

**a)**  $A + B = 27 + 32 = 59$

$$A + B - 59 = 0$$

Luego  $A + B - 59 \neq 0$  es *verdad*.

**b)**  $A = \text{"J"}; B = \text{"K"}$

$$\text{ORD}(\text{"J"}) = 74; \text{ORD}(\text{"K"}) = 91$$

Luego como  $91 > 74$ , el resultado de la relación  $B \leq A$  es *falso*.

**c)**  $A = \text{"D"}; B = \text{"e"}, C = \text{"!"}$

$$\text{ORD}(\text{"D"}) = 68; \text{ORD}(\text{"e"}) = 101; \text{ORD}(!) = 33$$

Luego como  $101 - 68 = 33$ , el resultado de la relación  $C = B - A$  es *verdad*.

- P10.4.** Se tienen las siguientes expresiones:

**a)**  $A \geq 54,5 \text{ AND } A < 97,1$

**b)**  $A > \text{Verano OR } A < \text{Invierno}$

**c)**  $[\text{ORD}(A) \geq 65 \text{ AND } \text{ORD}(A) \leq 90] \text{ OR } [\text{ORD}(A) \geq 97 \text{ AND } \text{ORD}(A) \leq 122]$

donde los operadores **OR** y **AND** representan a los operadores suma lógica y producto lógico.

**a)** Indicar en cada uno de los casos los tipos que deben tener la variable  $A$  para que las expresiones sean correctas.

**b)** Obtener, en cada caso, el valor de  $A$  para que el resultado lógico de la evaluación de las expresiones sea verdad.

**SOLUCIÓN**

**a)**  $A$  y  $B$  deben ser de tipo real.

$A$  es cualquier valor mayor o igual que 54,5 y menor que 97,1; es decir,  $A = (54,5; 97,1]$ .

**b)**  $A$  y  $B$  debe ser variable de tipo enumerado; posiblemente:

$$\text{estaciones\_año} = \{\text{primavera, verano, otoño, invierno}\};$$

con lo que  $A$  será otoño.

- c) La variable  $A$  debe ser de tipo carácter.

Consultando la tabla de código ASCII-Latín 1 se tiene:

$$\text{ORD}(A) \geq 65 \Rightarrow A \geq "A"; \text{ORD}(A) \leq 90 \Rightarrow A \leq "Z"$$

es decir,  $A$  es el conjunto de letras mayúsculas.

Por otra parte:

$$\text{ORD}(A) \geq 97 \Rightarrow A \geq "a"; \text{ORD}(A) \leq 122 \Rightarrow A \leq "z"$$

es decir,  $A$  es el conjunto de letras minúsculas.

La unión (**OR**) de los dos conjuntos anteriores es el conjunto de letras mayúsculas y minúsculas del abecedario. En consecuencia, la expresión:

$$[\text{ORD}(A) \geq 65 \text{ AND } \text{ORD}(A) \leq 90] \text{ OR } [\text{ORD}(A) \geq 97 \text{ AND } \text{ORD}(A) \leq 122]$$

será *verdad* si  $A$  es una letra (mayúscula o minúscula).

## ARRAYS

- P10.5.** Obtener la ocupación de memoria (en bytes) del array VM(100,100, 3), suponiendo que es de tipo real: a) en simple precisión, b) en doble precisión y c) en cuádruple precisión.

### SOLUCIÓN

El tamaño del array será:

$$t_{VM} = 100 \cdot 100 \cdot 3 = 30.000 \text{ elementos}$$

- a) En simple precisión, cada elemento ocupa 32 bits = 4 Bytes; con lo que la ocupación total de memoria del array será:

$$C_{VM} = 30.000 \cdot 4 = 120.000 \text{ Bytes} = 118 \text{ KBytes}$$

- b) En doble precisión, cada elemento ocupa 64 bits = 8 Bytes; con lo que la ocupación total de memoria del array será:

$$C_{VM} = 30.000 \cdot 8 = 240.000 \text{ Bytes} = 235 \text{ KBytes}$$

- c) En cuádruple precisión, cada elemento ocupa 128 bits = 16 Bytes; con lo que la ocupación total de memoria del array será:

$$C_{VM} = 30.000 \cdot 16 = 480.000 \text{ Bytes} = 469 \text{ KBytes}$$

- P10.6.** Dado el array siguiente:

$$C = \begin{pmatrix} \text{naranja} & \text{violeta} & \text{verde} \\ \text{azul} & \text{rojo} & \text{amarillo} \\ \text{amarillo} & \text{naranja} & \text{azul} \end{pmatrix}$$

que está definido en el tipo de datos color: **color** = {rojo, naranja, amarillo, verde, morado, azul}, obtener los valores de:

- a)  $C(3,1)$ .  
 b)  $\text{SUC}(C(3,2))$ .  
 c)  $\text{PRED}(C(2,3))$ .

### SOLUCIÓN

- a)  $C(3,1) = \text{amarillo}$ .  
 b)  $\text{SUC}(C(3,2)) = \text{SUC}(\text{naranja}) = \text{amarillo}$ .  
 c)  $\text{PRED}(C(2,3)) = \text{PRED}(\text{amarillo}) = \text{naranja}$ .



**P10.7.** Suponga que un array,  $A(iM, jM, kM)$  de 3 dimensiones (donde  $iM$ ,  $jM$  y  $kM$  son los valores máximos de las dimensiones) es almacenado por un determinado procesador de lenguajes por columnas, a partir de la dirección  $db$ . Obtener una expresión general que nos proporcione la posición exacta en memoria de un elemento  $A(i, j, k)$  genérico del array.

#### SOLUCIÓN

Para entender fácilmente la solución del problema, supongamos que tengamos el siguiente array de 2 dimensiones:

$$A = \begin{bmatrix} A(1,1) & A(1,2) & A(1,3) \\ A(2,1) & A(2,2) & A(2,3) \\ A(3,1) & A(3,2) & A(3,3) \end{bmatrix}$$

Si sus elementos se almacenan por columnas, se almacenarán en este orden:

$$A(1,1), A(2,1), A(3,1), A(1,2), A(2,2), A(3,2), A(1,3), A(2,3), A(3,3)$$

y sus direcciones en memoria serán las que se indican en la siguiente tabla:

$$\begin{Bmatrix} db & db + 3 & db + 6 \\ db + 1 & db + 4 & db + 7 \\ db + 2 & db + 5 & db + 8 \end{Bmatrix}$$

es decir, por ejemplo, el elemento:

- $A(2,1)$  se almacena en  $db + 1$ .
- $A(2,2)$  se almacena en  $db + 4$ .
- $A(3,2)$  se almacena en  $db + 5$ .
- $A(3,3)$  se almacena en  $db + 8$ .

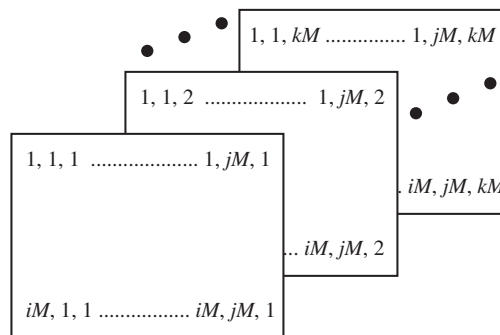
Generalizando,  $A(i, j)$  se almacenará en la posición:  $p_{ij} = db + 3 \cdot (j - 1) + i - 1$ .

Para tres dimensiones (Figura 10.16) se almacenará primero el primer plano ( $k = 1$ ), la primera columna ( $j = 1$ ), luego el primer plano ( $k = 1$ ), la segunda columna ( $j = 2$ ), etc., y, finalmente, el último plano ( $k = kM$ ), primera columna ( $j = 1$ ), etc. En definitiva, la expresión que nos da la **dirección polinomial** o posición en función de los índices del elemento es:

$$p_{ij} = db + (k - 1) \cdot jM \cdot iM + (j - 1) \cdot iM + i + 1$$

**P10.8.** ¿Qué estructuras de datos son más apropiadas para representar las siguientes informaciones?

- a) La lista de los alumnos matriculados en una determinada asignatura.
- b) Los movimientos de una tarjeta de crédito.
- c) El horario de llegadas y salidas de aviones de un aeropuerto.



**Figura 10.16.** Representación de un array de tres dimensiones.

## SOLUCIÓN

- a) Array lineal de cadenas de caracteres, o lista encadenada siendo cada elemento un registro.
- b) Registros con campos tales como: fecha, tipo de movimiento, debe, haber y saldo.
- c) Lista encadenada, de forma que se pueda actualizar con facilidad. Cada elemento de la lista será un registro.

## CADENAS DE CARACTERES

**P10.9.** Suponga que un determinado procesador de lenguajes dispone de las siguientes funciones para operar con cadenas de caracteres:

- **Concatenar:** **CON**(a,b), concatena *a* y *b*.
- **Extraer** una subcadena de una cadena. **EXT**(n:m), extrae de los caracteres *n* a *m*, inclusive.
- **Comparar** dos cadenas: **COM**(a,b), proporciona el valor 0 si  $a = b$ ; 1 si  $a < b$  y 2 si  $a > b$ .
- **Obtener la longitud**, **LON**(a), proporciona el número de caracteres que contiene la cadena *a*.
- **Identificar subcadena:** **IDEN**(a,b), proporciona la posición a partir de la cual se encuentra la subcadena *b* en la cadena *a*. Si el resultado es 0, indica que no existe dicha cadena en *a*.

Si *a* = “Era de noche y sin embargo llovía”, *b* = “sin”, *c* = “descanso”.

Indique cuál debe ser el tipo de la variable *v* para que las expresiones siguientes sean correctas, y el valor que se almacena en *v* después de evaluar cada una de ellas.

- a)  $d = \text{CON}(a,b)$ ;  $v = \text{CON}(d,c)$
- b)  $v = \text{EXT}(7:11)$
- c)  $v = \text{COM}(b, \text{EXT}(16:3))$
- d)  $v = \text{EXT}(20:20 + \text{LON}(b))$
- e)  $v = \text{IDEN}(a,b)$

## SOLUCIÓN

- a) *v* debe ser una cadena de caracteres.  
 $d = \text{CON}(a,b) = \text{“Era de noche y sin embargo llovía sin”}$ .  
 $v = \text{CON}(d,c) = \text{“Era de noche y sin embargo llovía sin descanso”}$ .
- b) *v* debe ser una cadena de caracteres.  
 $v = \text{EXT}(7:11) = \text{“noche”}$ .
- c) *v* debe ser una cadena de caracteres.  
 $v = \text{COM}(b, \text{EXT}(16:18)) = \text{COM}(b, \text{“sin”}) = \text{“descanso sin”}$ .
- d) *v* debe ser una cadena de caracteres.  
 $v = \text{EXT}(20:20 + \text{LON}(b)) = \text{EXT}(20:23) = \text{“emb”}$ .
- e) *v* debe ser un entero.  
 $v = \text{IDEN}(a,b) = 16$ .

## LISTAS

**P10.10.** Suponga que tiene la lista encadenada que se indica en la Tabla 10.2, y cuyo puntero de cabecera contiene la dirección AC130.

- a) ¿Qué nombre se obtiene leyendo de forma ordenada la lista?
- b) Modificar los enlaces de forma que aparezca el nombre “Juan”.

Dirección	Dato	Puntero al siguiente
AC128	Q	AC12E
AC12A	N	00000
AC12C	O	AC136
AC12E	U	AC134
AC130	J	AC12C
AC132	R	AC17B
AC134	I	AC12A
AC136	A	AC128

Tabla 10.2. Lista encadenada del Problema 10.10.

## SOLUCIÓN

a) Orden de la lista:

Enlace →	AC130	AC12C	AC136	AC128	AC12E	AC134	AC12A	0000
Contenido →	J	O	A	Q	U	I	N	<i>final</i>

b) Nuevos enlaces:

Contenido →	J	U	A	N	<i>final</i>
Enlace →	AC130	AC12E	AC136	AC12A	0000

luego la lista almacenada quedará como sigue:

Dirección	Dato	Puntero al siguiente
AC128	Q	AC12E
AC12A	N	<b>00000</b>
AC12C	O	AC136
AC12E	U	<b>AC136</b>
AC130	J	<b>AC12E</b>
AC132	R	AC17B
AC134	I	AC12A
AC136	A	<b>AC12A</b>

Tabla 10.3. Lista encadenada, solución del Problema 10.10.

**P10.11.** Suponga que tiene la lista encadenada que se indica en la Tabla 10.4. La lista se denomina NOMBRES y el puntero de cabecera contiene la dirección C13E.

Dirección	Dato	Puntero al siguiente
C128	Juan	C144
C130	Pedro	C136
C132	Alberto	C13C
C134	Nuria	C138
C136	María	C134
C138	Félix	C13A
C13A	Beatriz	0000
C13C	Laura	C128
C13E	Carlos	C132
C140	Susana	C142
C142	Marta	C130
C144	Enrique	C140
C146		
C148		

Tabla 10.4. Lista encadenada del Problema 10.11.

Obtener:

- a) El tercer elemento de la lista.
- b) Suponga que entre el tercer y el cuarto elemento hay que incluir “Fernando”, que se almacenaría en la posición C146 de memoria, ¿qué modificaciones hay que hacer en la lista?
- c) Suponga que deseamos eliminar el dato María, ¿qué modificaciones habría que hacer en la lista?

#### SOLUCIÓN

- a) Los primeros elementos de la lista, por orden son (entre paréntesis incluimos su dirección): (C13E) Carlos; (C132) Alberto; (C13C) Laura. Luego el tercer elemento de la lista es Laura.
- b) El nuevo elemento debe ir inmediatamente después de Laura e inmediatamente antes de (C128) Juan, por lo que Laura deberá apuntar a (C146) Fernando, y Fernando a (C128) Juan. Estos son los únicos cambios que hay que hacer.
- c) Para borrar (C136) María, que apunta a C134, habría que cambiar el puntero del elemento anterior a María (el de Pedro) por C134, y eso es suficiente.

Como consecuencia de los cambios anteriores, el contenido de memoria quedaría como muestra la Tabla 10.5.

Obsérvese que aunque María continúa en la memoria, no pertenece a la lista, ya que ninguna dirección apunta a ese elemento (C136).

- P10.12.** Modificar los punteros de la lista encadenada de la Tabla 10.5, de forma que todos sus elementos aparezcan ordenados alfabéticamente.

#### SOLUCIÓN

El puntero de cabecera habría que modificarlo al primer nombre de la lista “Alberto”, dirección: C132.

A partir de él buscamos sucesivamente los nombres por orden alfabético, colocando como puntero la dirección del siguiente nombre. El resultado es el que se indica en la Tabla 10.6.

Dirección	Dato	Puntero al siguiente
C128	Juan	C144
C130	Pedro	<b>C134</b>
C132	Alberto	C13C
C134	Nuria	C138
C136	María	C134
C138	Félix	C13A
C13A	Beatriz	0000
C13C	Laura	<b>C146</b>
C13E	Carlos	C132
C140	Susana	C142
C142	Marta	C130
C144	Enrique	C140
C146	<b>Fernando</b>	<b>C128</b>
C148		

Tabla 10.5. Problema 10.11.

Dirección	Dato	Puntero al siguiente
C128	Juan	C13C
C130	Pedro	C140
C132	Alberto	C13A
C134	Nuria	C130
C136	María	C142
C138	Félix	C128
C13A	Beatriz	C13E
C13C	Laura	C136
C13E	Carlos	C144
C140	Susana	0000
C142	Marta	C132
C144	Enrique	C138
C146		
C148		

Tabla 10.6. Problema 10.12.

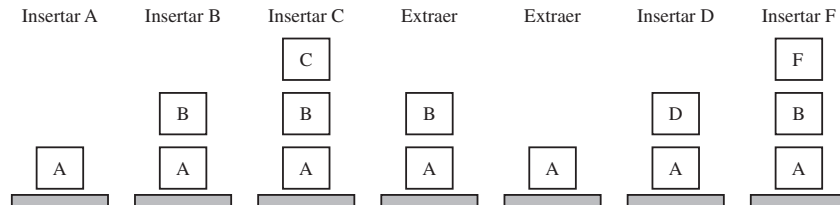
## PILAS

- P10.13.** Una pila inicialmente se encuentra vacía; si se realizan las operaciones que se indican a continuación, indicar los datos que se van extrayendo y el contenido final de la pila.

- 1) Insertar A    2) Insertar B    3) Insertar C    4) Extraer
- 5) Extraer    6) Insertar D    7) Insertar F

## SOLUCIÓN

En la Figura 10.17 se muestra la solución.



**Figura 10.17.** Ejemplo de operaciones realizadas con una pila.

**P10.14.** Suponga una pila que se gestiona en un bloque de memoria principal utilizando un puntero de pila (*SP*, de *Stack Pointer*), siendo la base de la pila la dirección 4A01 y el contenido del puntero de pila  $SP = 4BFF$ . Si el contenido de memoria es el que se da en la Tabla 10.7:

- a) ¿Cuántos elementos hay en la pila (suponiendo que cada uno de ellos ocupa una palabra)?
- b) ¿Qué valor se obtiene de la pila al ejecutar una operación de extracción?
- c) ¿Cuál es el nuevo valor de  $SP$ ?

	Dirección	Contenido
	4BF9	EA48
	4BFA	F35A
	4BFB	E7BC
	4BFC	2B89
	4BFD	5FFF
	4BFE	37AA
SP→	4BFF	42BC
	4A00	7ACD
	4A01	4BCC

**Tabla 10.7.**

## SOLUCIÓN

- a) Como la base de la pila es 4A01 y el puntero de la cabecera de la pila contiene 4BFF, la pila contiene dos elementos (7ACD y 42BC).
- b) Al hacer una operación de extracción se obtiene  $M(SP) = M(4BFF) = 7ACD$ .
- c) El nuevo valor del puntero es  $SP \leftarrow SP + 1 = 4A00$ .

**P10.15.** Suponga una pila que se gestiona en un bloque de memoria principal utilizando un puntero de pila ( $SP$ ), y que el valor inicial de este puntero es  $SP = H'4A00$ . En cada posición de memoria se incluye un dato de la pila, y éstos se van almacenando de las posiciones altas a bajas de memoria. Suponga que se realizan en la pila sucesivamente las siguientes operaciones:

- 1) Insertar C    2) Insertar a    3) Insertar l    4) Insertar a
- 5) Insertar d    6) Insertar e    7) Insertar r    8) Extraer
- 9) Extraer    10) Insertar o

Indicar:

- a) Cómo irían variando los contenidos de la memoria al hacer las operaciones.
- b) El algoritmo genérico que habría que realizar con el puntero y la memoria para implementar las inserciones.
- c) El algoritmo genérico que habría que realizar con el puntero y la memoria para implementar las extracciones.

SOLUCIÓN

- Insertar C:  $SP \leftarrow 4A00 - 1 = 4BFF;$   $M(4BFF) \leftarrow C.$
- Insertar a:  $SP \leftarrow 4BFF - 1 = 4BFE;$   $M(4BFE) \leftarrow a.$
- Insertar l:  $SP \leftarrow 4BFE - 1 = 4BFD;$   $M(4BFD) \leftarrow l.$
- Insertar a:  $SP \leftarrow 4BFD - 1 = 4BFC;$   $M(4BFC) \leftarrow a.$
- Insertar d:  $SP \leftarrow 4BFC - 1 = 4BFB;$   $M(4BFB) \leftarrow d.$
- Insertar e:  $SP \leftarrow 4BFB - 1 = 4BFA;$   $M(4BFA) \leftarrow e.$
- Insertar r:  $SP \leftarrow 4BFA - 1 = 4BF9;$   $M(4BF9) \leftarrow r.$
- Extraer:  $dato \leftarrow M(4BF9) = r;$   $SP \leftarrow 4AF9 + 1 = 4BFA.$
- Extraer:  $dato \leftarrow M(4BFA) = e;$   $SP \leftarrow 4AFA + 1 = 4BFB.$
- Insertar o:  $SP \leftarrow 4BFB - 1 = 4BFA;$   $M(4BFA) \leftarrow o.$

Los contenidos de memoria finales serían los siguientes:

	Dirección	Contenido
SP→	4BF9	
	4BFA	<i>o</i>
	4BFB	<i>d</i>
	4BFC	<i>a</i>
	4BFD	<i>l</i>
	4BFE	<i>a</i>
	4BFF	<i>C</i>
	4A00	s
	4A01	a

b) Inserciones:

$SP \leftarrow SP - 1 = 4BFF;$      $M(SP) \leftarrow dato\_a\_insertar$

c) Extracciones:

$dato \leftarrow M(SP);$      $SP \leftarrow SP + 1$

COLAS

**P10.16.** Una lista inicialmente se encuentra vacía; si se realizan las operaciones que se indican a continuación, indicar los datos que se van extrayendo y el contenido final de la lista.

- 1) Insertar A
- 2) Insertar B
- 3) Insertar C
- 4) Extraer
- 5) Extraer
- 6) Insertar D
- 7) Insertar F

SOLUCIÓN

En la Figura 10.18 se muestra la solución.

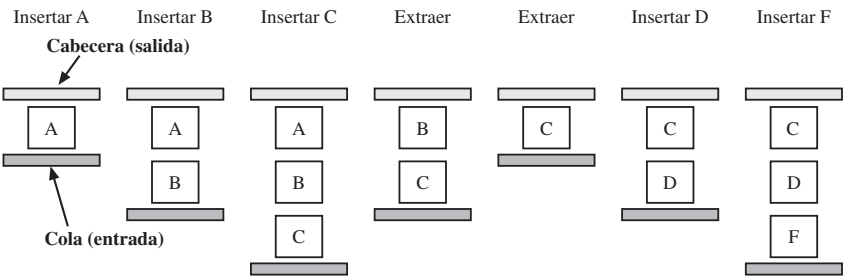


Figura 10.18. Ejemplo de operaciones realizadas con una pila.

**P10.17.** Suponga una cola que se gestiona en la memoria principal utilizando un puntero de cabecera (*PCab*) y otro de final de cola (*PCola*), que el valor inicial de estos punteros es  $PCab = PCola = H'4BFA$ . En cada posición de memoria se incluye un dato de la cola, y éstos se van almacenando en las posiciones altas de memoria y extrayendo por las bajas. Suponiendo que sucesivamente se realizan las siguientes operaciones:

- 1) Insertar M      2) Insertar a      3) Insertar ñ      4) Insertar a      5) Extraer
- 6) Insertar m      7) Insertar e      8) Extraer      9) Extraer

Obtener:

- a) La variación los contenidos de la memoria al hacer estas operaciones en la cola.
- b) Un algoritmo genérico para las operaciones de inserción.
- c) Un algoritmo genérico para las operaciones de extracción.

SOLUCIÓN

$$PCab = PCola = H'4BFA$$

- Insertar M:  $PCola \leftarrow 4BFA + 1 = 4BFB$ ;  $M(4BFB) \leftarrow M$ .
- Insertar a:  $PCola \leftarrow 4BFB + 1 = 4BFC$ ;  $M(4BFC) \leftarrow a$ .
- Insertar ñ:  $PCola \leftarrow 4BFC + 1 = 4BFD$ ;  $M(4BFD) \leftarrow ñ$ .
- Insertar a:  $PCola \leftarrow 4BFD + 1 = 4BFE$ ;  $M(4BFE) \leftarrow a$ .
- Extraer:  $dato\_extraido \leftarrow M(4BFA) = X$ ;  $PCab \leftarrow 4BFA + 1 = 4BFB$ .
- Insertar m:  $PCola \leftarrow 4BFE + 1 = 4BFF$ ;  $M(4BFF) \leftarrow m$ .
- Insertar e:  $PCola \leftarrow 4BFF + 1 = 4C00$ ;  $M(4C00) \leftarrow e$ .
- Extraer:  $dato\_extraido \leftarrow M(4BFB) = M$ ;  $PCab \leftarrow 4BFB + 1 = 4BFC$ .
- Extraer:  $dato\_extraido \leftarrow M(4BFC) = a$ ;  $PCab \leftarrow 4BFC + 1 = 4BFD$ .

Al final la memoria quedará con los contenidos que se muestran en la tabla siguiente:

	Dirección	Contenido	
PCab,PCola (posición inicial) →	4BF4	S	Cola resultante
	4BF5	T	
	4BF6	R	
	4BF7	U	
	4BF8	V	
	4BF9	Z	
	4BFA	X	
	4BFB	M	
	4BFC	A	
	4BFD	ñ	
PCab →	4BFE	a	
	4BFF	m	
PCola →	4C00	e	
	4C01	R	

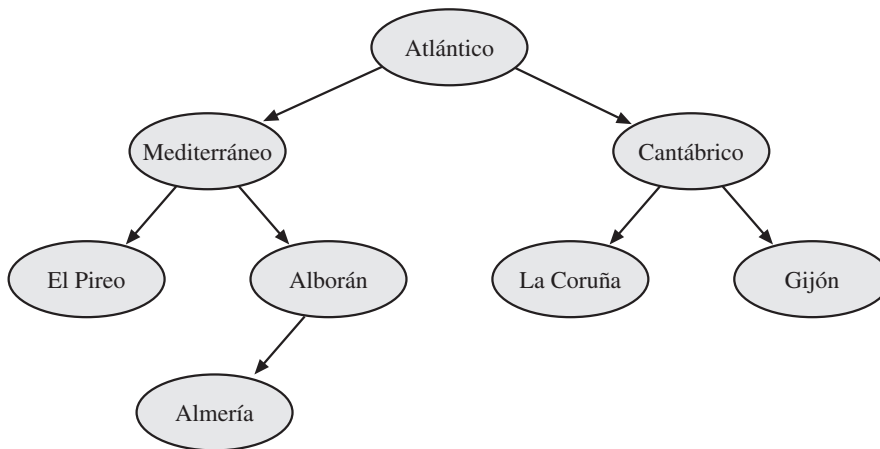
b) Inserción:

$$PCola \leftarrow PCola - 1; \quad M(PCola) \leftarrow \text{dato\_a\_insertar}$$

c) Extracción:

$$\text{dato\_extraido} \leftarrow M(PCab); \quad PCab \leftarrow PCab + 1$$

## ÁRBOLES



**Figura 10.19.** Ejemplo de árbol.

**P10.18.** Para el árbol de la Figura 10.19, suponiendo que cada nodo contiene 4 Bytes de información, obtener:

- Una representación estática en memoria; suponiendo que el contenido del puntero raíz es AC00.
- Una representación en memoria por medio de enlaces; suponiendo que el contenido del puntero raíz es AC00 y que cada puntero se almacena en 2 Bytes.
- Lo que ocupa en memoria en las dos representaciones anteriores.

### SOLUCIÓN

a) *Representación estática:*

Para establecer las direcciones suponemos que el direccionamiento se da en bytes. La solución es la que se indica a continuación:

AC00	AC04	AC08	AC0C	AC10
Atlántico	Mediterráneo	Cantábrico	El Pireo	Alborán
AC14	AC18	AC1C	AC20	AC24
La Coruña	Gijón	Nulo	Nulo	Almería
AC28	AC2C	AC30	AC34	AC38
Nulo	Nulo	Nulo	Nulo	Nulo



***b) Representación dinámica:***

<b>AC00</b>	<b>AC04</b>	<b>AC06</b>
Atlántico	AC28	AC30
<b>AC08</b>	<b>AC0C</b>	<b>AC1E</b>
La Coruña	<i>NULO</i>	<i>NULO</i>
<b>AC10</b>	<b>AC14</b>	<b>AC16</b>
Alborán	AC20	<i>NULO</i>
<b>AC18</b>	<b>AC1C</b>	<b>AC1E</b>
El Pireo	<i>NULO</i>	<i>NULO</i>
<b>AC20</b>	<b>AC24</b>	<b>AC26</b>
Almería	<i>NULO</i>	<i>NULO</i>
<b>AC28</b>	<b>AC2C</b>	<b>AC2E</b>
Mediterráneo	AC18	<i>AC10</i>
<b>AC30</b>	<b>AC34</b>	<b>AC36</b>
Cantábrico	AC08	AC38
<b>AC38</b>	<b>AC3C</b>	<b>AC3E</b>
Gijón	<i>NULO</i>	<i>NULO</i>

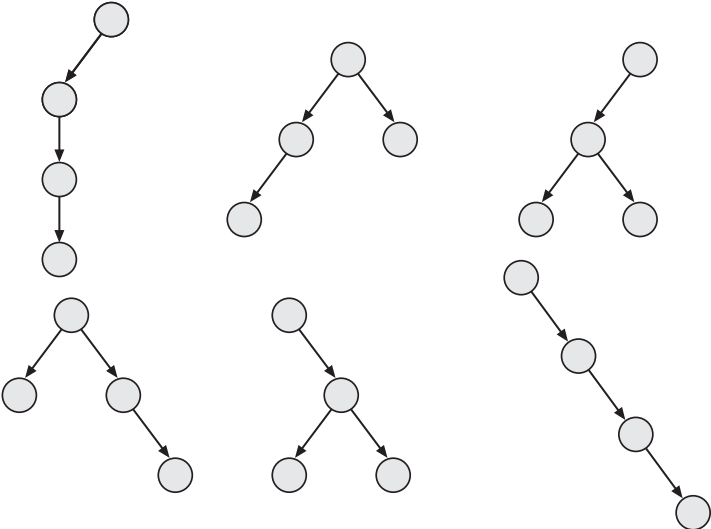
**c) Ocupación de memoria:**

- *Representación estática.* El árbol binario completo ocupa (teniendo en cuenta las hojas vacías):  $2^4 - 1 = 15$ ; como cada nodo ocupa 4 Bytes, en total:  $C = 15 \cdot 4 = 60$  Bytes.
- *Representación dinámica.* Hay 8 nodos, y cada uno ocupa:
  - Información del nodo: 4 Bytes.
  - Información de los 2 punteros de cada nodo:  $2 \cdot 2$  Bytes/puntero = 4 Bytes.

Total =  $8 \cdot 8 = 64$  Bytes

**P10.19.** Dibujar todos los posibles árboles binarios con 4 nodos.

## SOLUCIÓN



**Figura 10.20.** Posibles árboles de 4 nodos. La estructura primera y última son equivalentes.

## Problemas propuestos



## TIPO DE DATOS

**P10.20.** Indicar el valor lógico resultado de evaluar las siguientes expresiones de relación:

a)  $A = 27; B = 27$

$$A + B \geq 62$$

b)  $A = "3"; B = "a"$

$$A \leq B$$

c)  $A = "2"; B = "5", C = "g"$

$$C = A + B$$

**P10.21.** Se tienen las siguientes expresiones:

a)  $A \geq 1$  AND  $A \leq 20$  AND  $A - \text{int}(A) = 0$  (la función *int* obtiene la parte entera del argumento);

b)  $A > \text{CHR}(47)$  AND  $A < ":"$ ;

c)  $A > \text{CHR}(96)$  AND  $\text{ORD}(A) < 123$ ;

donde los operadores **OR** y **AND** representan a los operadores suma lógica y producto lógico, respectivamente:

a) Indicar en cada uno de los casos los tipos que deben tener la variable *A* para que las expresiones sean correctas.

b) Obtener, en cada caso, el valor de *A* para que el resultado lógico de la evaluación de las expresiones sea verdad.

## ARRAYS

**P10.22.** Obtener la ocupación de memoria (en bytes) del array  $R(1000, 10, 30)$ , suponiendo que es de tipo: a) entero con  $n = 16$  bits; b) real en simple precisión; c) real en doble precisión, y d) real en cuádruple precisión.

**P10.23.** Dado el array siguiente:

$$S = \begin{pmatrix} \text{lunes} & \text{lunes} & \text{martes} & \text{martes} \\ \text{viernes} & \text{martes} & \text{miercoles} & \text{martes} \\ \text{miercoles} & \text{sabado} & \text{miercoles} & \text{jueves} \\ \text{do min go} & \text{lunes} & \text{do min go} & \text{jueves} \end{pmatrix}$$

que está definido en el tipo de datos *dia\_semana*, obtener los valores de:

a)  $S(4, 1)$ .

b)  $\text{SUC}(S(3, 3))$ .

c)  $\text{PRED}(S(4, 4))$ .

**P10.24.** Indique cómo se almacenaría en memoria el array:

$$\begin{matrix} A & B & C \\ D & E & F \\ G & H & I \end{matrix}$$

suponiendo:

a) Que se almacenase por filas.

b) Que se almacenase por columnas.

**P10.25.** Suponga que un determinado procesador de lenguajes almacena por filas y a partir de la dirección  $db = 50$  un array declarado como  $A[100][100][50]$ . Obtener la dirección de memoria donde se almacenará el elemento  $A[7][5][32]$ .

*Nota:* El programa está escrito en Java.

**P10.26.** ¿Qué estructuras de datos son más apropiadas para representar las siguientes informaciones?

a) La información relativa a cada uno de los empleados de una empresa.

b) Los datos de temperatura de cada día, recogidos cada cuatro horas, en una estación meteorológica.

c) Las peticiones que van llegando al sistema operativo para utilizar una determinada impresora conectada en red.

## CADENAS DE CARACTERES

**P10.27.** Suponga que un determinado procesador de lenguajes dispone de las siguientes funciones para operar con cadenas de caracteres:

- **Concatenar:**  $\text{CON}(a, b)$ , concatena *a* y *b*.
- **Extraer** una subcadena de una cadena.  $\text{EXT}(a, n:m)$ , extrae de la cadena *a* los caracteres *n* a *m*, inclusive.
- **Comparar** dos cadenas:  $\text{COM}(a, b)$ , proporciona el valor 0 si  $a = b$ ; 1 si  $a < b$  y 2 si  $a > b$ .
- **Obtener la longitud.**  $\text{LON}(a)$ , proporciona el número de caracteres que contiene la cadena *a*.
- **Identificar subcadena:**  $\text{IDEN}(a, b)$ , proporciona la posición a partir de la cual se encuentra la subcadena *b* en la cadena *a*. Si el resultado es 0, indica que no existe dicha cadena en *a*.

Si  $x = \text{"Siete pelícanos de plumas oscuras y moteadas volaban bajo sobre el verde mar"}$ ,  $y = \text{"plumas"}$ .

Indique cuál debe ser el tipo de la variable *z* para que las expresiones siguientes sean correctas, y el valor que se almacena en *z* después de evaluar cada una de ellas:

a)  $\text{COMP}(\text{EXT}(19:6), y)$ .

b)  $\text{COMP}(\text{EXT}(y, 1:5), \text{EXT}(a, 1:5))$ .

c)  $\text{IDEN}(a, b)$ .

## LISTAS

**P10.28.** Suponga que se tiene la lista encadenada que se indica en la tabla siguiente.

Dirección	Dato	Puntero al siguiente
A4810	Zaragoza	00000
A4813	Ávila	A4840
A4817	Cáceres	A4837
A481C	Oviedo	A4827
A4820	La Coruña	A481C
A4823	Santander	A4830
A4827	Palencia	A483C
A482C	Albacete	A4813
A4830	Zamora	A4810
A4833	Salamanca	A4823
A4837	Granada	A4820
A483C	Pamplona	A4833
A4840	Burgos	A4817
A4843		
A4847		

La lista se denomina CIUDADES y su inicio se encuentra en la posición A482C.

Obtener:

- El cuarto elemento de la lista.
- Suponga que se desea incluir en la lista en el sitio que le corresponda “Cuenca”, que se almacenaría en la posición 748 de memoria. ¿Qué modificaciones habría que hacer en la lista?
- Suponga que deseamos cambiar el dato La Coruña por A Coruña, ¿qué modificaciones habría que hacer en la lista?

**P10.29.** Indicar el procedimiento para unir dos listas encadenadas LA y LB.

### PILAS

**P10.30.** Una pila inicialmente se encuentra vacía; si se realizan las operaciones que se indican a continuación, indicar los datos que se van extrayendo y el contenido final de la pila.

- |                |                |                |
|----------------|----------------|----------------|
| 1) Insertar 48 | 2) Insertar 54 | 3) Insertar 25 |
| 4) Extraer     | 5) Insertar 12 | 6) Insertar 18 |
| 7) Extraer     | 8) Insertar 15 | 9) Extraer     |

### COLAS

**P10.31.** Suponga una cola que se gestiona en la memoria principal utilizando un puntero de cabecera (*PCab*) y otro de final de cola (*Pcola*), que el valor inicial de estos punteros es  $PCab = H'4BFA$  y  $Pcola = H'4C00$ . ¿Cuáles son los nuevos valores de los punteros si se hacen 4 extracciones y 3 inserciones?

*Nota:* Suponer que cada dato de la cola ocupa una palabra de memoria.

**P10.32.** Suponga una cola circular que se gestiona en la memoria principal utilizando un puntero de cabecera (*PCab*) y otro de final de cola (*Pcola*) y que el valor inicial de estos punteros es  $PCab = H'4C0D$  y  $Pcola = H'4C0F$ . Si cada dato de la cola ocupa una palabra de memoria y las dirección primera y última del bloque de memoria donde se almacena la cola son  $di = H'4C00$  y  $df = H'4C0F$ , respectivamente:

- ¿Cuántos elementos contiene en ese momento la cola?
- ¿Cuáles son los nuevos valores de los punteros si se hacen 4 extracciones y 3 inserciones?

**P10.33.** Indicar la forma de conocer que:

- Una lista enlazada está vacía.
- Una pila está vacía.
- Una cola lineal está vacía.
- Una cola circular está llena o vacía.

### ÁRBOLES

**P10.34.** La tabla que se da a continuación (Tabla 10.8) representa un árbol almacenado en memoria con el dato del nodo seguido de tres punteros por nodo: al padre, al primer hijo y al siguiente hermano. Dibujar el árbol que representa.

*Nota:* FF representa valor nulo (FIN).

Dirección	Contenido
A0	p
A1	B8
A2	FF
A3	FF
A4	k
A5	C0
A6	FF
A7	FF
A8	j
A9	C0
AA	FF
AB	A4
AC	f
AD	B0
AE	FF
AF	B4
B0	d
B1	B8
B2	C0
B3	AC
B4	g
B5	B0
B6	FF
B7	FF
B8	c
B9	FF
BA	B0
BB	FF
BC	i
BD	C0
BE	FF
BF	A8
C0	e
C1	B8
C2	BC
C3	A0

**Tabla 10.8.** Descripción del árbol del Problema 10.35.

**P10.35.** Indicar, para el árbol de la Figura 10.19:

- a) Su orden.
- b) Su altura.
- c) El grado de cada uno de sus nodos.
- d) El nodo raíz.
- e) Los subárboles del nodo raíz.

**P10.36.** Dibujar un árbol de las comunidades autónomas españolas del Mediterráneo con las provincias que comprenden. Para este árbol, y suponiendo que los punteros ocupan 3 Bytes:

- Evaluar la ocupación de memoria que tendría si se representase por medio de una tabla, con punteros a padre e hijos.
- Evaluar la ocupación de memoria que tendría si se representase con 3 punteros/enlaces por nodo (padre, primer hijo y siguiente hermano).
- Realizar un esquema de la representación en memoria, con 3 enlaces por nodo.

**P10.37.** Dada la representación en memoria del árbol que se indica en la Tabla 10.9, dibujarlo.

Dirección	Contenido del nodo	Puntero al padre	Puntero al primer hijo	Puntero al siguiente hermano
1	Enrique	Nulo	2	Nulo
2	Manolo	1	14	4
3	Luz	2	Nulo	Nulo
4	Carmen	1	5	6
5	Lourdes	4	Nulo	15
6	Matías	1	7	10
7	Alberto	6	Nulo	8
8	Juan	6	Nulo	9
9	María	6	Nulo	Nulo
10	Marina	1	11	Nulo
11	Fermín	10	Nulo	12
12	Carlos	10	Nulo	13
13	Ramón	10	Nulo	Nulo
14	Ignacio	2	Nulo	3
15	Antonio	4	Nulo	16
16	Blanca	4	Nulo	17
17	Luís	4	Nulo	Nulo

**Tabla 10.9.**

**P10.38.** Dibujar todos los posibles árboles binarios con 5 nodos.

**P10.39.** Indicar la condición para detectar si un árbol almacenado en memoria con enlaces está vacío o no.



# Algoritmos

Según se ha indicado en lecciones anteriores, un computador sólo puede realizar las acciones u operaciones definidas en el lenguaje de programación que se utilice. En consecuencia, cualquier tratamiento de información que se desee desarrollar, por complejo que sea, debe descomponerse en las operaciones (relativamente sencillas) que es capaz de interpretar o compilar el lenguaje de programación. Si no se dispone de un traductor de lenguaje de alto nivel, estas operaciones son las definidas en el repertorio de instrucciones máquina. Esta lección se dedica al concepto de algoritmo, cuya aplicación hace posible la adaptación y descomposición de problemas para ser resueltos por el computador.

## 11.1. Concepto de algoritmo

Se puede definir un **algoritmo** como una secuencia de pasos a seguir, ordenados, no ambiguos y finitos que resuelven un problema.

Un algoritmo, por lo general, utiliza un conjunto de datos de entrada y proporciona unos datos de salida.

Según la definición de algoritmo se deben cumplir los siguientes requisitos:

- Cada paso tiene que ser *ejecutable*, es decir, que se pueda llevar a cabo.
- Los pasos a seguir tienen que estar formados por acciones bien definidas, es decir, *no ambiguas*. La información contenida en cada paso debe ser suficiente para determinar de forma única y completa las acciones a realizar.
- El algoritmo es un proceso que tiene que tener un *fin*, es decir, tiene que estar formado por una secuencia finita de operaciones, y terminar en un tiempo finito.

Un algoritmo es algo abstracto que se puede implementar de muchas formas. Es importante distinguir entre un algoritmo y su representación. Los algoritmos se representan mediante **programas**. A un programa en ejecución se le denomina **proceso**. Es decir, un programa es la representación de un algoritmo y un proceso es la ejecución del algoritmo.

No hay procedimientos rigurosos a la hora de construir un algoritmo, sino que es algo creativo, y como tal complejo. Existen normas generales y herramientas que ayudan a implementar algoritmos.

**EJEMPLO 11.1**

Diseñar un algoritmo para obtener el valor mayor ( $VM$ ) de una lista,  $L(i)$ , de  $N$  números.

**SOLUCIÓN****Máximo ( $L, N; VM$ )**

```
Hacer  $i = 1$  y  $VM = L(1)$ 
Mientras  $i < N$  repetir
    Hacer  $i = i + 1$ 
    Si  $L(i) > VM$ , hacer  $VM = L(i)$ 
Fin
```

Obsérvese que, para hacer más legible el algoritmo, en su inicio incluimos un nombre identificador (*Máximo*, en este caso) y entre paréntesis los nombres de los datos que necesita el algoritmo para resolver el problema (en este caso, la lista  $L$  y el número de sus elementos,  $N$ ), y separado por punto y coma (;) el resultado (en este caso  $VM$ ). Los parámetros o valores de entrada y salida del programa se suelen denominar **argumentos del algoritmo**. En el caso del Ejemplo 11.1 los argumentos son:  $L$ ,  $N$  y  $VM$ .

## 11.2. Representación de algoritmos

Existen diferentes métodos para representar un algoritmo, pero todos ellos tienen en común que utilizan alguna forma de lenguaje. Este lenguaje debe ser preciso y estar bien definido. En estos lenguajes se suelen utilizar unos elementos de construcción denominados **primitivas**. Cada primitiva tiene una sintaxis (una representación simbólica) y una semántica (un significado), y si se definen suficientemente bien luego el algoritmo se puede traducir fácilmente a un programa para computador.

Los procedimientos más habituales para representar un algoritmo consisten en utilizar pseudocódigo y organigramas, como se analiza a continuación.

### 11.2.1. PSEUDOCÓDIGO

El **pseudocódigo** es una forma de representar informalmente un algoritmo basándose en el lenguaje natural. Suele tener una estructura semejante a la de un programa pero sin seguir reglas sintácticas estrictas.

**EJEMPLO 11.2**

Diseñar un algoritmo para contar hasta 6.

**SOLUCIÓN****Contar hasta seis**

```
Hacer  $N = 0$ 
Mientras  $N < 6$ 
    Incrementar  $N$  en 1
    Escribir  $N$ 
Fin
```

**EJEMPLO 11.3**

Diseñar un algoritmo para calcular el factorial de un número entero  $n$ .

## SOLUCIÓN

**Factorial ( $n$ ; fact)**

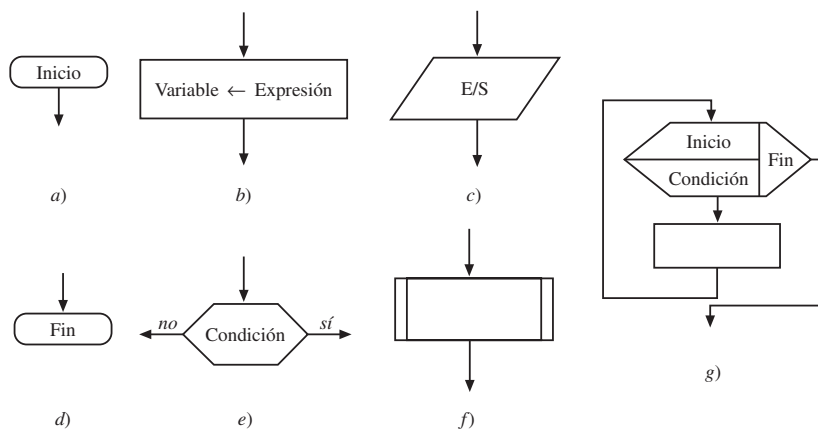
```

Leer n
fact = 1
i = 1
Mientras ( $i \leq n$ )
    fact = fact x i
    i = i + 1
Fin

```

**11.2.2. ORGANIGRAMAS**

Los **organigramas** o **diagramas de flujo** son herramientas gráficas para representar algoritmos. En este tipo de representación no necesariamente se dan los detalles de funcionamiento, sino los pasos a seguir de principio a fin. Está formado por una serie de símbolos que tienen al menos una flecha que viene del paso anterior y otra que va al paso siguiente. Los símbolos representan distintas acciones: asignación, lectura/escritura, decisiones, saltos, principio, fin, etc.

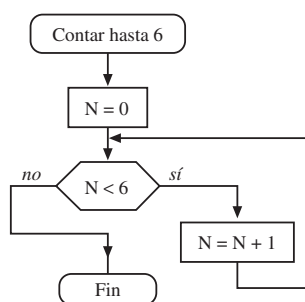


**Figura 11.1.** Símbolos utilizados para realizar organigramas: a) Inicio, b) Asignación, c) Entrada/Salida, d) Fin, e) Bifurcación, f) Llamada a procedimiento, g) Bucle.

**EJEMPLO 11.4**

Hacer el organigrama correspondiente al Ejemplo 11.2.

## SOLUCIÓN



**Figura 11.2.** Organigrama correspondiente al Ejemplo 11.2.



## 11.3. Estructuras de control

Cualquier programa se puede escribir con la ayuda de tres elementos o constructores (Figura 11.3): las *asignaciones*, las *decisiones*, y los *ciclos* que son elementos que permiten alterar el orden secuencial del programa; es decir, que controlan el flujo de las instrucciones.

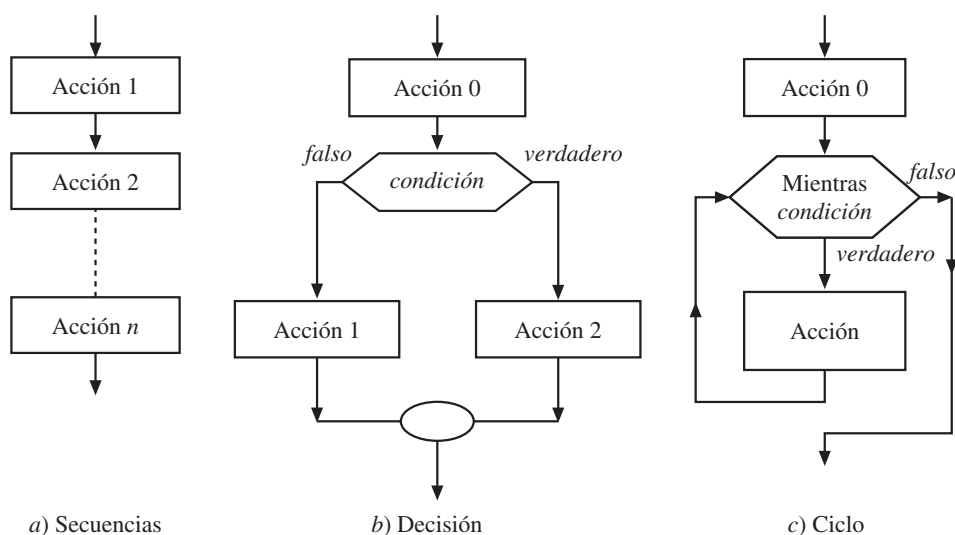
- **Secuencias:** una secuencia es una serie de instrucciones (Figura 11.3a) que pueden contener:
  - **Asignaciones.** Una asignación consiste en la evaluación de una expresión (aritmética, lógica, etc.) y almacenamiento de su valor en una variable.
  - **Entradas/Salidas.** Son operaciones que permiten intercambiar información con un dispositivo externo. En una operación de *entrada* o *lectura* (**leer teclado**, etc.) se asigna a una variable el valor almacenado en un soporte masivo o generado por un dispositivo de entrada. En una operación de *salida* o *escritura* (**imprimir**, **visualizar**, etc.) el valor de una variable se transfiere a un dispositivo de memoria masiva o a un dispositivo de salida.
- **Decisiones.** Son acciones de control de flujo. En una decisión se evalúa una expresión lógica y según su resultado se ejecutan una serie de acciones o actividades que están en uno u otro camino, es decir, son bifurcaciones (Figura 11.3b). En pseudocódigo las decisiones suelen especificarse de la siguiente forma:

```

si condición
    hacer acción 1
si no acción 2
  
```

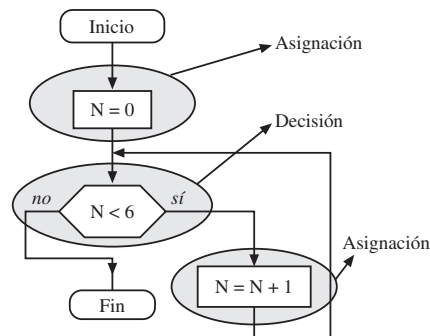
Donde **hacer** puede ser la realización de asignaciones o cálculos, una operación de E/S o un salto. Tradicionalmente esta estructura se ha denominado *If ... then ... else ...*. Muy frecuentemente no se incluye la opción «si no», sobreentendiéndose que si no se cumple la condición hay que continuar con el paso siguiente (el que viene a continuación).

- **Ciclos:** son estructuras de control que indican la repetición de una serie de instrucciones (**cuerpo del ciclo**) (Figura 11.3c).



**Figura 11.3.** Estructuras de control básicas.

En el Ejemplo 11.2 se tiene un ciclo iterativo que se repite 6 veces ( $N < 6$ ).



**Figura 11.4.** Constructores del Ejemplo 11.2.

Existen **ciclos iterativos** que son los que se repiten hasta que se alcance un determinado valor y **ciclos condicionales**, en los que se repiten las instrucciones mientras o hasta que se verifique una determinada condición (expresión lógica).

Los **ciclos iterativos** se controlan mediante tres parámetros: contador del ciclo, valor inicial del contador y valor final del contador, y que deben especificarse en el programa. En pseudocódigo los ciclos iterativos se suelen expresar como:

```

Para i=inicio hasta final repetir
    cuerpo del bucle (acción a repetir)
  
```

Donde en este caso hemos denominado *i*, *inicio* y *final* al contador del bucle, valor inicial del contador y valor final del contador, respectivamente.

La estructura anterior, a nivel de detalle, implica la realización de tres operaciones:

- Inicialización del contador (hacer  $i = inicio$ ).
- Incremento del contador (hacer  $i = i + 1$ ).
- Salto condicional o comprobación para ver si se ha llegado al final del bucle, y que puede realizarse al inicio o final del bucle, como se pone de manifiesto en el siguiente ejemplo.

## EJEMPLO 11.5

Se tiene una lista  $L(i)$  de  $N$  números y se desea averiguar las posiciones donde existan valores igual a 0. Describir el algoritmo de tres formas: *a)* con la estructura **Para...hasta...repetir**, *b)* con comprobación al inicio del bucle y *c)* con comprobación al final del bucle.

### SOLUCIÓN

*a)* Estructura Para...hasta...repetir (la de mayor contenido semántico Figura 11.5a):

```

Posiciones con ceros (versión 1)
Para  $i=1$  hasta  $N$  repetir
    Si  $L(i) = 0$  Escribir  $i$ 
Fin
  
```

b) Estructura con comprobación del final del bucle al inicio del mismo:

**Posiciones con ceros (versión 2)**

Paso 1:  $i=0$   
 Paso 2: Si  $i=N$  ir a Paso 6; comprobación de fin del bucle  
 Paso 3:  $i=i+1$   
 Paso 4: Si  $L(i)=0$  escribir  $i$   
 Paso 5: Ir a Paso 2  
 Paso 6: Fin

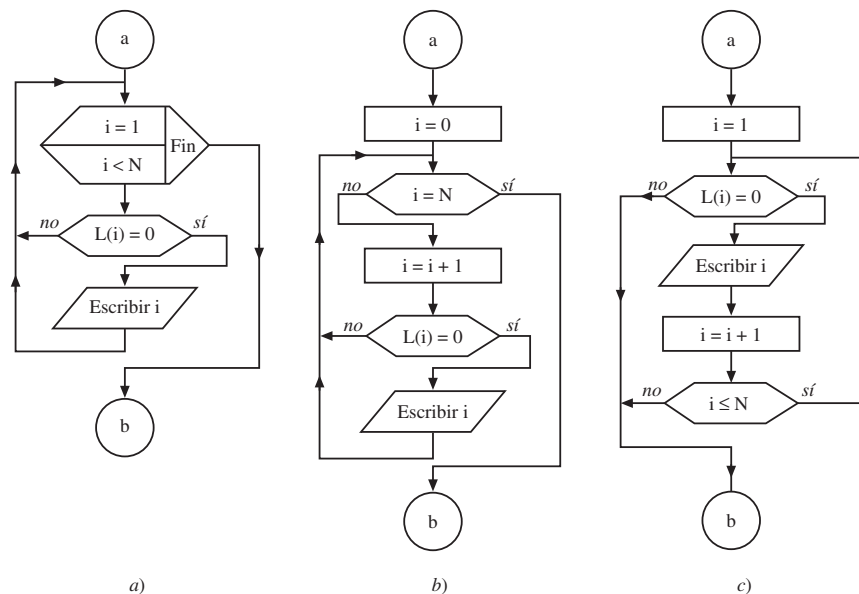
En la Figura 11.5b puede verse un organigrama de este programa.

c) Estructura con comprobación del final del bucle al final del mismo:

**Posiciones con ceros (versión 3)**

Paso 1:  $i=1$   
 Paso 2: Si  $L(i)=0$  escribir  $i$   
 Paso 3:  $i=i+1$   
 Paso 4: Si  $i \leq N$  ir a Paso 2; comprobación de fin de bucle  
 Paso 5: Fin

En la Figura 11.5c puede verse un organigrama de este programa.



**Figura 11.5.** Organigramas de los algoritmos de detección de ceros: a) con mayor contenido semántico y con comprobación del final del bucle; b) en el inicio, y c) al final.

En los **ciclos condicionales** también suelen utilizarse dos variantes o tipos de ciclos:

1. **Mientras condición repetir** (o **Ciclo while**).

Se repite el segmento de programa que se incluya a continuación mientras se cumpla la condición (que es una expresión lógica).

2. **Hasta que condición repetir** (o **Ciclo until**).

Se repite el lazo hasta el momento en que deje de cumplirse la condición.

**EJEMPLO 11.6**

Diseñar el algoritmo del Ejemplo 11.5 utilizando las estructuras de ciclo condicional.

**SOLUCIÓN**

a) Estructura *Mientras*:

```

Posiciones con ceros (versión 4)
i = 1
Mientras i ≤ N repetir
    Si L(i) = 0 Escribir i
    i = i + 1
Fin

```

b) Estructura *Hasta*:

```

Posiciones con ceros (versión 5)
i = 1
Hasta que i > N repetir
    Si L(i) = 0 Escribir i
    i = i + 1
Fin

```

En la Tabla 11.1 se resumen las estructuras analizadas.

<b>Secuencias</b>	Asignaciones Entradas/Salidas	i = i + 1 <b>leer, escribir, imprimir, visualizar, ...</b>
<b>Decisiones</b>		<b>Si condición hacer</b> acción 1 <b>si no</b> acción 2
<b>Ciclos</b>	Repetitivos <ul style="list-style-type: none"> <li>• Comprobación de fin al inicio del bucle</li> <li>• Comprobación de fin al final del bucle</li> </ul>	<b>Para</b> i = <i>inicio</i> <b>hasta</b> <i>final</i> <b>repetir</b> acción
	Condicionales <ul style="list-style-type: none"> <li>• Ciclos <i>While</i></li> <li>• Ciclos <i>Until</i></li> </ul>	<b>Mientras</b> <i>condición</i> <b>repetir</b> acción <b>Repetir</b> acción <b>hasta que</b> <i>condición</i>

**Tabla 11.1.** Constructores para describir un algoritmo.

En informática se utilizan algoritmos de distinta naturaleza, algunos de los cuales se plantean como problema de este capítulo. Entre los algoritmos básicos se encuentran los siguientes:

- Algoritmos de tratamiento de caracteres (ver, por ejemplo, Problema 11.9).
- Algoritmos de ordenación, entre los que destacan:
  - Ordenación por selección (Problemas 11.5 y 11.29).
  - Ordenación por inserción (Problema 11.27).
  - Ordenación binaria (Problema 11.10).
- Algoritmos de búsqueda, como por ejemplo los de:
  - Búsqueda secuencial (Problema 11.11).
  - Búsqueda binaria (Problema 11.12).
- Algoritmos aritméticos (Problemas 11.4, 11.13 y 11.14).

## 11.4. Subalgoritmos

Un subalgoritmo es una porción de algoritmo que realiza una tarea concreta, que tiene un nombre y que puede ser utilizado en distintas partes del algoritmo que lo utiliza. Es decir, un algoritmo se puede descomponer en subalgoritmos más sencillos, facilitando así el entendimiento y resolución del problema completo. A los subalgoritmos también se les suele conocer con los nombres de **subrutinas**, **rutinas** o **procedimientos**.

Utilizar subalgoritmos tiene ciertas ventajas:

- *Facilita la construcción y comprensión* de los programas, ya que es más sencillo escribir programas pequeños que lleven a cabo una tarea concreta.
- Al poder llamar a una subrutina siempre que se necesite, se evita tener que reescribir el código cada vez que se necesite realizar la tarea correspondiente; los subalgoritmos *reducen la duplicación de código*.

### EJEMPLO 11.7

Diseñar un algoritmo para calcular el número de combinaciones sin repetición de  $m$  símbolos distintos tomados de  $n$  en  $n$  veces.

*Nota:* Utilizar como subrutina el algoritmo de la obtención de factorial de un número considerado en el Ejemplo 11.3.

### SOLUCIÓN

Recuérdese que el número de combinaciones sin repetición de  $m$  símbolos tomados de  $k$  en  $k$  viene dado por:

$$\binom{m}{k} = \frac{m!}{k! \cdot (m - k)!}$$

Con lo cual un algoritmo podría ser el siguiente:

```

Combinaciones(m,k;C)
  Factorial(m;Fm)
  Factorial(k,Fk)
  Factorial(m-k,Fmk)
  C = Fm / (Fk · Fmk)
Fin de Combinaciones

```

Normalmente una subrutina utiliza unos parámetros o datos de entrada y genera uno o varios valores de salida. Como se indicó en la Sección 11.1, estos valores se pueden dar u obtener de la subrutina de forma explícita, indicándolos entre paréntesis, después del nombre de ella. Así, recordemos que el algoritmo de obtención del máximo lo denominábamos **Máximo(L,N;VM)**, donde  $L$  y  $N$  son parámetros de entrada y  $VM$  es el valor de salida (en este caso, el máximo de la lista  $L$  de  $N$  componentes). En el caso de la subrutina que calcula el factorial, la denominábamos **Factorial(n;fact)**. Otra posibilidad que admiten la mayoría de los lenguajes de programación es definir subrutinas que actúan como **funciones**. Cuando se llama a una función ella misma actúa como una variable que toma el valor de retorno de la subrutina. Así, por ejemplo, caso de definir el algoritmo del máximo como función, la expresión:

$$x = 58 + \text{máximo}(L, N)$$

asignaría a  $x$  el valor de 58 más el valor del máximo de la lista  $L$ .

**EJEMPLO 11.8**

Repetir el algoritmo del Ejemplo 11.7, considerando que se dispone de la función  $\text{Factorial}(n)$ , que proporciona el factorial del número entero  $n$ .

**SOLUCIÓN**

```

Combinaciones(m,k;c) (versión 2)
    c = Factorial(m) / [Factorial(k) · Factorial(m-k)]
Fin de Combinaciones

```

Obsérvese que el algoritmo  $\text{Combinaciones}(m,k;c)$ , a su vez, puede utilizarse por otro algoritmo como si fuese una subrutina. Es decir, una rutina puede llamar a otra, y esta última a otra, y así sucesivamente (ver Sección 4.2.2).

## 11.5. Recursividad

Un algoritmo es recursivo cuando aparece en la definición de él mismo. Es decir, cuando se llama a sí mismo para resolver una parte del mismo.

Un problema se puede resolver siempre de dos formas, recursivamente o utilizando ciclos, es decir, iterativamente. Muchas veces la forma natural de resolver un problema es la recursiva.

**EJEMPLO 11.9**

Realizar un programa que calcule el factorial de un número  $n$  de forma recursiva.

```

Factorial (n)
    Si n = 0, Factorial = 1
    Si no Factorial = n · Factorial (n - 1)
Fin

```

Obsérvese que el propio programa *Factorial* se llama a sí mismo, utilizando como argumento de entrada  $n - 1$ . Veamos, por ejemplo, cómo se ejecuta el algoritmo cuando  $n = 4$ .

1.  $\text{Factorial}(4) \rightarrow$  Como  $n = 4$ ; se llama a  $\text{Factorial}(3)$ .
2.  $\text{Factorial}(3) \rightarrow$  Como  $n = 3$ ; se llama a  $\text{Factorial}(2)$ .
3.  $\text{Factorial}(2) \rightarrow$  Como  $n = 2$ ; se llama a  $\text{Factorial}(1)$ .
4.  $\text{Factorial}(1) \rightarrow$  Como  $n = 1$ ; se llama a  $\text{Factorial}(0)$ .
5.  $\text{Factorial}(0) \rightarrow$  Como  $n = 0$ ; se hace  $\text{Factorial}(0) = 1$ , y se retorna a  $\text{Factorial}(1)$ .
6.  $\text{Factorial}(1) \rightarrow$  Como  $n \neq 0$ ; se hace  $\text{Factorial}(1) = 1 \cdot \text{Factorial}(0) = 1$ , y se retorna a  $\text{Factorial}(2)$ .
7.  $\text{Factorial}(2) \rightarrow$  Como  $n \neq 0$ ; se hace  $\text{Factorial}(2) = 2 \cdot \text{Factorial}(1) = 2$ , y se retorna a  $\text{Factorial}(3)$ .
8.  $\text{Factorial}(3) \rightarrow$  Como  $n \neq 0$ ; se hace  $\text{Factorial}(3) = 3 \cdot \text{Factorial}(2) = 6$ , y se retorna a  $\text{Factorial}(4)$ .
9.  $\text{Factorial}(4) \rightarrow$  Como  $n \neq 0$ ; se hace  $\text{Factorial}(4) = 4 \cdot \text{Factorial}(3) = 24$ , y concluye el programa.

## 11.6. Proceso de creación de un programa

Para desarrollar un programa en un lenguaje de alto nivel se suelen seguir los siguientes pasos:

- **Planteamiento del problema:** antes de buscar una solución hay que entender y conocer con todo detalle el problema, haciendo un planteamiento minucioso del mismo.
- **Análisis del problema:** se debe analizar y examinar con todo detalle el problema, explorando distintas soluciones del mismo. En esta etapa es muy importante determinar los datos que se necesitan. Conocidos los datos de entrada y cómo debe ser la salida hay que pensar en el formato más adecuado para representarlos; si no se escoge una representación adecuada, el algoritmo no será bueno.
- **Diseño del algoritmo** que resuelve el problema: esta fase está muy relacionada con la anterior e incluso a veces se pueden hacer ambas a la vez; es decir, mientras se va diseñando el algoritmo se va viendo qué tipo de representación es la más adecuada. Esta es una tarea creativa que se va complicando con la complejidad del problema a resolver. Una forma natural de desarrollar algoritmos es utilizando una técnica de **diseño descendente**, consistente en empezar a trabajar a nivel abstracto (muy global) para dividir el problema en partes más sencillas y concretas. Este proceso se va repitiendo hasta llegar a problemas suficientemente simples como para resolverlos directamente. Este tipo de diseño se adapta muy bien a la programación estructurada. Las técnicas de construcción o **diseño ascendente** actúan de forma inversa: parten de elementos o módulos físicos concretos y a partir de ellos, en pasos sucesivos de mayor complejidad, obtienen el sistema en su mayor nivel de abstracción.
- **Prueba:** el último paso en la creación de un programa es comprobar que es eficiente y que funciona correctamente. La verificación de un programa es muy importante y por tanto también lo es encontrar técnicas efectivas que la lleven a cabo. Un programa se puede probar haciendo un seguimiento del mismo; es decir, dando unos datos de entrada y comprobando instrucción a instrucción si se van generando valores correctos. Sin embargo, de esta forma sólo sabemos que el programa funciona correctamente en esos casos (con los datos de prueba seleccionados). En otras situaciones no tenemos la certeza absoluta de que funcione. Para evitar esto, se suelen utilizar técnicas de la lógica formal para probar que un programa es correcto, produciendo los resultados esperados. Por ejemplo:
  - Para comprobar que un ciclo funciona bien, se aplica una demostración por inducción matemática.
  - Si dos partes del programa son independientes se prueban por separado.
- **Optimización:** consiste en buscar otro algoritmo que resuelva mejor el problema, o en reemplazar partes del programa por otras más eficientes. Para ello habría que evaluar el programa con los distintos algoritmos, lo que en la mayoría de los casos es inviable por la complejidad de la evaluación.
- **Implementación del algoritmo**, que, como se analizará en el capítulo siguiente, consiste en:
  - Representar el algoritmo en un lenguaje de programación adecuado.
  - Traducir el programa a código máquina.
  - Ejecutarlo.
  - Depurarlo y probarlo.

## 11.7. Conclusiones

Cualquier problema o aplicación que deseemos resolver con ayuda de un computador debe descomponerse y expresarse en términos de las operaciones que es capaz de representar el lenguaje de programación que se esté utilizando; esta tarea se realiza utilizando el concepto de algoritmo.

En este capítulo se han analizado y estudiado los conceptos de algoritmo (Sección 11.1) y de su representación (Sección 11.2). Por otra parte se han descrito las estructuras de control (Sección 11.3), que permiten alterar el orden secuencial de ejecución de las instrucciones, y las posibilidades de los subalgoritmos (Sección 11.4) y de la recursividad (Sección 11.5) para desarrollar algoritmos. La exposición teórica del capítulo ha concluido con una sección (Sección 1.6) dedicada a describir el proceso de desarrollo de un programa.

## Test



**T11.1.** Un algoritmo, a diferencia de un programa:

- a) Puede ser indefinido.
- b) Sus pasos pueden ser ambiguos.
- c) Es abstracto.
- d) Los pasos no tienen por qué ser ejecutables.

**T11.2.** Los argumentos de un algoritmo:

- a) Son los parámetros o datos de entrada y de salida.
- b) Son las condiciones de las estructuras de decisión.
- c) Es el conjunto de estructuras de control que utiliza.
- d) Es el conjunto de medios de entrada y salida que utiliza (teclado, impresora, pantalla, etc.).

**T11.3.** Las primitivas de un algoritmo que sirven para evaluar una expresión son:

- a) Asignaciones.
- b) Entradas/salidas.
- c) Decisiones.
- d) Ciclos.

**T11.4.** Las primitivas de un algoritmo que sirven para intercambiar información con el exterior son:

- a) Asignaciones.
- b) Entradas/salidas.
- c) Decisiones.
- d) Ciclos.

**T11.5.** Las primitivas de un algoritmo que sirven para repetir acciones son:

- a) Asignaciones.
- b) Entradas/salidas.
- c) Decisiones.
- d) Ciclos.

**T11.6.** Las primitivas de un algoritmo que sirven para evaluar test o condiciones son:

- a) Asignaciones.
- b) Entradas/salidas.
- c) Decisiones.
- d) Ciclos.

**T11.7.** Las primitivas de un algoritmo que sirven para controlar el flujo del programa son:

- a) Tanto asignaciones como entradas/salidas.
- b) Decisiones.
- c) Ciclos.
- d) Tanto decisiones como ciclos.

**T11.8.** Las primitivas de un algoritmo que sirven para almacenar una variable son:

- a) Tanto asignaciones como entradas/salidas.
- b) Decisiones.
- c) Ciclos.
- d) Tanto decisiones como ciclos.

**T11.9.** La estructura “**Si condición hacer ... si no ...**” es:

- a) Una asignación.
- b) Una decisión.
- c) El inicio de un ciclo repetitivo.
- d) El inicio de un ciclo condicional.

**T11.10.** La estructura “**Para  $i=10$  hasta  $10$  IF repetir**” es:

- a) Una asignación.
- b) Una decisión.
- c) Un inicio de un ciclo repetitivo.
- d) El inicio de un ciclo condicional.

**T11.11.** La estructura “**Mientras condición repetir**” es:

- a) Una asignación.
- b) Una decisión.
- c) El inicio de un ciclo repetitivo.
- d) El inicio de un ciclo condicional.

**T11.12.** La estructura “**Hasta que condición repetir**” es:

- a) Una asignación.
- b) Una decisión.
- c) El inicio de un ciclo repetitivo.
- d) El inicio de un ciclo condicional.

**T11.13.** Una función en programación es:

- a) Una instrucción de asignación que se puede describir en una única línea.



- b) Una subrutina, cuya denominación coincide con la variable que almacena el resultado de su ejecución, pudiéndose utilizar la subrutina como una variable más.
- c) Una subrutina cuyo resultado es una única variable.
- d) Es cualquier expresión lógica que se incluye en un salto o ciclo condicional.

**T11.14.** Si un algoritmo (subrutina) se llama a sí mismo se dice que es:

- a) Iterativo.
- b) Redundante.

- c) Recursivo.
- d) Procedimental.

**T11.15.** La redacción (diseño) de un algoritmo debe de hacerse:

- a) Después del planteamiento del problema y de la elección de la representación de los datos.
- b) Antes de la elección de la representación de los datos.
- c) Al final del diseño descendente.
- d) Al inicio del diseño ascendente.



## Problemas resueltos

### CONCEPTO DE ALGORITMO

**P11.1.** Suponga el siguiente algoritmo para regar con una manguera las macetas de una terraza:

- Paso 1: Abrir grifo de la manguera.
- Paso 2: Posicionar la manguera encima de una maceta y regarla adecuadamente.
- Paso 3: Ir al Paso 2.

¿Por qué no es correcto el algoritmo anterior? Corríjalo hasta que sea correcto.

#### SOLUCIÓN

No es correcto debido a que existe una ambigüedad en cuanto a las macetas que hay que regar y no está definida la finalización del algoritmo.

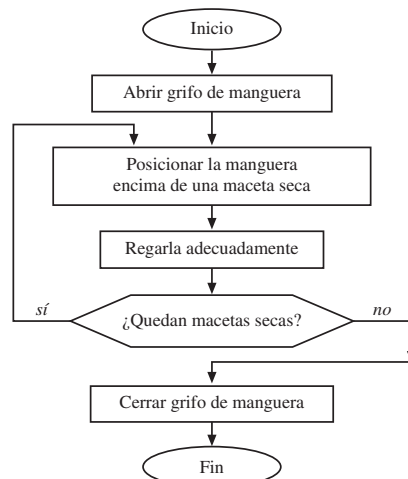
Un algoritmo correcto podría ser el siguiente:

- Paso 1: Abrir grifo de la manguera.
- Paso 2: Posicionar la manguera encima de una maceta *seca* y regarla adecuadamente.
- Paso 3: Si quedan macetas secas ir al Paso 2.
- Paso 4: Cerrar el grifo de la manguera (Fin del algoritmo).

### REPRESENTACIÓN DE ALGORITMOS

**P11.2.** Hacer el organigrama del algoritmo descrito en el Problema 11.2.

#### SOLUCIÓN



**Figura 11.6.** Algoritmo para regar las macetas de una terraza.

## ESTRUCTURAS DE CONTROL

**P11.3.** Suponiendo un lenguaje de programación en que las bifurcaciones únicamente se realizan con instrucciones cuya semántica es:

*si se verifica condición saltar a ..., si no continuar*

indicar cómo se implementarían las estructuras de ciclos. Suponer que el contador de ciclo se denomina *i*, el valor inicial *I0*, el valor final *IF*, y el conjunto de operaciones a realizar dentro del ciclo *cuerpo\_ciclo*.

## SOLUCIÓN

A. Estructura de ciclo repetitivo:

```
Para i=I0 hasta IF repetir
    cuerpo-ciclo
Fin
```

Implementaciones de bajo nivel:

a) Con comprobación del final del bucle al inicio del mismo.

**Implementación de bajo nivel de un ciclo repetitivo**

```
Paso 1: i = I0 - 1
Paso 2: Si i = IF ir a Paso 6 si no continuar
Paso 3: i = i + 1
Paso 4: cuerpo-ciclo
Paso 5: Ir a Paso 2
Paso 6: Fin
```

b) Con comprobación del final del bucle al final del mismo.

**Implementación de bajo nivel de un ciclo repetitivo (versión 2)**

```
Paso 1: i = I0
Paso 2: cuerpo-ciclo
Paso 3: i = i + 1
Paso 4: Si i ≤ IF ir a Paso 2, si no continuar
Paso 5: Fin
```

B. Estructura de ciclo condicional Mientras:

```
i = I0
Mientras i ≤ IF repetir
    cuerpo-ciclo
    i=i+1
Fin
```

**Implementación de bajo nivel de un Mientras**

```
Paso 1 i = I0; IFF = IF + 1
Paso 2 Si i = IFF saltar a Paso 6
Paso 4 cuerpo-ciclo
Paso 3 i = i + 1
Paso 5 Ir a Paso 2
Paso 6 Fin
```

C. Estructura de ciclo condicional Hasta que:

```
i = I0
Hasta que i > IF repetir
    cuerpo-ciclo
    i=i+1
Fin
```

**Implementación de bajo nivel de un Hasta que**

Paso 1  $i = I0$ ;  $IFF = IF + 1$   
 Paso 2 *cuerpo-ciclo*  
 Paso 3  $i = i + 1$   
 Paso 4 Si  $i < IFF$  saltar a Paso 2 si no continuar  
 Paso 5 Fin

Obsérvese que:

- A bajo nivel las distintas estructuras dan lugar a algoritmos muy similares
- Se obtiene menor número de instrucciones y mayor velocidad haciendo la comprobación del final del bucle al final del mismo.
- Estas implementaciones son las que se harían con CODE-2.

**P11.4.** Describir por medio de pseudocódigo un algoritmo para comprobar si dos arrays  $A(1:N, 1:M)$  y  $B(1:N, 1:M)$  son iguales o no. Después de desarrollar el algoritmo, identificar los distintos tipos de primitivas utilizadas (asignaciones, E/S, decisiones y ciclos).

**SOLUCIÓN**

```

i=0
j=0
Hasta que i=N repetir
    i=i+1
    Hasta que j=N repetir
        j=j+1
        Si  $A(i, j) \neq B(i, j)$  hacer
            Escribir "Arrays distintos"
        Fin
    Escribir "Arrays iguales"
Fin

```

Obsérvese que este algoritmo finaliza en cuanto encuentra dos elementos de los arrays distintos.

**SUBROUTINAS**

**P11.5. Ordenación por selección con dos listas.** Hacer un algoritmo que a partir de una lista,  $LO$ , de  $N$  números genere otra nueva,  $LN$  con los mismos números pero ordenados de mayor a menor, suponiendo que se dispone de una subrutina Máximo ( $L, N$ ), que a partir de una lista,  $L$ , nos proporciona el valor máximo de ella, y que en la lista original no hay ningún número menor que  $-1.000$ .

**SOLUCIÓN**

La idea básica consiste en buscar en la lista original,  $LO$ , su valor máximo, y, una vez encontrado: 1) llevarlo a la lista ordenada  $LN$  y 2) hacerlo igual a  $-1.000$  en la lista original, para poder encontrar el siguiente máximo. Un algoritmo que implementa esta idea es el siguiente

**Ordenación con dos listas**

```

Para i=1 hasta N repetir
    LN(i) = Máximo(LO, N)
    LO(i) = -1000
Fin

```

Obsérvese que con este algoritmo se destruye la lista original,  $LO$  (todos sus elementos acaban siendo  $-1.000$ ), por lo que, en caso de tener que conservarse, al inicio del algoritmo se debería hacer una copia de ella en otra lista, y realizar la búsqueda de los máximos sucesivos en esta última.

Por otra parte, si no conociésemos el valor mínimo de los elementos de la lista  $LO$ , y dispusiésemos de una función Mínimo( $L, N$ ), podríamos modificar el algoritmo de la siguiente forma:

**Ordenación con dos listas (versión 2)**

```

VM = Mínimo(LO, N) - 1
Para i=1 hasta N repetir
    LN(i) = Máximo(LO, N)
    LO(i) = VM
Fin

```

## RECURSIVIDAD

**P11.6.** Idear un algoritmo recursivo para escribir en sentido inverso los números de una lista,  $L$ .

## SOLUCIÓN

Una subrutina para invertir la lista de forma recursiva es la siguiente

```

Invertir
hacer  $i=i+1$ 
si  $i \neq N$ 
    Invertir
Escribir  $L(i)$ 
Fin de invertir (retornar al programa de llamada)
  
```

Para comprobar su funcionamiento supongamos que  $N = 3$ , e inicialmente  $i = 0$ . El funcionamiento, paso a paso, del algoritmo es el siguiente:

- Primer paso:  $i = i + 1 = 1$ , como  $i \neq 3$  se vuelve a llamar a Invertir.
- Segundo paso:  $i = i + 1 = 2$ , como  $i \neq 3$  se vuelve a llamar a Invertir.
- Tercer paso:  $i = i + 1 = 3$ , como  $i = 3$  se escribe  $L(3)$  y se retorna al punto donde se llamo por última vez a Invertir.
- Cuarto paso: Se ejecuta la instrucción inmediatamente después de la llamada a Invertir que se hizo en el Segundo paso; esto es, **escribir**  $L(i)$ , como en ese caso  $i = 2$ , se escribe  $L(2)$ . Al ejecutar Fin de invertir se retorna al punto donde se llamó a Invertir, que en esta ocasión fue en el Primer paso.
- Quinto paso: Se ejecuta la instrucción inmediatamente después de la llamada a Invertir que se hizo en el Primer paso; esto es, **escribir**  $L(i)$ , como en ese caso  $i = 1$ , se escribe  $L(1)$ .

La subrutina anterior requiere un algoritmo principal de llamada, que inicialice el valor de  $i$  a 0, y que haga concluir satisfactoriamente el algoritmo. Este algoritmo puede ser el siguiente:

```

Invertir lista
Hacer  $i=0$ 
Invertir
Fin
  
```

## ALGORITMOS BÁSICOS

**P11.7.** Diseñar un algoritmo que genere todas las combinaciones posibles, sin repetición, de los 3 símbolos que se encuentran como elementos de una tabla  $T$ . Comprobar paso a paso el funcionamiento del algoritmo.

## SOLUCIÓN

## Combinaciones de tres elementos

```

Para  $i = 1$  hasta 3 repetir
    Para  $j = 1$  hasta 3 repetir
        Para  $k = 1$  repetir
            Si  $i \neq j$  y  $j \neq k$ 
                Escribir  $T(i), T(j), T(k)$ 
        Fin
    Fin
Fin
  
```

*Comprobación del algoritmo:*

Suponemos que  $T = (a, b, c)$ ; es decir,  $T(1) = a$ ;  $T(2) = b$  y  $T(3) = c$ .

El comportamiento del algoritmo puede hacerse con ayuda de la tabla que se muestra a continuación, donde se incluyen, de izquierda a derecha, los valores de  $i, j, k$  en el orden en que se van generando en los tres ciclos, si se cumple o no la condición de la sentencia de decisión, y los valores que se van escribiendo ( $T(i), T(j), T(k)$ ) en cada caso.

$i$	$j$	$k$	$i \neq j$ y $j \neq k$	Combinación que se escribe	$i$	$j$	$k$	$i \neq j$ y $j \neq k$	Combinación que se escribe
1	1	1	no		2	1	2	no	
1	1	2	no		2	1	3	sí	bac
1	1	3	no		2	2	1	no	
1	2	1	no		2	2	2	no	
1	2	2	no		2	2	3	no	
1	2	3	sí	abc	2	3	1	sí	bca
1	3	1	no		2	3	2	no	
1	3	2	sí	acb	2	3	3	no	
1	3	3	no		3	1	1	no	
2	1	1	no						

Etcétera.

- P11.8.** Diseñar dos algoritmos para determinar si un número entero,  $X$ , es o no par. El primero de ellos comprobando si  $X$  se puede obtener como suma de doses, y el segundo comprobando si el número es el doble de la parte entera de su mitad. ¿Cuál de los dos algoritmos es más rápido?

SOLUCIÓN

```

X_par (versión 1)
Leer X
Mientras X > 0 repetir
    Si X = 1, Escribir "X es impar"
    Si X = 2, Escribir "X es par"
    Hacer X = X - 2
Fin

X_par (versión 2)
Leer X
Hacer Y = 2 * int(X/2);      int es una función que proporciona la parte entera
Si Y = X, Escribir "X es par"
    Si no, Escribir "X es impar"
Fin

```

- P11.9.** Realizar un algoritmo para sustituir las letras mayúsculas por minúsculas que hubiese en un texto que se encuentra en un array  $T$ , y cuyo último carácter es  $ETX$ . Cada elemento del array contiene un carácter Unicode.

SOLUCIÓN

Observando la tabla de caracteres Unicode (Apéndice 2) comprobamos que las letras minúsculas tienen los códigos del H'0061 (para la "a") al H'007A (para la "z"). Por otra parte las letras mayúsculas están comprendidas entre los códigos H'0041 (para la "A") al H'005A (para la "Z").

En consecuencia podemos comprobar si un carácter es mayúsculas comprobando si su valor está entre los límites H'0041 = D'65 y H'005A = D'90; caso de que así sea podemos pasarlo a minúsculas sin más que sumarle H'20 = D'32.

Un algoritmo puede ser el siguiente:

```

Mayúsculas a minúsculas
Hacer i = 1
Mientras T(i) ≠ ETX
    Si T(i) < 91 y T(i) > 64
        hacer T(i) = T(i) + 32
    Hacer i = i + 1
Fin

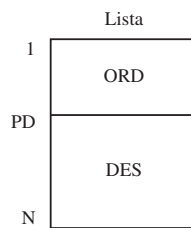
```

Obsérvese que aquí la condición lógica de la decisión consiste en que se cumplan simultáneamente dos condiciones:  $T(i) < 91$  y  $T(i) > 64$ ; donde el término "y" corresponde a la operación lógica de producto lógico (intersección).

**P11.10. Ordenación por burbuja.** Se dispone de una lista,  $L$ , de  $N$  elementos. Realizar un algoritmo para su ordenación de menor a mayor, siguiendo el procedimiento que se describe a continuación. El algoritmo empieza comparando el último elemento de la tabla con el penúltimo, si éste es mayor que el primero se intercambian, sino quedan como están. A continuación se comparan el de posición  $N-1$  con el de posición  $N-2$ , y, como antes, si este último ( $N-2$ ) es mayor que el siguiente ( $N-1$ ) se intercambian, y así sucesivamente hasta llegar al comienzo de la tabla. A continuación el proceso debe repetirse desde el final de la tabla. Puede observarse que poco a poco se van obteniendo en la primera parte de la tabla los números ordenados, y los procesos de comparación deben realizarse sólo en la segunda parte de la tabla, que cada vez será menor. Todo ocurre como si los elementos más pequeños de la última parte de la tabla fuesen subiendo poco a poco (como burbujas) hacia la primera parte.

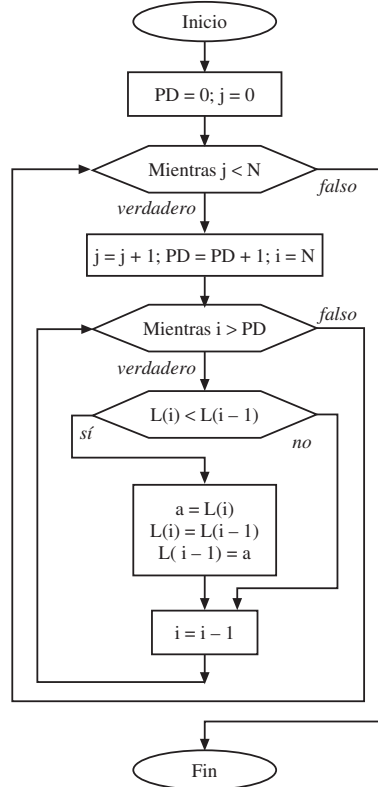
#### SOLUCIÓN

Denominamos  $PD$  a la primera posición de la parte desordenada de la lista; es decir, la parte ordenada va de la posición 1 a la  $PD-1$ , y la desordenada de la  $PD$  a la  $N$  (Figura 11.7).



**Figura 11.7.** Ordenación por el método de la burbuja.

Con esta definición el organigrama del algoritmo es el que se muestra en la Figura 11.8.



**Figura 11.8.** Algoritmo de ordenación por el método de las burbujas.

- P11.11. Búsqueda secuencial.** Realizar un algoritmo para una subrutina *BUSCADENA(T,C;PC)* que busque la primera vez que aparece un carácter, *C*, en un array lineal, *T*, que contiene un texto cuyo último elemento es el carácter de control *ETX*. La subrutina debe proporcionar la posición de la cadena, *PC*, en el array. Suponer que cada elemento del array contiene un carácter Unicode.

#### SOLUCIÓN

Llamamos *i* al puntero que recorre los elementos del array que contiene el texto. Un algoritmo posible es el siguiente:

```

Búsqueda de un carácter
Hacer i = 1
Mientras T(i) ≠ ETX
    Si C = T(i)
        Hacer PC = i
        Fin de la búsqueda
    Hacer i = i + 1
Escribir: "No existe en el array el carácter buscado"
Fin

```

Puede observarse que la ejecución de este algoritmo puede concluir en dos circunstancias: o bien cuando se encuentra el carácter buscado, o bien si se llega al final de la tabla, sin que el mismo aparezca.

- P11.12. Búsqueda binaria.** Se dispone de un array *TI(1:N,1:2)* con una tabla de índices, en el que cada fila tiene dos elementos, el primero un DNI, y el segundo un número que identifica la posición, *POS*, en un disco magnético a partir de la cual se encuentra la información que corresponde a la persona correspondiente. El array está ordenado de menor a mayor por DNI. Dado un número de DNI, se trata de encontrar la posición, *POS\_buscada*, en el disco asociada a ese DNI. Para realizar esta búsqueda es más eficiente utilizar una búsqueda binaria en lugar de una búsqueda secuencial. La búsqueda binaria consiste en acceder al DNI, que está en la mitad de la tabla; si el buscado es menor que el DNI al que se ha accedido, obviamente lo que buscamos está en la primera mitad de la tabla, por lo que podemos descartar en búsquedas sucesivas la otra mitad. Ahora accedemos a DNI, que está en medio de la parte no descartada (primera mitad de la tabla), y repetimos el proceso, hasta encontrar el DNI seleccionado. De la misma forma si el DNI buscado fuese mayor que el que se accede, se buscaría en la segunda parte, descartándose la primera. Desarrollar un algoritmo que resuelva el problema planteado mediante una búsqueda binaria.

#### SOLUCIÓN

Para describir el algoritmo vamos a utilizar tres parámetros:

- *IB*, que será la dirección del elemento de la tabla que se va a comprobar si coincide con el DNI buscado.
- *LIB*, es la dirección del límite inferior del tramo de búsqueda en la tabla.
- *LSB*, es la dirección del límite superior del tramo de búsqueda en la tabla.

Obviamente, en el primer paso la zona de búsqueda será toda la tabla, por lo que habrá que inicializar *LIB* a 1 y *LSB* a *N*.

Un algoritmo es el siguiente:

```

Búsqueda binaria
Leer DNI
Hacer LIB = 1; LSB = N; IB = 1
Mientras DNI ≠ TI(IB,1)
    Hacer IB = entero  $\left( \frac{LIB + LSB}{2} \right)$ ; dirección del centro de la zona de búsqueda
    Si DNI < TI(IB,1)
        Hacer LSB = TI(IB,1)
    Si DNI > TI(IB,1)
        Hacer LIB = TI(IB,1)
Escribir T(IB,2)
Fin

```

Obsérvese que como la dirección de busca de la tabla ha de ser entera, hacemos la *división entera* de la suma de los límites de la tabla entre 2.

## OTROS ALGORITMOS

- P11.13.** Desarrollar un algoritmo para resolver ecuaciones de segundo grado, utilizando como sistema de representación un organigrama. El algoritmo debe leer los tres coeficientes del polinomio ( $a$ ,  $b$  y  $c$ ) y proporcionar sus raíces.

*Nota:* Suponer que se dispone de una rutina *RAIZ\_2(x)* que proporciona la raíz cuadrada de un número,  $x$ .

### SOLUCIÓN

Recordemos que la solución de una ecuación de segundo grado de la forma:

$$ax^2 + bx + c = 0$$

es

$$x_1, x_2 = \frac{-b \pm \sqrt{D}}{2a}$$

donde  $D$  es el discriminante:

$$D = b^2 - 4ac$$

El algoritmo deberá tener en cuenta los tres casos posibles de soluciones: una única solución (caso de ser  $D = 0$ ), dos soluciones reales (si  $D > 0$ ) y soluciones imaginarias (si  $D < 0$ ).

Un algoritmo posible es el que se muestra en la Figura 11.9.

- P11.14.** Desarrollar un algoritmo para determinar el precio del billete de ida y vuelta en una compañía aérea en la que el precio por hora de vuelo es de 150 €, teniendo una reducción del 25 por 100 los vuelos de más de dos horas y media, y si entre la ida y la vuelta es mayor de 6 días se aplica la tarifa PEX (una reducción suplementaria del 55 por 100).

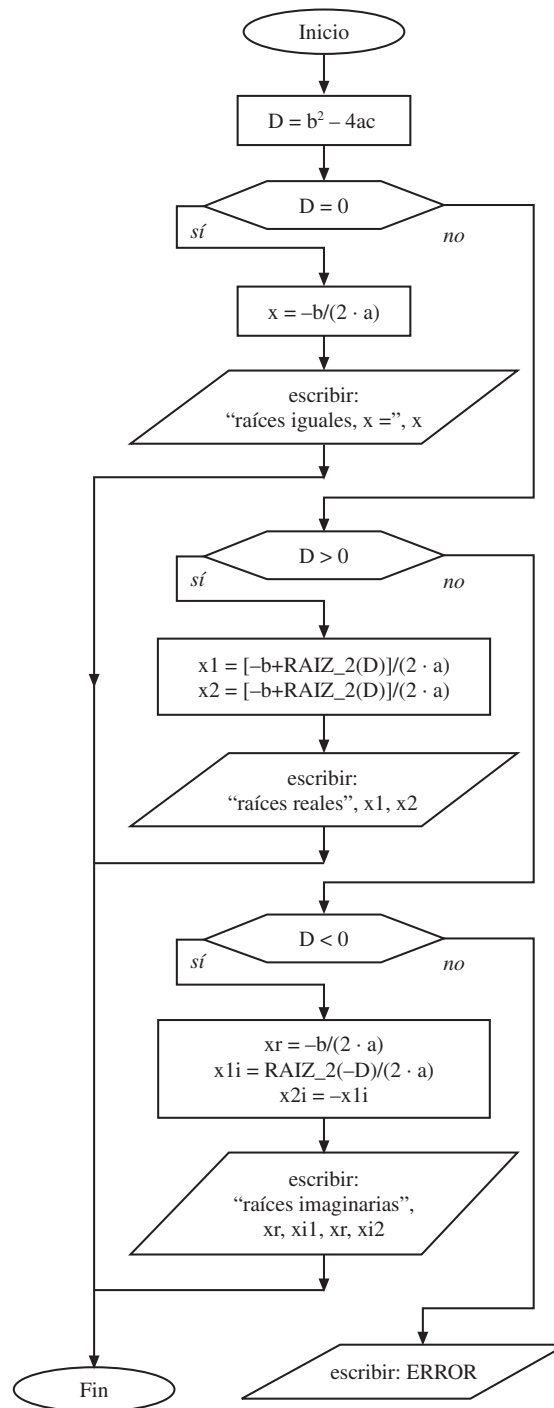
*Sugerencias:* El algoritmo debe leer las duraciones del vuelo y de la estancia, y proporcionar el precio del billete.

### SOLUCIÓN

En pseudocódigo un algoritmo que resolviese el problema podría ser el siguiente:

```
Precio billete
Leer duración_vuelo y duración_estancia;
Hacer precio = duración_vuelo · 150;
Si duración_estancia > 6 hacer precio=precio - 0,25 · precio
Escribir precio
Fin
```





**Figura 11.9.** Algoritmo para obtener las raíces de una ecuación de segundo grado.

## Problemas propuestos



## CONCEPTO DE ALGORITMO

**P11.15.** Considere la siguiente lista de números:

0; 8; -7; 5; 14; 3; -15

Obtener el máximo de ellos aplicando paso a paso el algoritmo del Ejemplo 11.1.

## REPRESENTACIÓN DE ALGORITMOS

**P11.16.** Hacer el organigrama del algoritmo descrito en el Ejemplo 11.1.

## ESTRUCTURAS DE CONTROL

**P11.17.** Realizar un algoritmo que cuente los números pares hasta 100, utilizando las siguientes estructuras de control:

- a) Ciclo repetitivo.
- b) Ciclo condicional *mientras*.
- c) Ciclo condicional *hasta que*.

**P11.18.** Suponiendo un lenguaje de programación en que las bifurcaciones únicamente se realizan con saltos incondicionales y condicionales cuya semántica, respectivamente, es:

*saltar a*

*si se verifica condición saltar a, si no continuar*

indicar cómo se implementaría, con el menor número posible de instrucciones, el ciclo repetitivo siguiente:

**Para**  $i=I_0$  a  $IF$  de  $K$  en  $K$  **repetir**  
     *cuerpo-ciclo*  
**Continuar**

*Nota:* Obsérvese que  $i$  varía de la siguiente forma:  $I_0, I_0 + K, I_0 + 2K, \dots$ , sin llegar a sobrepasar a  $IF$ .

**P11.19.** Diseñe un algoritmo que visualice la suma de los números enteros pares iguales o menores que  $N$ . Después de desarrollar el algoritmo, identifique los distintos tipos de primitivas utilizadas (asignaciones, E/S, decisiones y ciclos).

**P11.20.** Se tiene un array lineal  $L(i)$  de  $N$  números. Describir por medio de un organigrama un algoritmo para obtener el valor mínimo de dicha lista. Realizar cuatro versiones del algoritmo utilizando las estructura *mientras* (*while*), *hasta que* (*until*), y ciclos repetitivos con comprobación al inicio y al final. Después de desarrollar los algoritmos, identificar los distintos tipos de primitivas utilizadas (asignaciones, E/S, decisiones y ciclos).

## SUBROUTINAS

**P11.21.** Determinar el valor medio de los números pares y de los números impares de un conjunto de 500 datos que debe

leer el algoritmo. *Sugerencias:* Utilizar y realizar un procedimiento independiente para determinar si un número es par.

## RECURSIVIDAD

**P11.22.** Diseñar un algoritmo recursivo tal que dado un número  $N$  escriba  $N, N-1, N-2, \dots, 1$ .

## ALGORITMOS BÁSICOS

**P11.23.** Desarrollar un algoritmo que, a partir de una cantidad *Precio*, en euros, encuentre la descomposición que requiera menos unidades, en monedas de 1, 2, 20, 50 y 1 euro y en billetes de 5, 10, 20, 50, 100 y 200 euros.

**P11.24.** Hacer una subrutina que transforme una calificación numérica, *NotaN* (comprendida de 0 a 10), en su correspondiente literal, *NotaL*; donde:

- Si *NotaN* está comprendida entre 0 y 4,9; *NotaL* es *Suspenso*.
- Si *NotaN* está comprendida entre 5 y 7,4; *NotaL* es *Aprobado*.
- Si *NotaN* está comprendida entre 7,5 y 8,4; *NotaL* es *Notable*.
- Si *NotaN* está comprendida entre 8,5 y 9,9; *NotaL* es *Sobresaliente*.
- Si *NotaN* es 10; *NotaL* es *Matrícula de Honor*.

**P11.25.** Realizar un algoritmo para separar las palabras que hubiese en un texto que se encuentra en un array  $T$ , y cuyo último carácter es *ETX*. Cada elemento del array contiene un carácter Unicode; y las palabras obtenidas deben visualizarse una a una en la pantalla del computador.

**P11.26. Ordenación por inserción.** Se dispone de una lista,  $L$ , de  $N$  elementos. Realizar un algoritmo que la ordene de menor a mayor, siguiendo el procedimiento que se describe a continuación. Este algoritmo es muy similar al que se suele utilizar para ordenar manualmente fichas, cartas, documentos, etc. Consideramos la tabla dividida en dos zonas: una primera,  $PO$ , ordenada, y otra,  $PD$ , desordenada. Inicialmente  $PO$  sólo contiene el primer elemento de la tabla, y  $PD$  los  $N-1$  elementos restantes. A continuación consideramos el primer elemento de la parte desordenada, y lo ubicamos en el sitio que le corresponda en la parte ordenada para lograr el orden; es decir, delante del primer elemento (si es menor que él) o después (si es mayor que él). De esta forma la parte ordenada a aumentado en uno y la desordenada ha disminuido. A continuación seleccionamos el primer elemento de la nueva  $PD$ , y lo insertamos donde corresponda en  $PO$ ; y así sucesivamente.

**P11.27. Ordenación por selección.** Se dispone de una lista,  $L$ , de  $N$  elementos. Realizar un algoritmo que la ordene de me-

nor a mayor, siguiendo el procedimiento que se describe a continuación. Primero encontrar el elemento más pequeño de la tabla, y sustituirlo por el primero; después considerar sólo del 2.º al último elemento de la tabla, buscar aquí el menor número, y sustituirlo por el 2.º elemento, y así sucesivamente.

**P11.28. Búsqueda secuencial.** Realizar un algoritmo para una subrutina  $BUSCADENA(T, C; PC, LC)$  que busque una cadena de caracteres,  $C$ , en un array  $T$ , y cuyos últimos caracteres son  $ETX$ . La subrutina debe proporcionar la posición de la cadena,  $PC$ , en el array y la longitud de la misma,  $LC$ . Suponer que cada elemento del array contiene un carácter Unicode.

**P11.29.** Suponga que un determinado procesador de lenguajes dispone de las siguientes funciones para operar con cadenas de caracteres:

- **Concatenar:**  $CON(a, b)$  concatena  $a$  y  $b$ .
- **Extraer** una subcadena de una cadena.  $EXT(n: m)$  extrae de los caracteres  $n$  a  $m$ , inclusive.
- **Comparar** dos cadenas:  $COM(a, b)$  proporciona el valor 0 si  $a = b$ ; 1 si  $a < b$  y 2 si  $a > b$ .
- **Obtener la longitud,**  $LON(a)$  proporciona el número de caracteres que contiene la cadena  $a$ .
- **Identificar subcadena:**  $IDEN(a, b)$  proporciona la posición a partir de la cual se encuentra la subcadena  $b$  en la cadena  $a$ . Si el resultado es 0, indica que no existe dicha cadena en  $a$ .

Desarrollar un algoritmo-subrutina para realizar las funciones de **IDEN** utilizando las otras funciones.

**P11.30.** En los computadores que no disponen de procesador aritmético la raíz cuadrada de un número se obtiene por medio de un programa. Una forma de hacerlo es conociendo que la raíz cuadrada de un número  $X$  se puede calcular considerando como primer valor el propio número ( $X$ ) y generando uno nuevo a partir del previo haciendo la media del valor previo con el cociente entre el número inicial y el valor previo.

- Comprobar que el procedimiento es correcto, considerando, por ejemplo,  $X = 5$ .
- Diseñar un algoritmo para implementar el procedimiento anterior.

**P11.31.** Diseñar un algoritmo para incluir en una tabla,  $T$ , los  $M$  primeros números de Fibonacci. La sucesión de Fibonacci se define con la expresión analítica siguiente:

$$F(i) = F(i - 1) + F(i - 2)$$

siendo  $F(0) = 0$  y  $F(1) = 1$ .

## PROCESO DE CREACIÓN DE UN PROGRAMA

**P11.32.** Indicar si los procesos de diseño y de construcción de un coche utilizan la técnica descendente o ascendente.

**P11.33.** Se desea calcular el valor del número  $\pi$  con un computador, siguiendo las orientaciones que se dan a continuación.

Se supone que el lenguaje de programación a utilizar no dispone de ninguna función trigonométrica, pero sí de las operaciones aritméticas básicas. La idea principal consiste en aplicar el hecho de que  $\pi$  es cuatro veces la integral en el intervalo  $[0, 1]$  de la derivada del arco tangente, según se deduce de las expresiones siguientes:

$$\begin{aligned} \arctg'(x) &= \frac{1}{1+x^2} \\ \arctg(1) &= \frac{\pi}{4}, \quad \arctg(0) = 0 \\ \Rightarrow \int_0^1 \frac{1}{1+x^2} dx &= \arctg(x) \Big|_0^1 = \frac{\pi}{4} - 0 \end{aligned}$$

Con lo que  $\pi$  se puede obtener aplicando la siguiente expresión:

$$\pi = 4 \cdot \int_0^1 \frac{1}{1+x^2} dx$$

La integración se debe realizar de forma numérica, para lo cual se divide el intervalo  $[0, 1]$  en subintervalos, se calcula el área de la función en cada subintervalo y se suman las áreas obtenidas. El área en cada subintervalo se aproxima, por ejemplo, mediante el área de un rectángulo, el área de un trapecio o utilizando tres puntos de interpolación: los extremos del intervalo y el punto medio (fórmula de Simpson). En nuestro caso, dividir el intervalo  $[0, 1]$  en 10 subintervalos:  $x = 0; 0,1; 0,2; \dots 1,0$ ; y considerar cada uno de ellos como base de un rectángulo cuya altura es el valor de la función  $1/(1+x^2)$  en el valor de  $x$  inicio del intervalo.

- Desarrollar un algoritmo para obtener el valor de  $\pi$ , de acuerdo con las sugerencias anteriores.
- Indicar cómo se disminuiría el error del valor obtenido para  $\pi$  y cómo afectaría: 1) a la descripción del algoritmo y 2) al tiempo de ejecución.

# Lenguajes de programación

Un procesador de un computador únicamente puede interpretar y ejecutar programas escritos en lenguaje (código) máquina. Este lenguaje depende del procesador de que se trate y está muy alejado de los distintos lenguajes utilizados en los distintos dominios de aplicación de la informática (matemáticas, física, economía y finanzas, etc.). Se han desarrollado lenguajes de alto nivel que son independientes del computador y que permiten expresar o representar algoritmos de una forma que por un lado es fácilmente comprensible para las personas y por otro lado es traducible automáticamente a código máquina.

El objetivo de este capítulo es hacer una introducción a los lenguajes de programación, en general, y a los de alto nivel, en particular. Estos últimos constituyen una herramienta que hace posible la programación aislando los problemas de la aplicación de los detalles de la máquina que los ejecuta.

A lo largo del capítulo estudiaremos los distintos niveles de complejidad de los lenguajes de programación, los elementos con los que se construye un programa, las distintas fases que se siguen para traducir un programa de un lenguaje a otro, los pasos necesarios para ejecutar un programa y los distintos tipos y dominios de aplicación de los lenguajes de programación.

## 12.1. Concepto de lenguaje de programación

Para ejecutar un algoritmo en un computador es necesario representarlo en forma de **programa**. Para escribir un programa hay que utilizar un **lenguaje de programación**, que es un conjunto predefinido de palabras y símbolos que se utilizan siguiendo unas reglas prefijadas (sintaxis) para expresar algoritmos. **Programar** consiste, por tanto, en establecer órdenes para un computador utilizando un lenguaje de programación; y un **programa** es, por tanto, un conjunto ordenado de instrucciones para un computador indicándole las operaciones que se desea realice. Una **instrucción** está formada por un conjunto de símbolos que representan una orden de operación o tratamiento para el computador.

### 12.1.1. LENGUAJES MÁQUINA

Un computador sólo es capaz de entender y ejecutar directamente programas escritos en su lenguaje máquina. En el Capítulo 4 vimos con cierto detalle el lenguaje máquina (o código máquina) del computador CODE-2. Aunque CODE-2 es una máquina muy elemental, a partir de la expe-

riencia en la utilización de su lenguaje máquina podemos extraer una serie de características de este tipo de lenguajes, independientemente de la complejidad y potencia del computador al que pertenezcan.

Una peculiaridad del lenguaje máquina es que es muy engorroso de utilizar, necesitando el programador conocer la arquitectura física del computador con cierto detalle. La estructura del lenguaje máquina está totalmente adaptada a los circuitos del procesador y muy alejada de la forma o lenguaje habitual en el que solemos expresar y analizar los problemas resolubles con computador. Por ejemplo, para hacer cálculos aritméticos disponemos de un lenguaje o simbología matemática fácil de comprender y clara, que no se parece en nada al código máquina necesario para hacer dichos cálculos. Frente a esto, un programa escrito en lenguaje máquina es directamente interpretable por el procesador central: una vez introducido el programa en el computador (en su memoria principal), no se necesitan transformaciones previas para ser ejecutado (como ocurre con programas escritos en lenguajes simbólicos). Además un programa en código máquina se ejecuta muy eficientemente (con rapidez), ya que el usuario lo redacta específicamente para los circuitos que lo han de interpretar y ejecutar, y desde el código máquina se puede utilizar la totalidad de los recursos de la máquina.

Las principales características de los lenguajes máquina son las siguientes:

- a) *Las instrucciones son cadenas de ceros y unos* (Figura 12.1). Cada instrucción contiene un campo o apartado donde se especifica el **código de operación** de la instrucción y otros campos que indican el lugar (código de registro, dirección de memoria, dirección de puerto de E/S) donde se encuentran los operandos.

0000	1001	1100	0110	1010	1111	0101	1000
1010	1111	0101	1000	0000	1001	1100	0110
1100	0110	1010	1111	0101	1000	0000	1001
0101	1000	0000	1001	1100	0110	1010	1111

**Figura 12.1.** Extracto de un programa en lenguaje máquina (instrucciones de 32 bits).

- b) *Los datos se utilizan por medio de las direcciones de registros o de memoria* donde se encuentran. En las instrucciones no aparecen nombres de variables (tal como  $x$ ,  $y$ ,  $z$ , etc.), sino que el programador debe hacer una *asignación de registros y de direcciones de memoria* para todas las variables y constantes del programa.
- c) *Las instrucciones realizan operaciones relativamente simples.*
- d) *Existe muy poca versatilidad para la redacción de las instrucciones.* Éstas tienen un formato rígido en cuanto a posición de los distintos campos que configuran la instrucción (código de operación, direcciones de memoria, códigos de puertos, etc.). El código de operación debe seleccionarse estrictamente entre los que figuran en una tabla o repertorio fijo.
- e) *El lenguaje máquina depende y está ligado íntimamente al procesador* del computador. Si dos computadores tienen procesadores diferentes (uno no es una réplica del otro), tienen distintos lenguajes máquina.
- f) *En un programa en código máquina no pueden incluirse comentarios* que faciliten la legibilidad del mismo. Además, debido a su representación totalmente numérica, es muy difícil de reconocer o interpretar por el usuario.

Para superar las limitaciones señaladas de los lenguajes máquina se han desarrollado **lenguajes simbólicos**. Estos lenguajes facilitan notablemente el trabajo de programación y hacen los programas más legibles. Se caracterizan porque en vez de ceros y unos se pueden utilizar nombres simbólicos para identificar las instrucciones y para denominar las variables y direcciones de memoria. Hay dos tipos de lenguajes simbólicos: lenguajes ensambladores y lenguajes de alto nivel.

### 12.1.2. LENGUAJES ENSAMBLADORES

Para facilitar la programación ya en la década de los años cincuenta se desarrollaron programas ensambladores. En comparación con un lenguaje máquina, este tipo de lenguajes permiten al programador:

- a) Escribir las instrucciones utilizando una *notación simbólica* o nemotécnica, en vez de códigos binarios, para representar los códigos de operación. Normalmente los códigos nemotécnicos están constituidos por tres o cuatro letras que, en forma abreviada, indican la operación a realizar. Usualmente, debido al origen anglosajón de los fabricantes de computadores, los nemotécnicos son abreviaturas en inglés. La suma, por ejemplo, en la mayoría de los ensambladores se representa como “ADD”.
- b) Utilizar *direcciones simbólicas* de memoria, en lugar de direcciones binarias. Así, por ejemplo, los datos pueden ser referenciados como X, Y, Z, A, B, DEBE, HABER, CONT, etc. Existen sentencias declarativas (también denominadas *directivas* o *seudoinstrucciones*) para indicar al traductor la correspondencia entre direcciones simbólicas y direcciones de memoria. Con estas seudoinstrucciones el traductor crea una tabla con cuya ayuda, al generar las instrucciones máquina, se sustituyen las direcciones simbólicas por las direcciones binarias correspondientes.
- c) Insertar líneas de *comentarios* entre las líneas de instrucciones. El traductor las elimina automáticamente, no incluyéndolas en el código máquina que genera.

El traductor de lenguaje ensamblador a lenguaje máquina se denomina **traductor de ensamblador** o sencillamente **ensamblador**, y mejora o resuelve los problemas a), b) y f) citados al referirnos a los lenguajes máquina, persistiendo las limitaciones c), d) y e), ya que este tipo de lenguajes hace corresponder a cada instrucción en ensamblador una instrucción en código máquina. Un programa en ensamblador no puede ejecutarse directamente por el computador, siendo necesario ser traducido (*ensamblado*) previamente (Sección 12.4).

La mayoría de ensambladores actuales en realidad son macroensambladores. Con ellos se solventa en cierta medida la limitación de tener un repertorio de instrucciones muy reducido. Un **lenguaje macroensamblador** dispone de **macroinstrucciones**, como por ejemplo transferir un bloque de datos de memoria principal a disco, multiplicar, dividir, etc. La macroinstrucción es una llamada a un módulo o rutina, llamada **macro**, de una biblioteca que el traductor inserta en el lugar de la llamada correspondiente, previamente a realizar el proceso definitivo de generación del código máquina. Obviamente a cada macroinstrucción, a diferencia de las instrucciones, le corresponden varias instrucciones máquina y no sólo una.

### 12.1.3. LENGUAJES DE ALTO NIVEL

Aunque los lenguajes ensambladores facilitan notablemente la tarea de programar, siguen presentando inconvenientes notables, como son la dependencia de la arquitectura del procesador, la poca versatilidad del programador para crear sus propias instrucciones y su semántica, que sigue muy alejada de la de los dominios de las aplicaciones.

Para facilitar aún más el trabajo de programación se desarrollaron los **lenguajes de alto nivel**. El primer lenguaje de programación ampliamente difundido fue el FORTRAN, que fue diseñado en 1954 por John Backus en IBM. Se ideó para realizar programas de aplicación en el ámbito científico-técnico; se buscaba un lenguaje muy próximo al utilizado en matemáticas, y de hecho el nombre del lenguaje significa “traductor de fórmulas” (*FORmula TRANslator*). El traductor de FORTRAN para un determinado computador X permite traducir programas escritos en lenguaje FORTRAN al lenguaje máquina de dicho computador X. Desde entonces se desarrollaron cientos de lenguajes de programación de alto nivel, por cierto la mayoría de vida muy efímera.

Los lenguajes de alto nivel se caracterizan por:

- Ser independientes de la arquitectura del computador. El programador no tiene por qué conocer los detalles del procesador que utiliza, y por tanto los programas son *transportables*, ya que se pueden utilizar en computadores con distintos lenguajes máquina.
- Disponer de instrucciones potentes, conteniendo *operadores* y *funciones* de gran diversidad: aritméticas (seno, coseno, módulo, etc.), especiales (cambiar un dato de tipo real a entero, por ejemplo), lógicas (comparar, la función lógica Y, etc.), de tratamiento de caracteres (buscar una subcadena en una cadena de caracteres, por ejemplo), etc. Como consecuencia de lo anterior, durante el proceso de traducción, por lo general, una sentencia en un lenguaje de alto nivel da lugar a múltiples instrucciones en lenguaje máquina.
- Usar una sintaxis parecida al lenguaje natural o al lenguaje del dominio de aplicación del problema planteado. Esto quiere decir que se puede utilizar texto (caracteres alfanuméricos y especiales), se pueden asignar nombres simbólicos a determinados componentes del programa para facilitar su comprensión, el programador puede definir variables con los nombres que quiera, dispone de instrucciones potentes con operadores y funciones de gran variedad, y, como en los lenguajes ensambladores, puede incluir comentarios para facilitar la legibilidad de los programas.

Como consecuencia de este alejamiento de la máquina y acercamiento a las personas, los programas escritos en lenguajes de programación no pueden ser directamente interpretados por el computador, siendo necesario realizar previamente su *traducción* a lenguaje máquina (Sección 12.3). Esta traducción se realiza mediante programas conocidos como **compiladores** e **intérpretes**. Este tipo de programas se incluyen, junto con editores y otro software que realiza tratamiento de textos, dentro de los denominados **procesadores de lenguajes**.

## 12.2. Elementos de un programa en un lenguaje de alto nivel

En esta sección se analizan los elementos principales de un programa redactado en un lenguaje de alto nivel (Tabla 12.1). Cuando no se especifique lo contrario, siempre que utilicemos el término programa nos referiremos a un programa escrito en lenguaje de alto nivel. Por otra parte, en esta sección trataremos fundamentalmente de **lenguajes imperativos** (Sección 12.5.1) que se caracterizan por estar constituidos por secuencias de órdenes, que al ser ejecutadas procesan los datos para producir el resultado deseado. Los lenguajes de programación clásicos y otros tipos de lenguajes tienen muchas características en común con ellos. Asimismo, todos los algoritmos estudiados en el capítulo anterior (Capítulo 11) se han descrito en forma imperativa.

Un programa imperativo, por lo general, está constituido por tres apartados:

- **Cabecera**, que especifica el nombre del programa.
- **Sentencias declarativas**, que dan información al procesador de lenguajes sobre los tipos y estructuras de datos que se utilizarán en el programa. En la declaración se suele incluir el nombre, tipo de la variable y en su caso otros parámetros (tamaño de la estructura, etc). En la Figura 12.2 se muestran ejemplos de declaraciones.
- **Sentencias imperativas**, son las instrucciones que describen los pasos del algoritmo que se redacta.

Todas las sentencias se componen de elementos que pueden ser de cuatro tipos:

- **Palabras reservadas**, que tienen un significado propio dentro del lenguaje de programación. Así, en el ejemplo de la Figura 12.2, *int*, *float*, *char* y *const* son palabras reservadas.



<b>Secciones</b>	<ul style="list-style-type: none"> <li>• Cabecera.</li> <li>• Sentencias declarativas.</li> <li>• Sentencias imperativas: <ul style="list-style-type: none"> <li>— De asignación.</li> <li>— De control: <ul style="list-style-type: none"> <li>■ Decisión sencilla.</li> <li>■ Decisión doble.</li> <li>■ Decisión múltiple.</li> <li>■ Ciclo repetitivo.</li> <li>■ Ciclo condicional.</li> </ul> </li> <li>— De entrada y salida.</li> </ul> </li> </ul>
<b>Elementos</b>	<ul style="list-style-type: none"> <li>• Palabras reservadas.</li> <li>• Identificadores.</li> <li>• Operadores.</li> <li>• Comentarios.</li> <li>• Signos de puntuación.</li> </ul>
<b>Módulos</b>	<ul style="list-style-type: none"> <li>• Programa principal.</li> <li>• Procedimientos (subrutinas).</li> <li>• Funciones.</li> </ul>

Tabla 12.1. Objetos en un programa de alto nivel imperativo.

```
// Ejemplos de sentencias de declaración y comentarios.
int cantidad, n, i, j, k    /* en C, C++, C# y Java */
float precio, total        /* en C, C++, C# y Java */
char sigla;                /* en C, C++, C# y Java */
const pi = 3.1416;         /* definición de un cte. denominada en C++
                           y en C# */
#define iva 0.14            /* definición de una cte. simbólica en C */
float iva=0.14;            /* definición e inicialización de una
                           variable real en Java */
```

Figura 12.2. Sentencias de declaración y comentarios en varios lenguajes de programación.

- **Identificadores**, que son los nombres que se dan a las estructuras, variables, constantes (operandos que no cambian) y funciones, dados arbitrariamente por el programador, dentro de las reglas del lenguaje. Los algoritmos manipulan estos objetos u operandos. Entre las reglas de los lenguajes es habitual, por ejemplo, que los nombres de las variables comiencen por una letra, y que no se puedan utilizar las palabras reservadas para esta finalidad. Recordemos (Sección 12.1) que el lenguaje máquina sólo opera con contenidos de direcciones de memoria (o registros), la utilización de tipos y estructuras de datos permiten al programador utilizar nombres simbólicos en vez de direcciones de memoria o de registros del procesador. En realidad las variables no son más nombres de posiciones de memoria. Los **literales** son los valores que aparecen directamente escritos en el programa, y que pueden ser de los distintos tipos de datos definidos en el lenguaje.
- **Operadores** definidos dentro del lenguaje, y que indican las operaciones que deben realizarse con los operandos. Por lo general, se incluyen operadores:
  - **De asignación**: por lo general es el signo = o el signo ←.
  - **Aritméticos**: +, −, \* y /.
  - **De relación**: <, <=, >, >=, ≠, =. El resultado de la evaluación de una expresión de relación es un valor lógico (verdadero o falso; *true* o *false*).
  - **Lógicos**: correspondientes al álgebra de Boole: and, or, xor, not, etc.
  - **Otros**: Así, en FORTRAN hay un operador // que une dos cadenas de caracteres; por ejemplo, si cadena1="Mañana" y cadena2="lloverá", la sentencia Prediccion=cadena1//cadena2, asignará a Prediccion la cadena "Mañana lloverá".



- **Comentarios**, que sirven para hacer más legible el programa y se escriben con delimitadores especiales. La primera línea de la Figura 12.2 y todas las sentencias al final contienen un comentario. El traductor del lenguaje elimina los comentarios.
- **Signos de puntuación**, para agrupar o separar, como paréntesis, corchetes, punto y coma, etc.

En la Tabla 12.2 se incluyen algunos de los operadores definidos en el lenguaje de programación C.

Tipo	Operador	Definición	Explicación o ejemplo
<b>Operadores de asignación</b>	=	Asignación de valor a variable.	$v = A + 3$
	+=	Sumar un valor a la variable.	$v += 2; /* v \leftarrow v + 2 */$
	=	Restar un valor a la variable.	$v -= 2; /* v \leftarrow v - 2 */$
	*=	Multiplicar un valor a la variable.	$v *= 2; /* v \leftarrow v * 2 */$
	/=	Dividir un valor a la variable.	$v /= 2; /* v \leftarrow v / 2 */$
	%=	Residuo de la división por un valor.	$v += 2; /* v \leftarrow v \% 2 */$
<b>Operadores aritméticos</b>	+	Suma.	$A + 3.5$
	-	Resta.	$A - 3.5$
	*	Multiplicación.	$A \cdot 3.5$
	/	División.	$A / 3.5$
	%	Residuo (resto de la división entera).	$A \% 3.5$
	++ --	Sumar 1 al valor de la variable. Restar 1 al valor de la variable.	$i++ /* i \leftarrow i + 1 */$ $i-- /* i \leftarrow i - 1 */$
<b>Operadores de relación</b>	<	<	$v < 7$
	<=	≤	$v \leq 7$
	>	>	$v > 7$
	>=	≥	$v \geq 7$
	=	=	$v = 7$
	!=	≠	$v \neq 7$
<b>Operadores lógicos</b>	!	NOT	$!(v > 5) /* \text{si } v > 5, \text{ falso} */$
	&&	AND	$(v > 5) \&\& (v < 10) /* \text{si } v > 5 \text{ y } v < 10, \text{ verdad} */$
		OR	$(v = 1)    (v = 2) /* \text{si } v \text{ es } 1 \text{ o } 2, \text{ verdad} */$

**Tabla 12.2.** Operadores definidos en el lenguaje de programación C.

Las sentencias imperativas se clasifican en sentencias de asignación, de control y de entrada/salida. Otros elementos de interés en la realización de programas son los procedimientos y las funciones. A continuación se describen brevemente.

### 12.2.1. SENTENCIAS DE ASIGNACIÓN

Las sentencias de asignación tienen la forma general de la Figura 12.3, con el significado siguiente: evaluar la *expresión* y el resultado almacenarlo como valor de la variable  $v$  (en la posición de memoria definida por  $v$ ). En general, los lenguajes de programación utilizan varios tipos de expresiones:

- **Expresión algebraica**, que combina variables, constantes de tipo numérico (entero o real) y funciones matemáticas con los operadores aritméticos.
- **Expresión lógica**, que combina variables y constantes de tipo lógico con operadores lógicos, o variables aritméticas con operadores de relación.
- **Expresión de caracteres**, que incluye variables y constantes de tipo carácter con operaciones definidas en este tipo (ver Secciones 10.1.4 y 10.2.2).

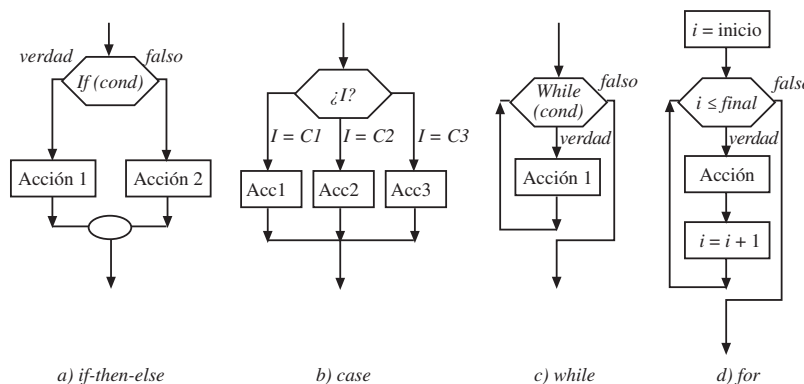
En pseudocódigo y APL	En Pascal y Ada	En C, C++, C# y Java
$v \leftarrow \text{expresión};$	$v := \text{expresión};$	$v = \text{expresión};$

**Figura 12.3.** Sentencia de asignación en distintos lenguajes de programación.

## 12.2.2. SENTENCIAS DE CONTROL

Las sentencias de control permiten incluir en el programa las estructuras de control que estudiamos en el Capítulo 11. Las más relevantes (Figura 12.4) implementan:

- **Decisiones sencillas**, que suponen realizar una acción si se verifica una condición (si-entonces o **if-then**).
- **Decisiones dobles**, del tipo *Si condición hacer acción 1 si no acción 2* (sentencias **if-then-else**).
- **Decisiones múltiples**, que permiten seleccionar entre múltiples opciones, siendo como una instrucción if-then-else, pero ampliada a más de dos casos (sentencias “según sea-hacer” o **case** o **switch**).
- **Ciclos repetitivos** del tipo *Para i = inicio hasta final repetir acción* (sentencias **for**).
- **Ciclos condicionales**, que implementan las estructuras *Mientras condición repetir* (sentencias **while**).



**Figura 12.4.** Estructuras de control implementadas por varias instrucciones.

En la Figura 12.5 se muestra la forma que tienen estas sentencias en distintos lenguajes de programación.

## 12.2.3. SENTENCIAS DE ENTRADA/SALIDA

Las sentencias de entrada/salida permiten comunicar el programa con los distintos periféricos: leer del teclado, escribir en pantalla o en una impresora, leer/escribir en disco duro, etc.

Las sentencias de E/S se componen de tres elementos:

- La *palabra reservada* que especifica la operación. En la Figura 12.6 se indican las utilizadas en algunos lenguajes (*readln*, *writeln*, etc.).

Sentencia	Lenguaje de programación	
	Pascal	C, C++, C# y Java
if-then-else	<pre> if cond then     begin         acción1     end else acción2 </pre>	<pre> if (cond) Acción1 else Acción2 </pre>
case	<pre> case c of     c1: acción1     c2: acción2     .....     cn: acciónn end (*case*) </pre>	<pre> switch (I) { case C1: Acc1; break;   case C2: Acc2; break;   .....   Case Cn: Accn; break; }; </pre>
while	<pre> while cond begin     acción end </pre>	<pre> while (cond)     Acción; </pre>
for	<pre> for i := inicio to final do begin     acción end </pre>	<pre> for (int i = inicio; i ≤ fin; i++)     Acción; </pre>

**Figura 12.5.** Sentencias de control típicas de algunos lenguajes de programación.

Sentencia	Lenguaje de programación		
	Pascal	C	C++, C# y Java
Entrada	readln (variables);	scanf (variables);	cin >> variables;
	Leer valores del teclado	Leer valores del teclado	Leer valores de la entrada estándar
Salida	writeln(variables);	printf("formato", variables);	cout << variables;
	Escribir valores en la pantalla	Escribir valores en la pantalla	Escribir valores en la salida estándar

**Figura 12.6.** Sentencias de E/S típicas de algunos lenguajes de programación.

- El *formato de E/S* que especifica cómo se debe generar la salida o cómo se encuentran los datos de entrada. En el formato de salida se especifican cuestiones tales como rótulos o textos explicativos de los datos de salida, número de cifras decimales de los números reales, espacios y saltos de línea, etc. El formato es opcional, y la forma de describirlo depende mucho del lenguaje de programación; habitualmente se incluye entre comillas.
- La *variable o variables implicadas en la operación de E/S*. Si es una entrada, se asignarán a las variables especificadas los valores leídos; si es una salida, los datos de salida serán los valores que contengan en ese momento las variables especificadas.

**EJEMPLO 12.1**

Las sentencias de Pascal que se incluyen en la Figura 12.7 realizan las siguientes operaciones:

- La primera sentencia se limita a escribir el texto indicado.
- La segunda escribe: "X1 = " y a continuación el valor que en ese momento almacene la variable *raiz1*.
- La tercera escribe: "X2 = ", a continuación el valor que en ese momento almacene la variable *raiz2* y después "; y".

```
writeln ("La ecuación tiene soluciones reales, éstas son:");  
writeln ("X1 = ", raiz1, "; y");  
writeln("X2 = ", raiz2);
```

**Figura 12.7.** Ejemplos de sentencias de E/S en Pascal.

- Cada vez que se ejecuta una sentencia *writeln* se avanza una línea en la imagen de pantalla, de forma que la imagen que se obtendrá es de la forma indicada en la Figura 12.8.

```
La ecuación tiene soluciones reales, éstas son:  
X1 = 48,25; y  
X2 = -3,15
```

**Figura 12.8.** Salida en pantalla obtenida con las sentencias de los Ejemplos 12.1 y 12.2.

**EJEMPLO 12.2**

Las sentencias en C que se incluyen en la Figura 12.9 produce los mismos efectos que las sentencias en Pascal analizadas en el ejemplo anterior (Ejemplo 12.1, Figura 12.8). Algunas observaciones de interés son las siguientes:

- El código `\n` indica que hay que saltar una línea.
- Los códigos que se inician con el carácter `%` especifican la posición donde debe incluirse un dato de la lista de variables y el formato de su escritura. Así, en el ejemplo se indica que después de "X1 = " hay que escribir el primer dato de la lista como número real (`%f`). Otros códigos son `%d` para especificar un número decimal, `%s` para una cadena de caracteres, etc.
- La lista de variables a imprimir se dan después del formato, y se deben corresponder con el orden especificado en este último.

```
printf("La ecuación tiene soluciones reales, éstas son: \n X1 = %f ; y \n  
X2 = %f", raiz1, raiz2);
```

**Figura 12.9.** Ejemplo de sentencia de E/S en C.

**12.2.4. SUBPROGRAMAS**

Con frecuencia un algoritmo puede descomponerse en módulos o subalgoritmos, resultando, por lo general, muy útil esta descomposición sobre todo cuando se abordan problemas complejos. Los módulos son unidades que realizan tareas relativamente sencillas y que, además, pueden utilizarse en varios programas. El desarrollo de un programa de forma modular facilita notablemente su legibilidad y su prueba.

En capítulos anteriores hemos implementado los subalgoritmos por medio de **subrutinas** o **rutinas**, y hemos analizado cómo se produce físicamente el proceso de llamada a una subrutina, y el de retorno de una subrutina al programa que lo llama.

Desde el punto de vista de los lenguajes de programación de alto nivel, los subalgoritmos se pueden implementar bajo la forma de **procedimientos** o **funciones**. Un procedimiento o una función es un conjunto de instrucciones que realizan una actividad o tarea determinada y que puede ser llamada y usada por otros programas o procedimientos.

Los procedimientos se inician con una **cabecera del procedimiento**, que especifica el nombre del mismo. A continuación se incluyen las sentencias que realizan la actividad del procedimiento, **acción del procedimiento**. Tradicionalmente un procedimiento concluye con una **instrucción de retorno** (*return*) al programa que lo llama.

Las variables declaradas dentro de un procedimiento se denominan **variables locales**, y sólo pueden utilizarse y tienen significado dentro del procedimiento; así procedimientos distintos pueden utilizar variables locales con igual denominación pero con significados distintos. Algunos lenguajes permiten en ciertas circunstancias definir **variables globales**, cuyos valores pueden darse y utilizarse por distintos programas. Normalmente un procedimiento debe intercambiar datos con la unidad de programa que lo llama, y esto se hace por medio de *parámetros*. Los que utiliza el procedimiento se denominan **parámetros formales**. En la cabecera del procedimiento, después de su nombre se incluyen entre paréntesis los nombres de los parámetros formales. El procedimiento utiliza los parámetros formales de entrada como variables, aunque sus valores no se asignen dentro de él. Las llamadas a los procedimientos desde las unidades de programación que los utilizan se hacen con el mismo formato de la cabecera incluyendo, por tanto, el nombre del procedimiento y a continuación, entre paréntesis, la lista de **parámetros reales** que contiene las variables y constantes con los valores con los que se desea se evalúe el procedimiento. En la lista de parámetros también deben incluirse los de salida, que son las variables que contendrán los valores resultado de la ejecución del procedimiento con los parámetros reales. En otras palabras, el procedimiento asigna valores a los parámetros reales de salida. Los nombres de los parámetros formales y reales no tienen por qué ser los mismos, realizándose una correspondencia correcta entre ellos por el orden que ocupan dentro de las respectivas listas. El procedimiento descrito para transferir datos entre programas y procedimientos se denomina **paso de parámetros por valor**. Una alternativa es el **paso de parámetros por referencia**; en este caso, en la lista de parámetros se incluyen las direcciones (punteros) de los parámetros reales, en lugar de dar acceso directo a los mismos. El procedimiento obtiene los valores de los parámetros y ubica los resultados en las direcciones especificadas en la llamada al procedimiento.

### EJEMPLO 12.3

En la Figura 12.10 se incluye el procedimiento, *Media*, que obtiene el valor medio, *Vmedio* de los datos incluidos en un array *Valores*.

```
void Media (float Valores[100], float
Vmedio)
{float suma = 0;
  for (int i=1; i<100; i++)
    suma=suma+Valores[i];
  Vmedio=suma/100;
  return void
}
```

a) Procedimiento Media

```
-----
Antes de llamada a Media hay que asignar
valores a X
-----
Media (X,MX)
A partir de aquí puede utilizarse el
valor MX
-----
A = B*47.05 + MX/C;
-----
```

b) Llamada al procedimiento Media

Figura 12.10. a) Ejemplos de procedimiento en C y b) de su llamada desde un programa.

En el ejemplo de la Figura 12.10:

- *Void* es una palabra reservada (que significa nulo o vacío) que se utiliza en C aquí para especificar que lo que sigue es un procedimiento (no una función)
- *Valores* y *Vmedio* son los parámetros formales del procedimiento, de entrada y salida, respectivamente.
- *Suma* es una variable local.
- *X* y *MX* son los parámetros reales, a través de los cuales se intercambian datos entre el programa de llamada y el procedimiento. *X* es de entrada y *MX* de salida, con respecto al procedimiento.

### 12.2.5. FUNCIONES

Un procedimiento puede intercambiar con el programa que lo llama uno o varios parámetros o incluso ninguno. Por ejemplo, puede definirse un procedimiento que escriba un texto determinado en el monitor de salida; en este caso, el programa que llama al procedimiento no intercambia ningún parámetro con éste.

Una función es un programa que implementa un subalgoritmo que genera un valor que se transfiere al programa de llamada a través del propio nombre de la función. La diferencia fundamental, por lo tanto, entre procedimiento y función es que en estas últimas el resultado de su ejecución no se transfiere por medio de parámetros. Un ejemplo de función es un procedimiento *sin(x)* que calculase el seno del valor *x*. Una vez definida la función, dentro de programa se podría utilizar *sin(y)* como una variable más: la llamada a *sin* genera el seno de *y*, y su valor se asigna a *sin*.

#### EJEMPLO 12.4

En la Figura 12.11 se muestra la función y un ejemplo de cómo llamarla. Obsérvese que el nombre de la función se utiliza como si fuese una variable más, real, en este caso.

```
float media (float Valores[100])
{float suma;
 suma=0;
 for (int i=1; i<100; i++)
     suma=suma+Valores[i];
 Media=suma/100;
 return Media;
}
```

a) Función Media

```
-----
Antes de utilizar Media hay que asignar
valores a X
-----
A = B*47.05 + Media(X)/C;
-----
Se asignan valores al array Y
-----
MY=Media(Y);
-----
```

b) Dos llamadas a la función Media

**Figura 12.11.** a) Ejemplos de función en C y b) de su utilización desde un programa.

En el ejemplo de la Figura 12.11:

- La función debe terminar con una sentencia *return* que indica también la variable que se asocia al valor de salida (resultado) de evaluar la función.
- En el programa de llamada se puede utilizar *Media(Y)* como una variable cualquiera. En la Figura 12.11b se hacen dos llamadas a la función *Media*, una con el conjunto de datos *X*, y otra con el conjunto de datos *Y*.

Los procesadores de lenguajes suelen incluir, además, **bibliotecas de funciones** que pueden ser utilizadas arbitrariamente por el programador. Así, es habitual disponer de funciones matemáticas (logaritmo, raíz cuadrada, etc.), trigonométricas (seno, coseno, etc.), de contabilidad (porcentaje, interés compuesto, etc.), para realizar las operaciones de entrada/salida, etc. Las **funciones de biblioteca** facilitan notablemente el trabajo de programación, y, como se analizará en la Sección 12.4, deben incorporarse al programa que las utiliza antes de ejecutarse.

### 12.3. El proceso de traducción

Como indicamos en la Sección 12.1, los lenguajes de programación de alto nivel posibilitan la utilización de una simbología y una terminología próximas a las usadas tradicionalmente en la descripción de problemas. Ahora bien, el procesador de un computador sólo puede interpretar y ejecutar código máquina; por tanto, para ejecutar un programa redactado en un lenguaje de alto nivel es necesario traducirlo a lenguaje máquina. Un **traductor** es un programa que traduce un programa de uno a otro lenguaje. El programa original se denomina **programa fuente**, y el programa obtenido, **programa objeto**. Recordemos que, según vimos en la Sección 11.1, un algoritmo es un ente abstracto que *se puede implementar de muy diversas formas*, y que, para ser ejecutado por un computador, debe representarse mediante un programa. El traductor realiza un cambio de representación de los mismos algoritmos: utiliza como datos de entrada el texto del programa fuente y genera como resultado el programa objeto que es otra forma del programa inicial redactado en otro lenguaje. En la Figura 12.12 se representa el programa Media, descrito en el Ejemplo 12.4 para obtener el valor medio de una serie de datos, redactado en C por el usuario, y cómo es utilizado por un traductor como conjunto de datos que debe procesar para generar un programa objeto.

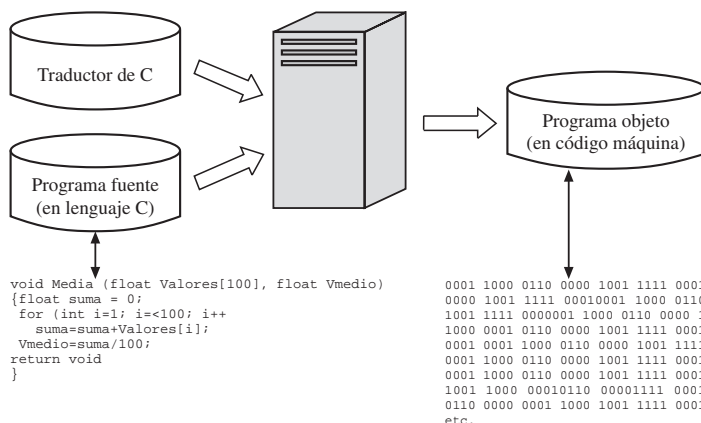


Figura 12.12. Ejemplo de proceso de traducción.

Los traductores descritos en el párrafo anterior se denominan **compiladores**, siendo una de sus características el generar un archivo con el programa objeto, de forma que, una vez obtenido éste, se puede utilizar cuantas veces se quiera, sin necesidad de repetir el proceso de traducción (evidentemente, a no ser que se modifique el programa fuente). Si la traducción se realiza a código máquina, esta copia es válida para ejecutarla cuantas veces se quiera. Una alternativa a los compiladores son los **intérpretes**, éstos son traductores que ejecutan las instrucciones conforme las van traduciendo, sentencia a sentencia, de forma que no se crea un programa objeto. Es decir, el intérprete capta una sentencia, la analiza y la traduce dando lugar a su ejecución inmediata. Tienen la ventaja de que el usuario puede corregir errores sobre la marcha, a medida que se va ejecutando el programa. Sin embargo, como con un intérprete cada vez que se ejecuta el programa hay que volver a traducirlo, el proceso

de ejecución resulta muy poco eficiente. Además, si, por ejemplo, hay un bucle que tiene que ejecutarse 100 veces, éste se traducirá esas mismas veces. Por otra parte, como veremos más adelante en esta misma sección, un intérprete no puede mejorar ni hacer una optimización global del programa (sólo lo puede hacer a nivel de sentencia).

El proceso de compilación fundamentalmente consta de cuatro actividades: análisis lexicográfico, análisis gramatical, generación de código y optimización (Figura 12.13), que no se producen de forma ni aislada ni completamente secuencial.

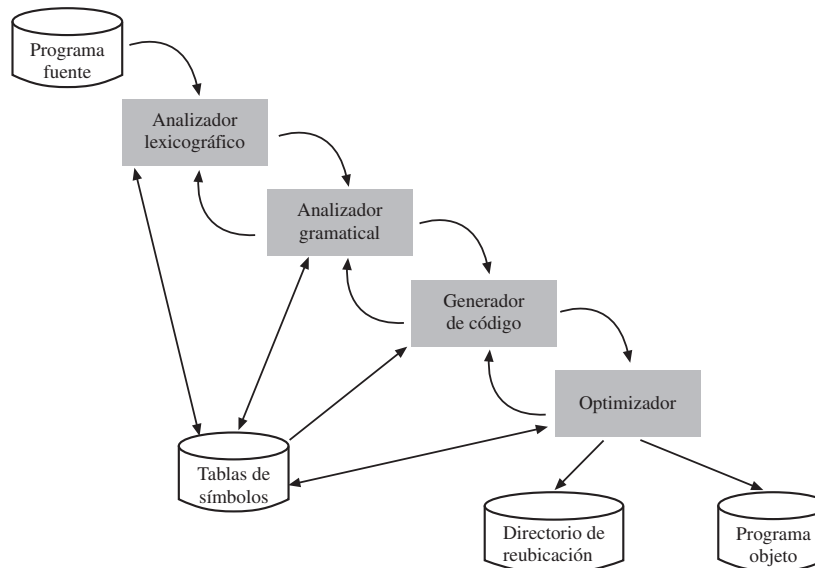


Figura 12.13. Acciones a desarrollar durante el proceso de traducción de un programa.

El **analizador lexicográfico** (*scanner*) reconoce las cadenas de símbolos del programa fuente que representan una entidad sencilla, tales como variables, valores numéricos, palabras reservadas y operadores. Una vez clasificadas esas **unidades de léxico**, genera un patrón de bits conocido como **símbolo lexicográfico** (*token*) para representar a cada unidad. El analizador lexicográfico descarta los comentarios del programa, ya que éstos no intervienen en el proceso de traducción.

El **analizador gramatical**, que realiza un análisis sintáctico (*parser*) y semántico del programa, considera el programa como un conjunto de símbolos lexicográficos y, a partir de ellos, identifica las estructuras gramaticales del programa reconociendo el papel de cada componente. El análisis gramatical se fundamenta en una serie de reglas que definen la sintaxis del lenguaje de programación. Estas reglas se suelen representar por medio de **diagramas sintácticos**, o por medio de formas en **notación BNF** (*Backus-Naur Form*).

### EJEMPLO 12.5

La Figura 12.14 muestra el diagrama sintáctico de las instrucciones *if-then-else*, que se interpreta de la siguiente forma: la instrucción comienza con la palabra reservada *if*, le sigue una expresión booleana, la palabra reservada *then* y después se incluyen una, u opcionalmente, varias sentencias. Posteriormente sigue la palabra reservada *else* y una o varias sentencias, aunque estos últimos elementos son opcionales.

Los términos que aparecen dentro de círculos se denominan elementos terminales, ya que no requieren descripciones adicionales; sin embargo, los términos en rectángulos requieren diagramas sintácticos adicionales que aclaren su estructura.



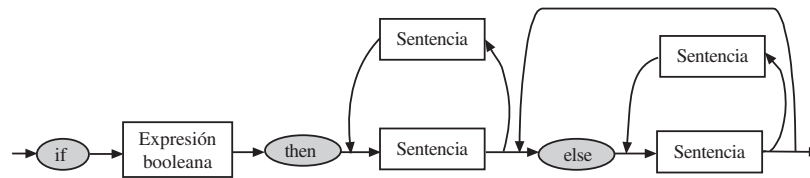


Figura 12.14. Diagrama sintáctico para la sentencia *if-then-else*.

### EJEMPLO 12.6

La estructura de la construcción *if-then-else* en BNF es la siguiente:

$$\langle \text{if-then-else} \rangle \rightarrow \text{if} \langle \text{expresión booleana} \rangle \text{then} \{ \langle \text{sentencia} \rangle \} (\text{else} \{ \text{sentencia} \} \mid 0)$$

ya que:

- $\langle \text{término} \rangle$  significa elemento no terminal.
- $\{ \text{término} \}$  significa iteración de término.
- $(\text{término1} \mid \text{término2})$  indica que se puede optar entre término1 o término2.

El analizador gramatical para comprobar si una estructura particular se ajusta a un diagrama sintáctico puede utilizar un **árbol de análisis sintáctico**, como se muestra en el siguiente ejemplo.

### EJEMPLO 12.7

El árbol de análisis sintáctico de la sentencia:

*if condición then z=z+1 else z=0;*

se muestra en la Figura 12.15. El nodo raíz del árbol es *Sentencia*, que a continuación se va descomponiendo en árboles y ramas, hasta llegar a los elementos terminales.

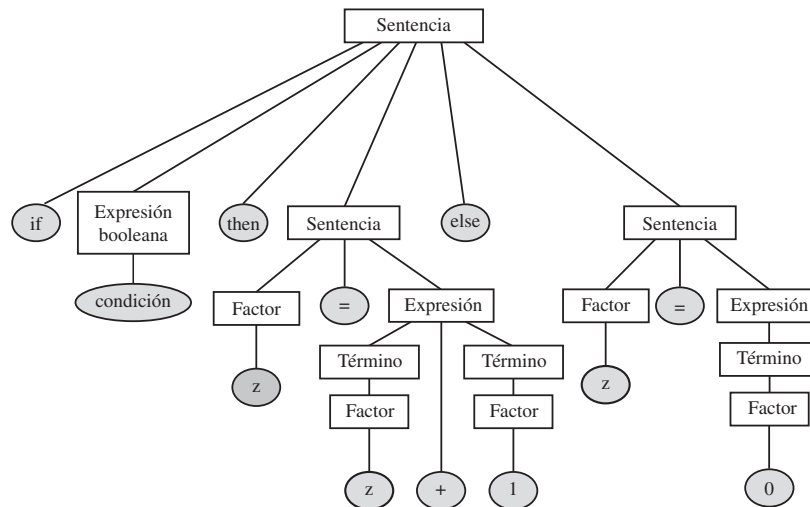


Figura 12.15. Árbol de análisis gramatical de la sentencia *if condición then z=z+1 else z=0;*

El proceso de traducción se hace con ayuda de unas tablas de símbolos que se generan durante las etapas de análisis y se consultan y completan en etapas posteriores. La **tabla de símbolos** contiene información con los nombres simbólicos de todas las variables y constantes definidas por el programador, el tipo a que pertenecen y su posición en memoria.

En la **generación de código** se crea un archivo con el programa en lenguaje objeto (normalmente lenguaje ensamblador o lenguaje máquina) que describe (de otra forma) los mismos algoritmos representados en el texto fuente. Habitualmente se utiliza un lenguaje intermedio (distinto del código objeto final), con el propósito de facilitar la optimización del código.

Durante la generación de código intermedio se completan y consultan las tablas generadas en fases anteriores (tablas de símbolos, de constantes, etc.). También se realiza la asignación de memoria a los datos definidos en el programa.

La generación de código puede realizarse añadiendo procedimientos en determinados puntos del proceso de análisis, que generan las instrucciones en lenguaje objeto equivalentes a cada construcción reconocida en el lenguaje fuente. Todos los traductores disponen de bibliotecas de procedimientos con este objetivo.

La **optimización** tiene por misión mejorar el código intermedio, analizándose globalmente el programa objeto. Las optimizaciones pueden ser de dos tipos: independientes o dependientes del procesador. Una optimización del primer tipo podría realizarse, por ejemplo, si el compilador encuentra una o varias instrucciones en un ciclo de  $N$  iteraciones que se pueden sacar de él; de esta forma dichas instrucciones se ejecutarían una sola vez en lugar de  $N$  veces. Las optimizaciones dependientes del procesador son también muy importantes, y un buen compilador debe considerarlas para aprovechar al máximo las prestaciones que ofrece el hardware. Así, por ejemplo, el compilador debe asignar las variables y constantes del programa a registros del procesador o a memoria principal de forma que se reduzcan al máximo los accesos a esta última.

En cualquiera de las fases de análisis el compilador puede dar mensajes sobre los errores que detecta en el programa fuente, cancelando en ocasiones la traducción para que el usuario realice las correcciones oportunas en el archivo fuente. La compilación es un proceso complejo y que consume a veces un tiempo muy superior a la propia ejecución del programa. Existen compiladores que permiten al usuario omitir o reducir, a su conveniencia, las fases de optimización, disminuyéndose así el tiempo global de la compilación.

## 12.4. Pasos a seguir para la ejecución de un programa

Una vez realizados el planteamiento y el análisis del problema, incluyendo el desarrollo de los algoritmos que lo resuelven, para obtener la solución por computador se deben seguir los siguientes pasos (Figura 12.16):

- Escribir el programa.
- Traducirlo.
- Enlazarlo.
- Cargarlo.
- Ejecutarlo.
- Depurarlo.

Los programas se escriben con un editor de textos o con programas específicos asociados al lenguaje que se vaya a utilizar y que contienen herramientas que facilitan la escritura en un lenguaje de alto nivel. Un **editor de textos** es un programa de utilidad que nos permite introducir y modificar (borrar, intercalar, cambiar, ampliar, duplicar, etc.) cómodamente información (de programas o datos) en

un archivo. Podríamos decir que esta fase es la introducción y corrección “mecanográfica” del programa. El editor genera un archivo con el **programa fuente**.

El siguiente paso es traducir el programa en lenguaje de alto nivel —el programa fuente— a lenguaje máquina, con un compilador o un intérprete. Por lo general, los compiladores generan el programa objeto en ensamblador, y en una etapa posterior, un traductor de ensamblador genera el código máquina.

Cuando se produce un fallo en un programa traducido con un **compilador** hay que corregir el error en el programa fuente, compilarlo y ejecutarlo de nuevo. Existen traductores, denominados **compiladores interactivos**, que permiten editar, compilar, depurar y ejecutar un programa de forma más sencilla y además se pueden hacer comprobaciones y modificaciones durante la ejecución (como en un intérprete) sin tener que volver a compilar todo.

Caso de que el compilador haya generado un programa en ensamblador, a continuación hay que *ensamblarlo* mediante un traductor de ensamblador a código máquina, de forma que se genera un nuevo archivo que contiene el programa en lenguaje máquina. El nuevo programa objeto se dice que es **reubicable**, y aunque está en código máquina no es directamente ejecutable. En efecto, la mayoría de programas hacen llamadas a diversos procedimientos o funciones del propio usuario o del sistema operativo o de la biblioteca de funciones del compilador que deben *unirse* al programa principal. Todos estos módulos son piezas del programa, y antes de unirse o enlazarse, deben estar compilados y ensamblados. Los procedimientos a unir han de ser **módulos reubicables**, en el sentido de que su direccionamiento debe ser relativo (virtual), comenzando la primera instrucción en la dirección 0 de memoria, y deben tener asociados unas tablas (**directorios de reubicación**) donde se encuentran anotadas las direcciones relativas de las instrucciones que hacen referencia a direcciones del propio módulo (instrucciones de saltos, por ejemplo) o a otros módulos (llamadas a procedimientos).

Para efectuar la unión de los distintos módulos, “encadenando” o “enlazando” las llamadas entre ellos, se utiliza un **enlazador**, que es un programa que genera un **archivo ejecutable** (también llamado **archivo binario**) listo para ser ejecutado. El enlazador, para realizar su trabajo, utiliza los directorios de reubicación de los módulos reubicables y genera una **tabla de símbolos externos**, que incluye los nombres y direcciones de las instrucciones a las que hay que saltar desde otros módulos.

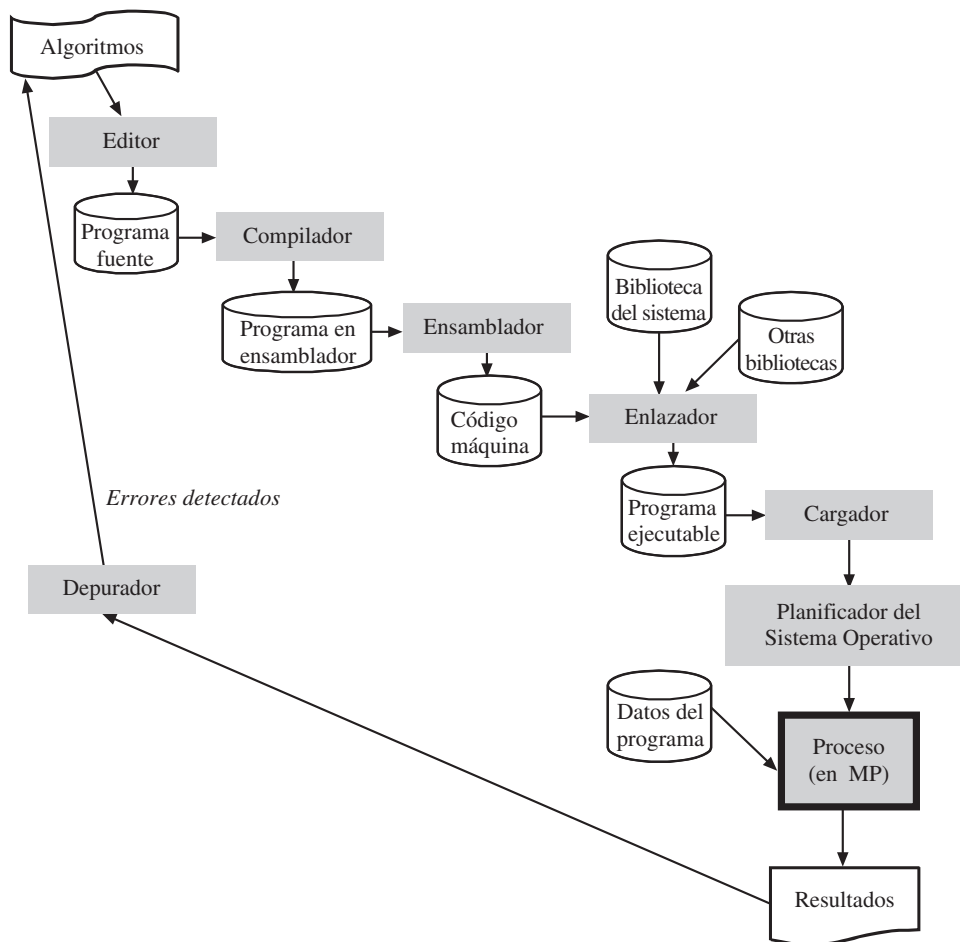
La fase siguiente es introducir o cargar el programa ejecutable en memoria, y prepararlo para su ejecución. Estas operaciones las realiza una utilidad denominada **cargador**, que suele formar parte del planificador del sistema operativo. El cargador se encarga de, con ayuda de la tabla de símbolos externos, reajustar las direcciones del programa a las direcciones físicas donde se almacenará el mismo. Piénsese que, por lo general, cuando se carga un programa en memoria cada vez lo hace en un sitio distinto, dependiendo de los espacios libres en el momento de efectuar la carga (para más detalles ver Sección 9.4).

Una vez que se tiene el archivo ejecutable, y es cargado en memoria, el programa se convierte en un **proceso** que se ejecuta bajo el control del planificador del sistema operativo de acuerdo con los detalles que se indican en la Sección 9.3.

Otra ayuda a la programación de que se suele disponer son las herramientas de **rastreo y depuración** de errores, que facilitan la corrección de errores y optimización de los programas. Estas utilidades permiten incluir en distintos puntos del programa controles para obtener los valores de las variables según se va ejecutando el programa, y ejecutar el programa instrucción a instrucción, mostrándose después de cada ejecución el contenido de las variables que se rastrean.

## 12.5. Clasificación de los lenguajes de programación

Los lenguajes de programación se pueden clasificar, según se indica en la Tabla 12.3, atendiendo a distintos criterios, como son su nivel, su estilo y su campo de aplicación. En la Sección 12.1 ya analizamos los distintos niveles de los lenguajes de programación: máquina, ensambladores y de alto nivel, que se definen en función de su acercamiento a la máquina o a la aplicación. En la presente sección haremos una introducción a la clasificación de los lenguajes atendiendo a los dos últimos criterios citados.



**Figura 12.16.** Pasos para la ejecución de un programa.

<b>Niveles</b>	<ul style="list-style-type: none"> <li>• Bajo nivel: Lenguaje (<i>código</i>) máquina.</li> <li>• Simbólicos:               <ul style="list-style-type: none"> <li>— Ensambladores.</li> <li>— Alto nivel.</li> </ul> </li> </ul>
<b>Estilos o paradigmas de programación</b>	<ul style="list-style-type: none"> <li>• Programación imperativa (de procedimientos).</li> <li>• Programación funcional.</li> <li>• Programación orientada a objetos.</li> <li>• Programación declarativa (lógica).</li> <li>• Programación guiada por eventos.</li> <li>• Programación concurrente.</li> </ul>
<b>Dominios de aplicación</b>	<ul style="list-style-type: none"> <li>• Aplicaciones científico-técnicas.</li> <li>• Aplicaciones de gestión de información.</li> <li>• Aplicaciones de inteligencia artificial.</li> <li>• Aplicaciones de programación de sistemas.</li> <li>• Aplicaciones para web.</li> </ul>

**Tabla 12.3.** Clasificación de los lenguajes de programación.

### 12.5.1. ESTILOS DE PROGRAMACIÓN

Según el estilo de programación, los lenguajes de alto nivel pueden clasificarse de acuerdo con los siguientes paradigmas:

- Programación imperativa (de procedimientos).
- Programación orientada a objetos.
- Programación funcional.
- Programación declarativa (lógica).
- Programación guiada por eventos.
- Programación concurrente.

Hay que destacar que un mismo lenguaje puede tener características que permiten programar con él en distintos estilos. A continuación se describen brevemente estos paradigmas.

#### PROGRAMACIÓN IMPERATIVA

La forma más tradicional de programar es utilizar **lenguajes imperativos**, también denominados **lenguajes de procedimientos**, que siguen la misma sistemática que los lenguajes máquina. El programador diseña un algoritmo detallando el *procedimiento* a seguir para resolver el problema y lo representa por medio de sentencias del lenguaje, de forma que cada una de ellas es una orden que indica al computador la realización de una tarea específica. Es definitiva, un programa en un lenguaje imperativo está formado por una secuencia de instrucciones que al ejecutarse procesan los datos obteniendo el resultado buscado.

#### PROGRAMACIÓN ORIENTADA A OBJETOS

La programación orientada a objetos supone una forma de resolver problemas, totalmente diferente a la empleada en los lenguajes imperativos, y más cercana a la manera de pensar de los humanos. La idea básica consiste en considerar los conjuntos de datos como objetos activos, y no como unidades pasivas, como hace la programación imperativa. Un **objeto** se define como un conjunto de **datos** con una serie de **atributos** y **funciones** (o acciones) asociadas. Por ejemplo, en la **programación orientada a objetos** u **OPP** (*Object-Object Programming*) podemos definir una lista como un objeto, que incluye además de la lista de datos propiamente dicha una colección de funciones para operar con ella como pueden ser ordenarla, encontrar sus elementos mínimo y máximo, insertar un nuevo elemento, borrar un elemento y detectar si está vacía. La utilización de los objetos se hace por medio de **paso de mensajes** que se emiten y reciben en los objetos a través de una interfaz normalizada muy bien definida.

Los lenguajes de OPP disponen de bibliotecas de objetos, pudiendo el usuario definir otros adicionales. Los objetos disponen de propiedades de gran interés que facilitan notablemente la programación, entre las que se encuentran las siguientes:

- Un objeto puede *contener* a su vez otros objetos.
- Los objetos están encapsulados, en el sentido de que únicamente se puede acceder a determinados atributos y datos desde el exterior, a través de la interfaz y por medio de mensajes. El **encapsulamiento** implica la existencia de un acceso restringido a ciertos datos y operaciones internas del objeto.
- Los objetos no realizan funciones de forma espontánea, sino que, según se ha comentado anteriormente, deben recibir un *mensaje* para llevar a cabo una determinada función.
- Un mensaje para realizar una operación determinada puede implicar la realización de tareas distintas en distintos objetos o clases. Por ejemplo, podemos tener distintos objetos geométricos,

tales como círculos, rectángulos, triángulos, etc., y dar a todos la orden de calcular la superficie de sus ejemplares (instancias); la tarea de calcular la superficie es distinta para cada objeto (el cálculo del área es diferente para un triángulo que para un círculo o un cuadrado), aunque la orden es común. Esta propiedad se denomina **polimorfismo**

- Distintos objetos que comparten atributos y funciones pueden formar una **clase**. De hecho en Java cualquier objeto necesariamente debe pertenecer a una clase. Los objetos de una clase poseen todos los atributos y funciones (o métodos) de esa clase.
- Un objeto puede reutilizar propiedades de otro, diciéndose en este caso que hay una **herencia**. De esta forma las clases se pueden dividir en **subclases** con atributos y funciones heredados de la clase de la que procede. Con la herencia se simplifica notablemente la definición de nuevos objetos.

Los lenguajes orientados a objetos más notables son Smaltalk, C++, C#, Eiffel y Java.

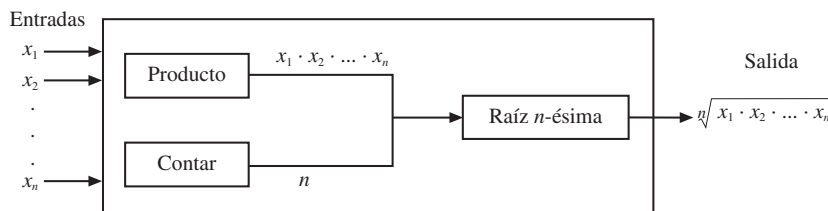
## PROGRAMACIÓN FUNCIONAL

Estos lenguajes están basados en la utilización y definición de funciones o relaciones. Un programa está constituido por un conjunto de funciones, cada una de ellas considerada como una caja negra que recibe entradas y genera resultados. Un lenguaje funcional contiene una colección de **primitivas** (cientos, las versiones actuales de LISP) que puede utilizar cualquier programador y éste a partir de ellas puede crear nuevas funciones. El programa determina la interacción y combinación de unas funciones con otras por medio de condiciones, composición funcional y recursividad.

Ejemplos destacados de estos lenguajes son LISP, ML, Haskell y Scheme.

### EJEMPLO 12.8

En la Figura 12.17 se muestra cómo se implementaría la media geométrica de una serie de números utilizando un lenguaje funcional. Se supone que el lenguaje dispone de las primitivas Producto, Contar y Raíz  $n$ -ésima; combinándolas adecuadamente creamos una función nueva (el contenido del recuadro grande) de forma que al proporcionarle unas entradas ( $x_1, x_2, \dots, x_n$ ) genera como salida la media geométrica de las entradas.



**Figura 12.17.** Ejemplo de programa en un lenguaje funcional.

La función se podría construir así:

```
(raíz (contar entradas) (producto entradas))
```

Estos lenguajes de programación tienen la ventaja de que fomentan la programación modular y la creación de nuevas funciones.

### PROGRAMACIÓN DECLARATIVA

Este estilo de programación utiliza el principio de razonamiento lógico, basado en la deducción, para resolver problemas. Normalmente se ha utilizado para simular sistemas bajo ciertas condiciones, es decir, para simular situaciones en las que se repite el cálculo de ciertos parámetros que a su vez se basan en valores previamente obtenidos (por ejemplo, economía, política, etc.).

La programación declarativa implica que el programador describa el problema, a diferencia de la programación imperativa donde el programador debe definir el algoritmo que lo resuelve. En efecto, el programa es un conjunto de declaraciones lógicas sobre el resultado que debería obtenerse en vez de detallar cómo podría conseguirse el mismo. Las premisas o declaraciones incluidas en el programa se aplican a los datos, obteniéndose así las soluciones posibles.

Ejemplos de estos lenguajes son GPSS, Prolog, CLP y SQL.

### EJEMPLO 12.9

En la Figura 12.18 se define una regla de deducción lógica (si (A es B) y (B tiene C), entonces (A tiene C)), de forma que aplicándola a los hechos 1 y 2 se obtiene el hecho 3.

<i>Premisa:</i>	
<b>Si (A es B) y (B tiene C), entonces (A tiene C)</b>	
<i>Ejecución de la premisa con datos:</i>	
Hecho 1: Marta es hija	(A es B)
Hecho 2: Una hija tiene padres	(B tiene C)
<i>Resultado de la ejecución:</i>	
Hecho 3: Marta tiene padres	(A tiene C)

**Figura 12.18.** Ejemplo de esquema de funcionamiento de un lenguaje declarativo.

En Prolog el esquema anterior se describiría así:

```
hija (Marta)
padres (hija)
```

Y para obtener un resultado habría que preguntar:

```
¿ - padres (Marta)
```

y el programa daría una respuesta afirmativa.

### PROGRAMACIÓN GUIADA POR EVENTOS

En la programación guiada por eventos el programa principal es un lazo indefinido que reacciona en respuesta a la recepción de incidencias generadas en orden e instantes de tiempo no predecibles (**eventos asíncronos**). Cuando se produce un evento el control del programa sale del lazo indefinido ejecutando el procedimiento asociado al evento, para, posteriormente, volver a entrar en el lazo indefi-

nido. Los eventos se producen, por ejemplo, cuando se pulsa una tecla de un teclado o de un ratón, cuando se desplaza una ventana en la pantalla del monitor, cuando llega un mensaje de Internet, cuando se recibe una señal de un sensor, etc. Entre los lenguajes más destacados para desarrollar programación dirigida por eventos se encuentran el Visual Basic y Java.

## PROGRAMACIÓN CONCURRENTE

La programación concurrente está ligada a la computación paralela. En efecto, como se ha visto en el Capítulo 9, un programa en el momento de ejecutarse puede descomponerse en pequeños procesos denominados **hebras** que se ejecutan concurrentemente en el computador. También un programa puede descomponerse en distintas unidades y ejecutarse éstas en paralelo en distintos procesadores. En ambos casos, tanto las hebras como las unidades ejecutables concurrentemente deben poder interactuar y compartir información. En definitiva, la programación concurrente trata de la definición de procesos cooperativos que se ejecutan independientemente pero pudiendo ocasionalmente compartir o intercambiar información. Algunos de los lenguajes con los que se puede desarrollar programación concurrente son el Pascal Concurrente, Java, Ada95 y FORTRAN de altas prestaciones (FORTRAN HP).

### 12.5.2. DOMINIOS DE APLICACIÓN DE LOS LENGUAJES DE PROGRAMACIÓN

Según indicamos en la Sección 12.1, los lenguajes de alto nivel tratan de salvar la gran distancia semántica que existe entre el código máquina y el **lenguaje natural** que utilizamos las personas. Desafortunadamente es extremadamente complejo desarrollar un sistema que pudiese traducir automáticamente cualquier algoritmo descrito dentro del dominio de cualquier actividad humana a código máquina. Debido a ello los lenguajes de alto nivel surgieron como herramienta para desarrollar programas preferentemente para un determinado campo de aplicación. Los ejemplos más notables se tienen en los dos primeros lenguajes de programación reconocidos universalmente:

- El FORTRAN (*FORmula TRANslation*, o “traductor de fórmulas”), que se diseñó, como su nombre indica, para poder representar expresiones matemáticas, estando su campo de aplicación, por tanto, orientado a aplicaciones científico-técnicas.
- El COBOL (*COmmon Business Oriented Language*), que se desarrolló en 1959 para el ámbito comercial y de gestión de la información, y que puede considerarse el lenguaje de programación más utilizado en la historia.

No obstante a lo indicado anteriormente, muchos lenguajes son de utilidad en distintos campos de aplicación. Los dominios más relevantes de aplicación de los lenguajes de programación son los siguientes:

- **Aplicaciones científico-técnicas.** En este tipo de aplicaciones normalmente se busca la realización de cálculos complejos con gran precisión y la obtención de soluciones en tiempos razonables.
- **Aplicaciones de gestión de información.** En este tipo de aplicaciones fundamentalmente se trata de definir bases de datos o archivos, recuperar la información de archivos o bases de datos y generación de informes. Entre estas aplicaciones se incluyen: contabilidad, control de inventario, gestión de personal, nóminas, gestión bancaria, bases de datos accesibles por Internet (comercio electrónico), etc.
- **Aplicaciones de inteligencia artificial.** En este tipo de aplicaciones se busca la manipulación de símbolos, inferencia lógica, representación funcional, etc.



- **Aplicaciones de programación de sistemas.** Este dominio hace referencia al desarrollo de programas o módulos del sistema operativo, de gestión de red, compiladores, etc. El software de sistemas debe estar lo mejor adaptado posible a la arquitectura del computador para el que se desarrolla.
- **Aplicaciones para web.** Este tipo de aplicaciones requiere la implementación de modelos interactivos, que faciliten la relación usuario sistema.

En la Tabla 12.4 se indican los principales lenguajes de programación utilizados en cada dominio de aplicación.

Dominio	Científico-técnicas	Gestión de información	Inteligencia artificial	Programación de sistemas	Web
Programación imperativa	FORTRAN 90	Cobol		Ensamblador, C	
Programación funcional		Lisp, Scheme, ML, Haskell			
Programación orientada a objetos					C++, Java
Programación declarativa		RPG, SQL	CLP, Prolog		
Programación guiada por eventos		Java, Tcl/Tk			Visual Basic, Tcl/Tk
Programación concurrente	FORTRAN HP				

**Tabla 12.4.** Dominios y estilos de programación.

## 12.6. Conclusiones

En este capítulo hemos presentado conceptos básicos relacionados con los lenguajes de programación. Se ha hecho especial hincapié en los lenguajes de alto nivel, que nos permiten describir los algoritmos en un lenguaje más próximo al lenguaje propio del dominio de la aplicación que se trata de implementar que al lenguaje de la máquina (ceros y unos). Una vez redactado un programa en un lenguaje de alto nivel es necesario traducirlo a código máquina, operación que se realiza con compiladores o intérpretes.

A lo largo del capítulo hemos analizado los distintos elementos con los que se construye un programa, las distintas fases que se siguen para traducir un programa de un lenguaje a otro, los pasos necesarios para ejecutar un programa y los distintos tipos y dominios de aplicación de los lenguajes de programación.

## Test



**T12.1.** Los lenguajes informáticos independientes del computador también son conocidos como lenguajes:

- a) Máquina.
- b) Ensambladores.
- c) Macroensambladores.
- d) De alto nivel.

**T12.2.** Los lenguajes ensambladores también son de tipo:

- a) Natural.
- b) Simbólico.
- c) De alto nivel.
- d) Máquina.

**T12.3.** El lenguaje con el que se programa directamente a un Pentium IV es de tipo:

- a) Natural.
- b) Simbólico.
- c) De alto nivel.
- d) Máquina.

**T12.4.** El español y el inglés son lenguajes de tipo:

- a) Natural.
- b) Simbólico.
- c) De alto nivel.
- d) Máquina.

**T12.5.** Los lenguajes cuya sintaxis es más cercana al lenguaje humano y por tanto más fácil de utilizar son los lenguajes:

- a) Ensambladores.
- b) Simbólicos.
- c) Máquina.
- d) De alto nivel.

**T12.6.** Un procesador puede interpretar y ejecutar directamente las instrucciones de un programa:

- a) Escrito en su lenguaje máquina.
- b) En lenguaje de alto nivel, siempre que sea de tipo intérprete.
- c) En escrito en cualquier lenguaje máquina.
- d) En pseudocódigo.

**T12.7.** Las sentencias que dan información al traductor sobre los tipos y estructuras de datos que se utilizarán en un programa son:

- a) Sentencias de cabecera.
- b) Sentencias declarativas.
- c) Sentencias imperativas.
- d) Comentarios.

**T12.8.** Las sentencias que describen los pasos del algoritmo a ejecutar por el programa son:

- a) Sentencias reservadas.
- b) Sentencias declarativas.

- c) Sentencias imperativas.
- d) Identificadores.

**T12.9.** Los valores lógicos se obtienen como resultado de evaluar:

- a) Expresiones algebraicas.
- b) Expresiones de caracteres.
- c) Expresiones de relación y expresiones lógicas.
- d) Sólo expresiones lógicas.

**T12.10.** Una expresión aritmética combina:

- a) Variables y constantes de tipo numérico con operadores aritméticos.
- b) Variables y constantes de tipo lógico con operadores lógicos.
- c) Variables y constantes de tipo carácter con operadores de relación.
- d) Variables y constantes de tipo numérico con operadores de relación.

**T12.11.** Una expresión lógica es aquella que combina:

- a) Variables y constantes de tipo carácter con operadores de relación.
- b) Variables y constantes de tipo lógico con operadores lógicos.
- c) Variables y constantes de tipo numérico con operadores de relación.
- d) O lo indicado en b) o lo indicado en c).

**T12.12.** Una estructura *if-then* sirve para implementar:

- a) Decisiones sencillas.
- b) Decisiones dobles.
- c) Ciclos repetitivos.
- d) Ciclos condicionales.

**T12.13.** Una estructura *if-then-else* sirve para implementar:

- a) Decisiones sencillas.
- b) Decisiones dobles.
- c) Ciclos repetitivos.
- d) Ciclos condicionales.

**T12.14.** Una estructura *case* sirve para implementar:

- a) Decisiones dobles.
- b) Decisiones múltiples.
- c) Ciclos repetitivos.
- d) Ciclos condicionales.

**T12.15.** Una estructura *for* sirve para implementar:

- a) Decisiones sencillas.
- b) Decisiones dobles.
- c) Ciclos repetitivos.
- d) Ciclos condicionales.

**T12.16.** Una estructura *while* sirve para implementar:

- a) Decisiones sencillas.
- b) Decisiones dobles.
- c) Ciclos repetitivos.
- d) Ciclos condicionales.

**T12.17.** Una variable local:

- a) Sólo tiene sentido y puede utilizarse dentro del procedimiento.
- b) Puede utilizarse por distintos programas.
- c) Es lo mismo que un parámetro real.
- d) Es lo mismo que un parámetro formal.

**T12.18.** Los valores de los parámetros reales se asignan:

- a) En el programa que llama al procedimiento o a la función.
- b) En el interior del procedimiento o la función.
- c) Son variables locales, por lo que sólo tienen sentido dentro del procedimiento o la función.
- d) Se pueden dar indistintamente en el programa de llamada como en el interior del subprograma (son variables globales).

**T12.19.** Los valores de los parámetros formales se establecen:

- a) En el programa que llama al procedimiento o a la función.
- b) En el momento de llamar al procedimiento o función.
- c) Son variables locales, por lo que sólo tienen sentido dentro del procedimiento o la función.
- d) Se pueden dar indistintamente en el programa de llamada como en el interior del subprograma (son variables globales).

**T12.20.** Un programa fuente es un programa:

- a) Escrito en un lenguaje de alto nivel.
- b) Cuyas instrucciones son consideradas como datos de entrada por un programa traductor.
- c) Escrito por el fabricante del computador.
- d) A partir del que se generan múltiples resultados.

**T12.21.** Un programa objeto es cualquier programa:

- a) Generado por un compilador.
- b) Realizado con un lenguaje orientado a objetos.
- c) Modificable por el usuario.
- d) Generado por un intérprete.

**T12.22.** El análisis gramatical es una etapa de compilación de un lenguaje de alto nivel cuyo objetivo es:

- a) Descomponer el programa fuente en sus elementos constitutivos.
- b) Extraer la estructura de cada instrucción, reconociendo los símbolos terminales del lenguaje, y el significado de las distintas construcciones sintácticas y elementos terminales.
- c) Optimizar el programa objeto.
- d) Sintetizar el programa objeto.

**T12.23.** La etapa de generación de código de un proceso compilación de un lenguaje de alto nivel tiene como objetivo:

- a) Descomponer el programa fuente en sus elementos constitutivos.
- b) Extraer la estructura de cada instrucción, reconociendo los símbolos terminales del lenguaje.
- c) Extraer el significado de las distintas construcciones sintácticas y elementos terminales.
- d) Sintetizar el programa objeto.

**T12.24.** La etapa de optimización de un proceso compilación de un lenguaje de alto nivel tiene como objetivo:

- a) Descomponer el programa fuente en sus elementos constitutivos.
- b) Extraer la estructura de cada instrucción, reconociendo los símbolos terminales del lenguaje.
- c) Extraer el significado de las distintas construcciones sintácticas y elementos terminales.
- d) Mejorar la eficiencia del programa objeto a generar.

**T12.25.** Las tablas de símbolos son generadas por los compiladores para contener:

- a) Los operadores que intervienen en el programa, junto con información adicional asociada a ellos.
- b) Los comentarios que simbolizan o describen lo que realizan las instrucciones.
- c) Los nemónicos de las instrucciones máquina del programa.
- d) Los nombres de las variables y constantes, junto con información adicional asociada a ellas.

**T12.26.** En la programación imperativa:

- a) Se utiliza el principio de razonamiento lógico, basado en la deducción, para resolver problemas.
- b) Se consideran los conjuntos de datos como unidades activas, a las que se asocian atributos y funciones.
- c) El programador describe con instrucciones el método a seguir para resolver el problema.
- d) Los programas se realizan mediante la utilización y definición de relaciones.

**T12.27.** En la programación declarativa:

- a) Se utiliza el principio de razonamiento lógico, basado en la deducción, para resolver problemas.
- b) Se consideran los conjuntos de datos como unidades activas, a las que se asocian atributos y funciones.
- c) El programador describe con instrucciones el método a seguir para resolver el problema.
- d) Los programas se realizan mediante la utilización y definición de relaciones.

**T12.28.** En la programación funcional:

- a) Se utiliza el principio de razonamiento lógico, basado en la deducción, para resolver problemas.
- b) Se consideran los conjuntos de datos como unidades activas, a las que se asocian atributos y funciones.
- c) El programador describe con instrucciones el método a seguir para resolver el problema.
- d) Los programas se realizan mediante la utilización y definición de relaciones.

**T12.29.** En la programación orientada a objetos:

- a) Se utiliza el principio de razonamiento lógico, basado en la deducción, para resolver problemas.
- b) Se consideran los conjuntos de datos como unidades activas, a las que se asocian atributos y funciones.
- c) El programador describe con instrucciones el método a seguir para resolver el problema.
- d) Los programas se realizan mediante la utilización y definición de relaciones.

**T12.30.** El lenguaje de programación Cobol está orientado a:

- a) Aplicaciones científicas.
- b) Aplicaciones de tratamiento de textos.
- c) Aplicaciones de inteligencia artificial.
- d) Aplicaciones comerciales.

**T12.31.** El lenguaje de programación FORTRAN está orientado a:

- a) Aplicaciones científicas.
- b) Aplicaciones de tratamiento de textos.
- c) Aplicaciones de inteligencia artificial.
- d) Aplicaciones comerciales.

**T12.32.** El lenguaje de programación C se diseñó para:

- a) Aplicaciones de tratamiento de textos.
- b) Aplicaciones de inteligencia artificial.
- c) Aplicaciones comerciales.
- d) Programación de sistemas.

**T12.33.** El lenguaje de programación FORTRAN es de tipo:

- a) Imperativo.
- b) Funcional.
- c) Declarativo.
- d) Orientado a objetos.

**T12.34.** El lenguaje de programación Cobol es de tipo:

- a) Imperativo.
- b) Funcional.

- c) Declarativo.
- d) Orientado a objetos.

**T12.35.** El lenguaje de programación LISP es de tipo:

- a) Imperativo.
- b) Funcional.
- c) Declarativo.
- d) Orientado a objetos.

**T12.36.** El lenguaje de programación BASIC es de tipo:

- a) Imperativo.
- b) Funcional.
- c) Declarativo.
- d) Orientado a objetos.

**T12.37.** El lenguaje de programación PROLOG es de tipo:

- a) Imperativo.
- b) Funcional.
- c) Declarativo.
- d) Orientado a objetos.

**T12.38.** El lenguaje de programación Ada es de tipo:

- a) Imperativo.
- b) Funcional.
- c) Declarativo.
- d) Orientado a objetos.

**T12.39.** El lenguaje de programación C es de tipo:

- a) Imperativo.
- b) Funcional.
- c) Declarativo.
- d) Orientado a objetos.

**T12.40.** El lenguaje de programación C++ es de tipo:

- a) Imperativo.
- b) Funcional.
- c) Declarativo.
- d) Orientado a objetos.

## Problemas resueltos



### CONCEPTO DE LENGUAJE DE PROGRAMACIÓN

**P12.1.** Se tiene la siguiente sentencia en un lenguaje de alto nivel:

$$z = a + 2*b$$

Determinar el número de instrucciones máquina que generaría un compilador a lenguaje máquina de CODE2 al traducir la sentencia anterior, suponiendo que  $a$ ,  $b$  y  $z$  son enteros positivos, a los que se les ha asignado las posiciones de memoria 004A, 004B y 004C, respectivamente.

#### SOLUCIÓN

Utilizamos los siguientes registros:  $r3$  para almacenar  $a$ ,  $r4$  para almacenar  $b$  y  $r5$  para almacenar temporalmente a  $z$ . El programa podría ser el siguiente (Capítulo 4):

```

LLI rD,4A      2D4A
LD r3,[00]     0300
LLI rD,4B      2D4B
LD r4,[00]     0400
ADDS rF,r4,r4   6F33
ADDS rF,rF,r3   6FF3
LLI rD,4C      2D4C
ST [00], rF     1F00

```

o en código binario:

```

0010 1101 0100 1010
0000 0011 0000 0000
0010 1101 0100 1011
0000 0100 0000 0000
0110 1111 0100 0100
0110 1111 1111 0011
0010 1101 0100 1100
0001 1111 0000 0000

```

Luego el número de instrucciones es 8.

## ELEMENTOS DE UN PROGRAMA EN UN LENGUAJE DE ALTO NIVEL

**P12.2.** En las siguientes sentencias:

```

if (x+y == 0) {
    i=i+1;
    else a=log(z) ;
}

```

identificar:

- a) Las palabras reservadas que contiene.
- b) Las variables y constantes que contiene.
- c) Los tipos de operadores que contienen.
- d) Los tipos de expresiones que contienen.
- e) Las estructuras de control que contiene.

### SOLUCIÓN

- a) Palabras reservadas: **if** y **else**
- b) Variables: **x, y, i, a, z**. Literales: **0, 1**
- c) Tipos de operadores: asignación (**=**); relación (**==**); aritméticos (**+**)
- d) Tipos de expresiones: lógica (**x + y == 0**), aritmética (**i = i + 1**)
- e) Las estructuras de control que contiene: **if-else**

**P12.3.** Las siguientes sentencias corresponden a un fragmento de un programa escrito en Perl:

```

print "Introduzca el valor del producto:";
$valor = <STDIN>;
$precio_iva = $valor * 1,07;
print "Precio con IVA = $precio_iva\n";

```

identificar:

- a) Las palabras reservadas que contiene.
- b) Las variables.
- c) Constantes que contiene.
- d) Los tipos de operadores que contienen.
- e) Los tipos de expresiones que contienen.
- f) Los tipos de sentencia que contiene.
- g) Los formatos involucradas en las sentencias de E/S.
- h) Los formatos involucradas en las sentencias de E/S.

## SOLUCIÓN

- a) Palabras reservadas: print, \n.
- b) Variables: \$valor, \$precio\_iva;
- c) Constantes: 1,07.
- d) Operadores: asignación (=), producto (\*).
- e) Expresiones: aritméticas (\$valor = <STDIN>; \$precio\_iva = \$valor \* 1,07).
- f) Sentencias:
  - asignación (\$valor = <STDIN>; \$precio\_iva = \$valor \* 1,07).
  - salida (print).
- g) Formatos en sentencias de E/S:
  - “Introduzca el valor del producto: ”;
  - “Precio con IVA = \n”
- h) Variables en sentencias de E/S.
  - \$precio\_iva

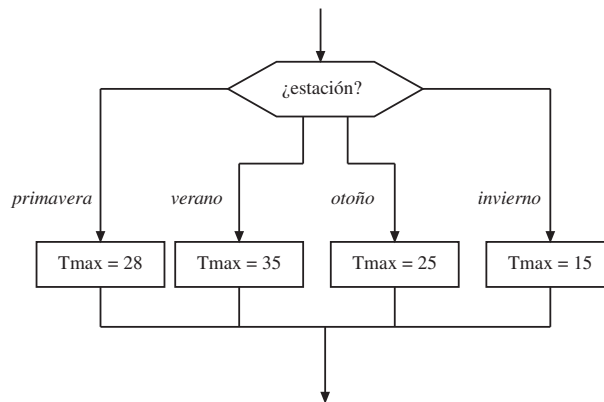
**P12.4.** Dado el siguiente extracto de programa:

```
switch (estacion)
{case "primavera": Tmax= 28;
case "verano": Tmax=35;
case "otoño": Tmax= 25;
case "invierno": Tmax=15 ;
};
```

- a) Realizar un organigrama del mismo.
- b) Construir una versión nueva del mismo sustituyendo la instrucción *case* por instrucciones *if-then-else*.

## SOLUCIÓN

- a) El organigrama es el que se muestra en la Figura 12.19.



**Figura 12.19.** Organigrama del Problema 12.4.

- b) Con la construcción *if-then-else* el programa quedaría así:

```
if (estacion == "primavera") {
    Tmax = 28;
else if (estacion == "verano")
    Tmax = 35;
else if (estacion == "otoño")
    Tmax = 25;
else if (estacion == "invierno")
    Tmax = 15;
}
```

**P12.5.** Se tiene el siguiente fragmento de código:

22A0		saltar a 22A8
22A2		J = J + 1
22A4		K = K + J
22A6		saltar a 22AE
22A8		Si Z=1 saltar a 22AC
22AA		Saltar a 22A2
22AC		K = K - 1

- Realizar un organigrama.
- Expresarlo con una sentencia *if-then-else*.

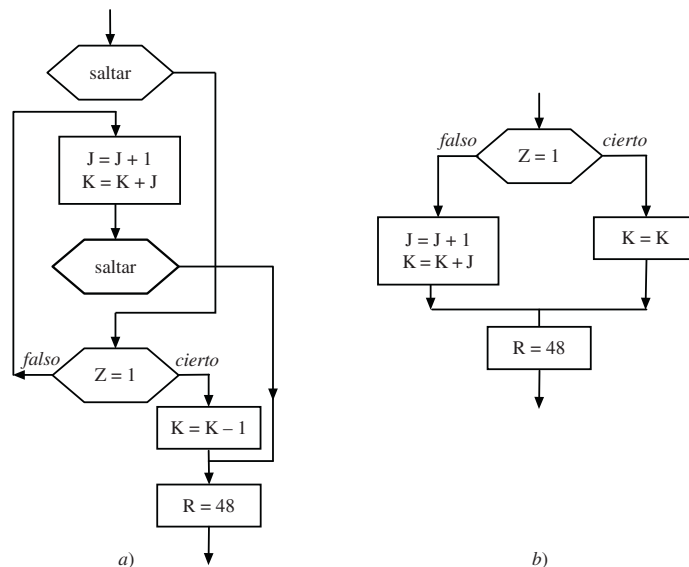
#### SOLUCIÓN

- En la Figura 12.20a) se muestra un organigrama del fragmento de código dado en el enunciado del problema. En la Figura 12.20b) se muestra un organigrama simplificado del mismo algoritmo.
- De la Figura 12.20b) se deduce que el nuevo programa sería:

```

if (Z == 1) {
    Z = K + 1;
else
    J = J + 1;
    K = K + J;
}
R = 48;

```



**Figura 12.20.** Organigrama: a) del Problema 12.5 y b) simplificación.

**P12.6.** Implementar en lenguaje máquina de CODE2 la estructura siguiente:

```

for (int i = 17; i < 25; i++) {
    k = k + 1;
    z = z + 12;
}

```

#### SOLUCIÓN

Hacemos la siguiente asignación de registros:

Variable o literal	Registro de CODE2
1	r1
i (valor inicial H'11)	r2
25 (límite del lazo)	r3
12	r4
K	r5
Z	r6
variable auxiliar	rF

El programa sería:

```

a  LLI  r1,01      ; constante 1
    LLI  r2,11     ; iniciación de i
    LLI  r3,19     ; límite del lazo
    LLI  r4,12     ; almacenar cte. 12
    ADDS r5,r5,r1   ; actualización de K
    ADDS r6,r6,r4   ; actualización de Z
    ADDS r2,r2,r1   ; incremento de i
    SUBS rF,r2,r3   ; i - 25
    LLI  rF,r2,r3   ; dirección de salto
    JS  a          ; saltar si i<25

```

**P12.7.** Suponga que dispone de una función Mayor(x,y) que proporciona el mayor de dos valores numéricos (x e y). ¿Cómo obtendría utilizando dicha función el valor mayor de cuatro valores: a, b, c y d?

**SOLUCIÓN**

Como los argumentos de una función son otra función, el valor mayor (v\_mayor) se puede obtener así:

```
v_mayor = Mayor(Mayor(a,b),Mayor(c,d))
```

## EL PROCESO DE TRADUCCIÓN

**P12.8.** Sabiendo que en la notación BNF el signo | denota alternativas posibles (operador OR booleano) y que la concatenación se efectúa sencillamente poniendo los símbolos o formas una junto a otra, y suponiendo que se tiene la siguiente construcción (gramática):

```

DígitoDecimal → 0|1|2|3|4|5|6|7|8|9
Entero → DígitoDecimal | Entero DígitoDecimal

```

explicar cómo se forma la categoría gramatical de los enteros.

**SOLUCIÓN**

La primera construcción indica que un dígito decimal es o un 0, o un 1, o un 2, ..., o un 9.

La segunda regla indica que un entero puede ser o un dígito aislado o un dígito decimal concatenado al final de un entero. Puede observarse que es una regla recursiva. Por ejemplo, 5734 es un número entero ya que lo podemos construir así:

```

Paso 1      entero → DígitoDecimal      5
Paso 2      entero → entero DígitoDecimal 57 (5 es el entero y 7 el dígito)
Paso 3      entero → entero DígitoDecimal 573 (57 es el entero y 3 el dígito)
Paso 4      entero → entero DígitoDecimal 5734 (573 es el entero y 4 el dígito)

```

**P12.9.** Sabiendo que en la notación BNF el signo | denota alternativas posibles (operador OR booleano) y que la concatenación se efectúa sencillamente poniendo los símbolos o formas una junto a otra, y suponiendo que se tiene la siguiente construcción (gramática):

```

Decimal → 0|1|2|3|4|5|6|7|8|9
Letra → A|B|C|D|...|Z|a|b|c|d|...|z
Nombre_var → Letra | Nombre_var Decimal | Nombre_var Letra

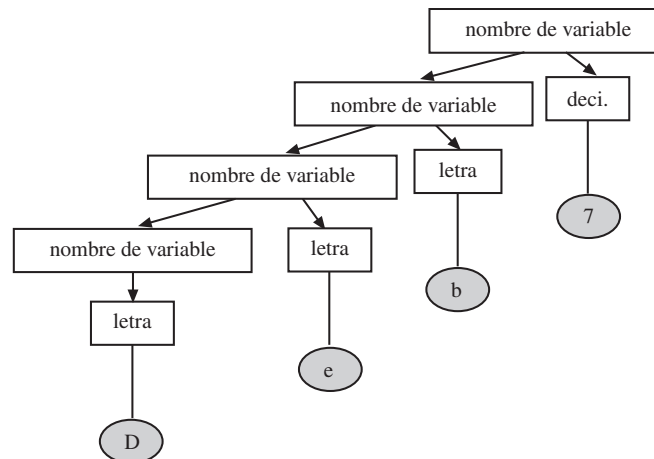
```



obtener el árbol de análisis sintáctico de las variables *Deb7*, *7Upa* para comprobar si son nombres de variables (Nombre\_var) sintácticamente correctos.

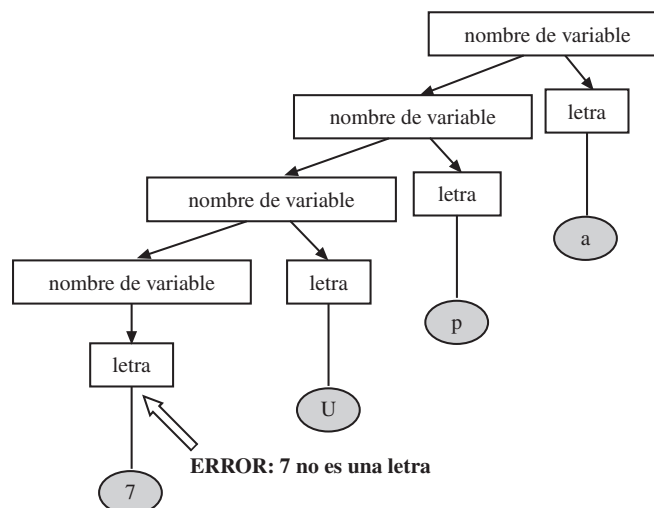
### SOLUCIÓN

En la Figura 12.21 se comprueba que el árbol sintáctico *Deb7* cumple con las reglas.



**Figura 12.21.** Árbol sintáctico de *Deb7*.

Sin embargo, como se muestra en la figura siguiente (Figura 12.22), *7Upa* no cumple la regla.



**Figura 12.22.** Árbol sintáctico de *7Upa*.

**P12.10.** Obtener los diagramas sintácticos de las siguientes reglas dadas en notación BNF:

factor  $\rightarrow a \mid b \mid c$

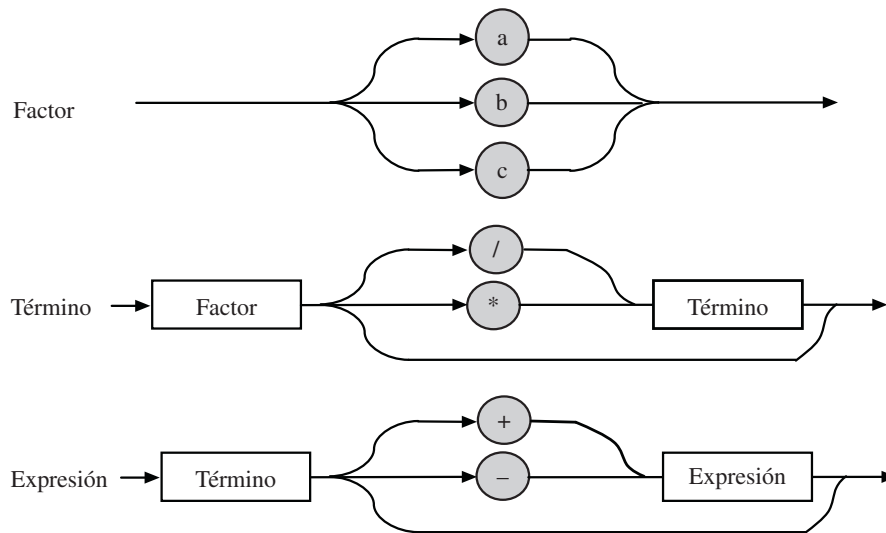
término  $\rightarrow$  factor  $\mid$  factor / término  $\mid$  factor\*término

expresión  $\rightarrow$  término  $\mid$  término + expresión  $\mid$  término - expresión

donde +, -, \* y / son los operadores de suma, resta, multiplicación y división, respectivamente.

### SOLUCIÓN

La solución se muestra en la Figura 12.23.

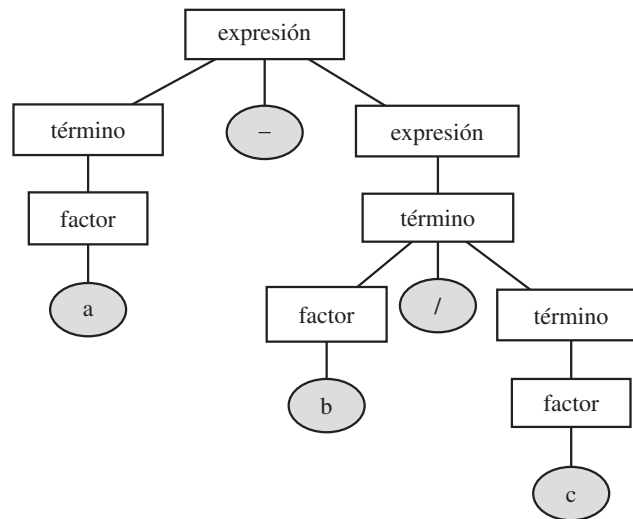


**Figura 12.23.** Diagramas sintácticos de las reglas dadas en el enunciado del Problema 12.10.

**P12.11.** Obtener el árbol de léxico para comprobar si la expresión  $a - b/c$  cumple las reglas para expresiones aritméticas definidas en el Problema 12.10.

**SOLUCIÓN**

En la Figura 12.24 se muestra el árbol de léxico solicitado.



**Figura 12.24.** Árbol sintáctico de la expresión  $a - b/c$ .

**P12.12.** Obtener el árbol de léxico para comprobar si la sentencia de asignación aritmética

$$z = x + z * v - y/w$$

cumple las reglas definidas en la Figura 12.25.

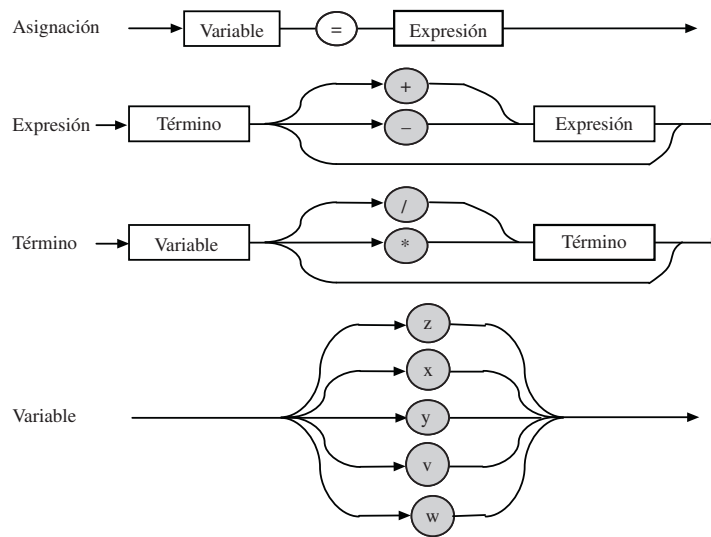


Figura 12.25. Ejemplos de diagramas sintácticos.

### SOLUCIÓN

La sentencia cumple las reglas, como se puede comprobar en la Figura 12.26.

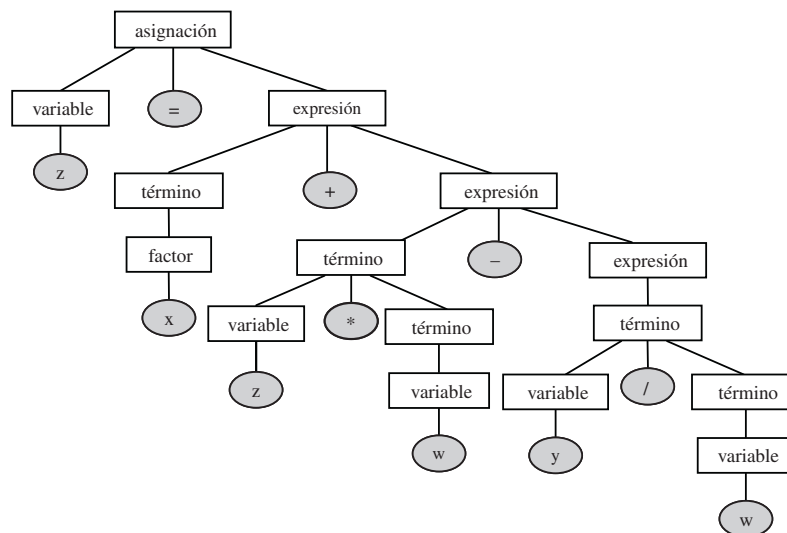


Figura 12.26. Árbol sintáctico de la expresión  $z = x + z * v - y/w$  (Problema 12.12).

**P12.13.** Se tiene el siguiente código:

```
i = 5000;
while (i ≥ 8)
{
    r = r + r;
    i = i - 1;
    k = p - 1;
};
j = 32;
```

- a) Indicar la mejora que podría introducir el optimizador de un compilador.
- b) Cuantificar en lo posible los efectos de la mejora.
- c) Indicar, justificando la respuesta, si un intérprete podría realizar esta optimización.

## SOLUCIÓN

- a) Mejora que podría introducir el optimizador de un compilador: sacar, a antes o a después del lazo, la sentencia  $k = p - 1$ . No tiene sentido que esté dentro del lazo, ya que siempre se asigna a  $k$  el mismo valor.
- b) Cuantificar en lo posible los efectos de la mejora: la sentencia  $k = p - 1$  se tendría que ejecutar una sola vez, en lugar de  $5.000 - 7 = 4.993$ . Esta reducción supone disminuir el tiempo de ejecución del programa.
- c) Un intérprete convencional no podría realizar esta optimización, ya que optimiza sólo sentencia a sentencia, y no considera fragmentos o bloques de instrucciones globalmente.

## TIPOS DE LENGUAJES DE PROGRAMACIÓN

**P12.14.** Suponga que en un lenguaje funcional, tal como LISP, se dispone de las funciones que se indican en la Tabla 12.5. Obtener una función que genere el valor medio de los datos de entrada.

- **Sumar** entradas.
- **Restar** entradas (*el primer dato es el minuendo y el segundo el sustraendo*).
- **Multiplicar** entradas.
- **Dividir** entradas (*el primer dato es el dividendo y el segundo el divisor*).
- **Contar** entradas (*cuenta el número de entradas*).
- **Ordenar** lista\_entrada (*ordena los elementos de la lista de menor a mayor*).
- **Primero** lista\_entrada (*proporciona el primer elemento de la lista de entrada*).
- **Último** lista\_entrada (*proporciona el último elemento de la lista de entrada*).
- **Elipri** lista\_entrada (*elimina el primer elemento de la lista de entrada*).

**Tabla 12.5.** Ejemplos de funciones de un lenguaje funcional.

## SOLUCIÓN

Para obtener el valor medio tenemos que dividir la suma de los valores de entrada entre el número de entradas. Es decir:

- suma de datos de entrada: **suma** datos,
- número de entradas: **contar** datos,
- valor medio:

**media** datos  $\leftarrow$  (**división** (**suma** datos) (**contar** datos))

**P12.15.** Suponga que en un lenguaje funcional, tal como LISP, se dispone de las funciones que se indican en la Tabla 12.5. Obtener una función que genere el valor mínimo de los elementos de una lista.

## SOLUCIÓN

Con las funciones de que disponemos, podemos primero ordenar la lista y luego extraer el primer elemento; es decir, la función sería:

**mínimo** lista  $\leftarrow$  **primero**(**ordenar** lista)

**P12.16.** Suponga que en un lenguaje declarativo, tal como Prolog, se tienen las reglas que se indican a continuación:

Regla	Regla en el lenguaje
$(\text{padre}(X,Y) \text{ AND } \text{padre}(Y,Z)) \Rightarrow \text{abuelo}(X,Z);$	$\text{abuelo}(X,Z) \text{ :- } \text{padre}(X,Y) , \text{padre}(Y,Z)$
$(\text{padre}(X,Y) \text{ AND } \text{padre}(X,Z)) \Rightarrow \text{hermano}(Y,Z)$	$\text{hermano}(Y,Z) \text{ :- } \text{padre}(X,Y), \text{padre}(X,Z)$
$\text{hermano}(X,Y) \Rightarrow \text{hermano}(Y,X)$	$\text{hermano}(X,Y) \text{ :- } \text{hermano}(Y,X)$

Donde la construcción **padre**(X,Y) significa que X es el padre de Y. Por otra parte, suponga que la pregunta **padre**(X,a) genera como respuesta todos los hijos de X. Si se incluyen los siguientes hechos:

```
padre (Alberto, Marta)
padre (Matías, Alberto)
hermano(Beatriz,Marta)
hermano (Enrique, Alberto)
```

qué resultados que generaría el programa a las siguientes preguntas:

- a)  $(\text{¿ - padre (Beatriz)})$
- b)  $\text{hermano(Marta,X)}$
- c)  $\text{padre(Matías,X)}$

SOLUCIÓN

- a)  $(\text{¿ - padre (Beatriz)})$ : **sí** (está definido en los hechos)
- b)  $\text{hermano(Marta,X)}$ : **Beatriz** (todos los hermanos de Beatriz)
- c)  $\text{padre(Matías,X)}$ : **Alberto, Enrique** (todos los hijos de Matías)

## Problemas propuestos



### CONCEPTO DE LENGUAJE DE PROGRAMACIÓN

**P12.17.** Realizar una tabla en la que las columnas correspondan a *Característica del lenguaje*, *Lenguaje máquina*, *Lenguaje ensamblador* y *Lenguaje de alto nivel*, y las filas a las siguientes características:

- Transportabilidad. Esta característica se refiere a si el lenguaje depende o no del procesador del computador.
- Variables simbólicas. Esta característica se refiere a si se pueden utilizar nombres simbólicos de variables o sólo direcciones de memoria o de registros.
- Expresividad. Esta característica se refiere a la proximidad o no al lenguaje del dominio de la aplicación.
- Riqueza semántica. Esta característica se refiere a si las acciones de las instrucciones son complejas o simples.
- Comentarios. Esta característica se refiere a si el programador puede incluir o no comentarios dentro del programa.

Indicar en las celdas de la tabla si los lenguajes citados reúnen esas características y en qué grado.

**P12.18.** Se tienen las siguientes sentencias en un lenguaje de alto nivel:

```
for (i = 0: i < n: i++)
    i = i + 1;
```

Determinar el número de instrucciones máquina que generaría un compilador a lenguaje máquina de CODE2 a partir de las sentencias anteriores.

### ELEMENTOS DE UN PROGRAMA EN UN LENGUAJE DE ALTO NIVEL

**P12.19.** Construir dos sentencias donde el significado del operador “=” sea distinto.

**P12.20.** En las siguientes sentencias (C):

```
while (a - b = 0)
{
    i=i+1;
    if (i=50) b=b+1;
}
textol=CON(texto0,"sin IVA" );
```

Identificar:

- a) Las palabras reservadas que contiene.
- b) Las variables y constantes que contiene.
- c) Los tipos de operadores que contienen.
- d) Los tipos de expresiones que contienen.
- e) Las estructuras de control que contienen.

**P12.21.** En las siguientes sentencias (C):

```
void main(){
    float lado,area;
    printf("Introduzca el valor del lado del cuadrado: ");
    scanf("%f",&lado);
    area=lado*lado;
    printf("El area del cuadrado es: %f cm2",area);
}
```

Identificar:

- a) Las palabras reservadas que contiene.
- b) Las variables y constantes que contiene.
- c) Los tipos de operadores que contienen.
- d) Los tipos de sentencias que contienen.

**P12.22.** Implementar en ensamblador de CODE2 la estructura siguiente:

```
if (J≤I) J=J+1;
    else I=I-1;
K=K+12;
```

**P12.23.** Se tiene el siguiente fragmento de código:

```
5AA0    I=0
5AA2    I=I+1
5AA4    K=I*I
5AA6    llamar a Procedimiento Acción1
5AA8    si I=48 ir a 5AA2
```

- a) Realizar un organigrama.
- b) Expresarlo con un estructura *if-then-else*.
- c) Expresarlo con una estructura *while*.
- d) Expresarlo con una estructura *for*.
- e) Expresarlo con una estructura *case*.

**P12.24.** Implementar en ensamblador de CODE2 el siguiente fragmento de código:

```
i = 50;
while (I≥8)
{r = r+r;
 i = i-1;
};
j = 32;
```

**P12.25.** Implementar en ensamblador de CODE2 el siguiente extracto de programa:

```
switch (i)
{case 3: Tmax = 28;
 case 4: Tmax = 35;
 case 5: Tmax = 25;
 case 6: Tmax = 15;
};
```

**P12.26.** Transformar el siguiente código en una función (C):

```
void main(){
    float lado,area;
    printf("Introduzca el valor del lado del cuadrado: ");
    scanf("%f",&lado);
    area=lado*lado;
    printf("El area del cuadrado es: %f cm2",area);
}
```

## EL PROCESO DE TRADUCCIÓN

- P12.27.** Sabiendo que en la notación BNF el signo | denota alternativas posibles (operador OR booleano) y que la concatenación se efectúa sencillamente poniendo los símbolos o formas una junto a otra, y suponiendo que se tiene la siguiente construcción (gramática):

$$\begin{aligned}\text{DígitoDecimal} &\rightarrow 0|1|2|3|4|5|6|7|8|9 \\ \text{Entero} &\rightarrow \text{DígitoDecimal} \mid \text{Entero DígitoDecimal}\end{aligned}$$

Obtener el árbol gramatical para comprobar si 7231 es un entero.

- P12.28.** Sabiendo que en la notación BNF el signo | denota alternativas posibles (operador OR booleano) y que la concatenación se efectúa sencillamente poniendo los símbolos o formas una junto a otra, y suponiendo que se tiene la siguiente construcción (gramática):

$$\begin{aligned}\text{Decimal} &\rightarrow 0|1|2|3|4|5|6|7|8|9 \\ \text{Letra} &\rightarrow A|B|C|D|\dots|Z|a|b|c|d|\dots|z \\ \text{Nombre\_var} &\rightarrow \text{Letra} \mid \text{Nombre\_var Decimal} \mid \text{Nombre\_var Letra}\end{aligned}$$

Expresar en lenguaje natural esta regla que define nombres de variables (Nombre\_var).

- P12.29.** Diseñar diagramas sintácticos para describir la sintaxis de las matrículas de coche de la Unión Europea que se forman con 4 números decimales, un espacio en blanco y tres letras mayúsculas.
- P12.30.** Obtener el árbol de análisis de léxico para comprobar si la expresión  $b - a * c + c$  cumple las reglas para expresiones aritméticas definidas en el Problema 12.10.
- P12.31.** Expresar en BNF los diagramas sintácticos de la Figura 12.25.
- P12.32.** Se tiene el siguiente código:

```
for (int i=17; i≤2500; i++)
{
    i=i+1;
    j=r*c-37;
    z = z+12;
}
```

- a) Indicar la mejora que podría introducir el optimizador de un compilador.
  - b) Cuantificar en lo posible los efectos de la mejora.
  - c) Indicar, justificando la respuesta, si un intérprete podría realizar esta optimización.
- P12.33.** Se ha diseñado el hardware para un robot capaz de moverse dentro de un laberinto utilizando un procesador CODE2. El robot es capaz de girar a derecha e izquierda, avanzar, detectar si está frente a la salida y detectar si está frente a la pared. Diseña un lenguaje de programación para el robot que incluya las operaciones de movimiento, las funciones lógicas *and*, *or* y *not* y la estructura de control *while*. Esboce la realización de un intérprete de este lenguaje suponiendo que las señales de los sensores de entrada se dan a través del puerto IP3 y las señales de control para los motores del robot a través del puerto OP3.

## TIPOS DE LENGUAJES DE PROGRAMACIÓN

- P12.34.** Realice una tabla indicando cinco lenguajes de programación de cada uno de los siguientes tipos: imperativos, funcionales, lógicos y orientados a objetos. En una segunda columna de la tabla indique la fecha aproximada de su definición. *Sugerencia:* Consulte por Internet.
- P12.35.** Realice una tabla indicando las fechas de normalización de los siguientes lenguajes de programación: Ada, Basic, C, C++, Cobol, FORTRAN, Java, LISP, Pascal, Prolog y Smalltalk. *Sugerencia:* Consulte en la web [www.ansi.org](http://www.ansi.org).
- P12.36.** Seleccione cuatro lenguajes de alto nivel y realice una tabla comparando los operadores aritméticos de cada uno de ellos.
- P12.37.** Seleccione cuatro lenguajes de alto nivel y realice una tabla comparando los operadores de relación de cada uno de ellos.
- P12.38.** Suponga que en un lenguaje funcional, tal como LISP, se dispone de las funciones que se indican en la Tabla 12.5. Obtener una función que genere el valor máximo de los elementos de una lista.
- P12.39.** Suponga que en un lenguaje funcional, tal como LISP, se dispone de las funciones que se indican en la Tabla 12.5. Obtener una función que proporcione el tercer valor de una lista.
- P12.40.** Para un lenguaje declarativo, tal como Prolog, defina las reglas para expresar las relaciones **primo**, **tío**, **sobrino**, **nieto** y **abuelo**.

# Archivos

La información se almacena en un computador de acuerdo con técnicas que siguen una estrategia y planificación previamente establecidas con el objetivo básico de ocupar poco espacio y acceder fácilmente y con rapidez a datos concretos. Para almacenar y acceder adecuadamente a la información en la memoria principal se utilizan los conceptos de tipos y estructuras de datos vistos en el Capítulo 10. En el presente capítulo efectuaremos una introducción a los archivos, como técnica para organizar los datos en la memoria masiva.

## 13.1. Concepto de archivo

Un **archivo** o **fichero** es un conjunto de información sobre un mismo tema, tratada como un todo y organizada de forma estructurada. El concepto de archivo determina un procedimiento de almacenar datos de forma permanente en dispositivos de memoria externa tales como discos o cintas.

Los programas de aplicación acceden al contenido de los archivos a través del sistema operativo, en particular, a través del **sistema de gestión de archivos** (Sección 9.6). Los archivos, desde el punto de vista del usuario y del programa de aplicación, están formados por **registros lógicos** que contienen información sobre un mismo tema, y que a su vez están divididos en **campos**.

### EJEMPLO 13.1

En la Tabla 13.1 se muestra un ejemplo de un archivo sobre información de aviones. Cada fila corresponde a un registro lógico, y cada apartado dentro de una fila es un campo. Los campos, por tanto, son: matrícula, flota, asientos, autonomía y compañía.

## 13.2. Tipos de archivos

Los archivos pueden ser de muchos tipos, según la función que vayan a desempeñar, y se pueden clasificar atendiendo a varios criterios. Una posible clasificación es la que se da en la Figura 13.1.

- **Archivos permanentes:** Contienen información relevante para una aplicación; es decir, los datos necesarios para su funcionamiento. Su vida es larga. A su vez se clasifican en:
  - **Archivos maestros:** contienen el estado actual de los datos que pueden ser cambiados por una aplicación. Es el núcleo de la aplicación. Ejemplo, un archivo con los datos de los clientes de un banco.



Aviones				
Matrícula	Flota	Asientos	Autonomía (km)	Compañía
EC-BIM	Airbus-340	249	12.700	Iberia
EC-CIM	Airbus-321	186	4.000	Iberia
EC-RAR	Airbus-320/200	150	3.500	Iberia
EC-PIO	Airbus-319	126	2.200	Iberia
EC-CAR	Boeing-747	404	13.000	Iberia
EC-AVE	Boeing-757	200	3.700	Iberia
EC-COJ	MD-88	150	2.900	Iberia
EC-PEP	MD-87	109	2.900	Iberia
EC-BIB	ATR-72	68	1.722	Iberia
EC-PIR	CRJ-200	50	3.045	Iberia
EC-DER	Fokker-50	50	2.252	Iberia
EC-CER	Dash-8-Q300	52	1.511	Iberia
EC-DIR	Boeing-737/300	142	3.700	AirEuropa
EC-LAP	Boeing-737/400	162	3.140	AirEuropa
EC-EPO	Boeing-737/600	116	2.950	AirEuropa
EC-PIO	Boeing-737/800	186	5.000	AirEuropa
EC-CIR	Boeing-767/300	263	9.260	AirEuropa

Tabla 13.1. Archivo de aviones.

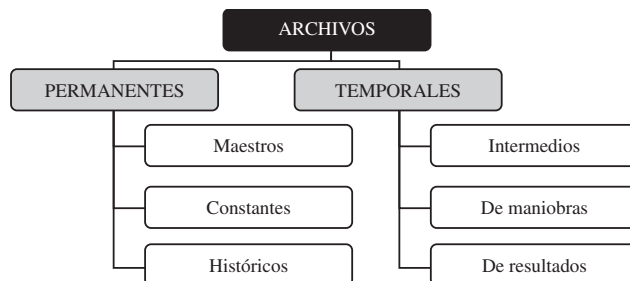


Figura 13.1. Clasificación de los archivos.

- **Archivos constantes:** tienen datos fijos para una aplicación. Las modificaciones son poco frecuentes, se suele acceder para consultas. Ejemplo, una tabla de números primos.
- **Archivos históricos:** contienen datos que fueron actuales anteriormente. Se conservan para reconstruir una situación. Ejemplo, un archivo con la información de los clientes que se han dado de baja en un banco.
- **Archivos temporales:** Almacenan información relevante para un determinado proceso, no para toda la aplicación. Se generan a partir de datos de archivos permanentes. Su vida es corta. Se clasifican en:
  - **Archivos intermedios:** almacenan resultados de un programa que van a ser utilizados por otro programa. Ejemplo, información almacenada en el portapapeles.
  - **Archivos de maniobras:** almacenan datos de un programa que no se pueden conservar en memoria principal por falta de espacio. Ejemplo, algunos programas de cálculo numérico.
  - **Archivos de resultados:** almacenan datos que se van a transferir a un dispositivo de salida. Ejemplo, un archivo de impresión.

### EJEMPLO 13.2

Supongamos una aplicación informática para gestionar la información sobre los libros de una biblioteca. Por un lado se dispondrá de un archivo con el catálogo de libros disponibles. Muy probable-

mente existirá un archivo con las editoriales donde se publican los libros. La información de los libros que no se usan se almacenará en un archivo de libros anticuados. También se dispone de un programa que genera por la impresora un boletín mensual con las nuevas obras adquiridas por la biblioteca el último mes. Los ficheros son del siguiente tipo:

- Catálogo de libros: *archivo maestro*.
- Archivo de editoriales: *archivo constante*.
- Archivo de libros anticuados: *archivo histórico*.
- Archivo generado para producir boletín mensual: *archivo intermedio*.
- Archivo de impresión de boletín mensual: *archivo de resultados*.

Hay distintas formas de organizar un archivo en un soporte de memoria. Las principales organizaciones se resumen en la Figura 13.2.

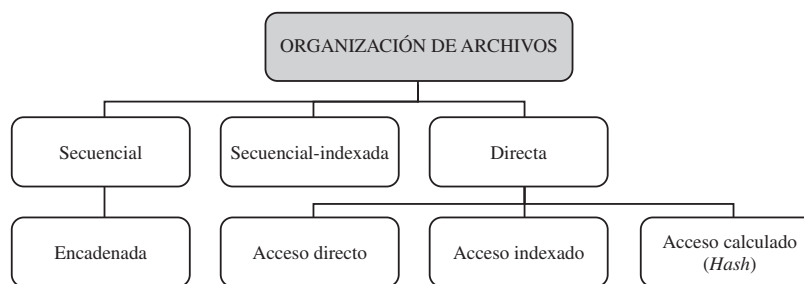


Figura 13.2. Organización de archivos.

En las próximas secciones analizaremos este tipo de organizaciones.

### 13.3. Organización secuencial

Un archivo con **organización secuencial** es aquel cuyos registros están almacenados de forma contigua, de manera que la única forma de acceder a él (para copiarlo en memoria principal, por ejemplo) es leyendo un registro tras otro desde el principio hasta el final. En los archivos secuenciales suele haber una marca que indica el fin del archivo, que se suele denominar **EOF** (*End Of File*); para detectar la terminación del archivo sólo hay que buscar esa marca. Es la única organización que puede utilizar un dispositivo no direccionable, cintas magnéticas, por ejemplo. Esta organización también se utiliza en los CD de audio y los DVD de vídeo, en los que la música o las imágenes se almacenan a lo largo de una espiral continua.

Los registros se suelen identificar por medio de una información ubicada en uno de sus campos. A este campo se le suele llamar **clave** o **llave**. Si se ordena un archivo secuencial según una clave, es más rápido hacer cualquier operación.

#### EJEMPLO 13.3

En el archivo de la Tabla 13.2 se utiliza como clave el campo “matrícula”: se accede a la información de un avión (un registro) concreto proporcionando la matrícula del mismo.

Operaciones que se pueden realizar:

- **Añadir:** sólo se pueden añadir registros al final del archivo.
- **Consultar o recuperar:** se hace en forma secuencial. Para leer el registro  $n$  hay que leer los  $n - 1$  anteriores.

Aviones				
Matrícula	Flota	Asientos	Autonomía (km)	Compañía
EC-AVE	Boeing-757	200	3.700	Iberia
EC-BIB	ATR-72	68	1.722	Iberia
EC-BIM	Airbus-340	249	12.700	Iberia
EC-CAR	Boeing-747	404	13.000	Iberia
EC-CER	Dash-8-Q300	52	1.511	Iberia
EC-CIM	Airbus-321	186	4.000	Iberia
EC-CIR	Boeing-767/300	263	9.260	AirEuropa
EC-COJ	MD-88	150	2.900	Iberia
EC-DER	Fokker-50	50	2.252	Iberia
EC-DIR	Boeing-737/300	142	3.700	AirEuropa
EC-EPO	Boeing-737/600	116	2.950	AirEuropa
EC-LAP	Boeing-737/400	162	3.140	AirEuropa
EC-PEP	MD-87	109	2.900	Iberia
EC-PIO	Airbus-319	126	2.200	Iberia
EC-PIO	Boeing-737/800	186	5.000	AirEuropa
EC-PIR	CRJ-200	50	3.045	Iberia
EC-RAR	Airbus-320/200	150	3.500	Iberia

**Tabla 13.2.** Archivo de aviones ordenado según la clave.

- **Insertar, modificar o eliminar:** para llevar a cabo cualquiera de estas operaciones intervienen cuatro archivos:

- *Archivo maestro:* Es el archivo original que hay que actualizar.
- *Archivo maestro nuevo:* Archivo con los datos actualizados.
- *Archivo de movimientos:* Archivo que contiene los cambios que hay que aplicar al archivo maestro. Este archivo tiene un código para cada operación que hay que hacer en cada registro (insertar, modificar o eliminar).
- *Archivo de errores:* Contiene una lista de los errores producidos en el proceso de actualización.

Todos los archivos se ordenan según una misma clave y si no hay errores se va aplicando cada cambio del archivo de movimientos al archivo maestro para ir generando el archivo maestro nuevo. Cuando se elimina un registro se suele realizar un **borrado lógico**, es decir, se marca escribiendo un valor determinado en un campo especial que indica que ese registro se ha borrado, aunque físicamente siga existiendo.

#### EJEMPLO 13.4

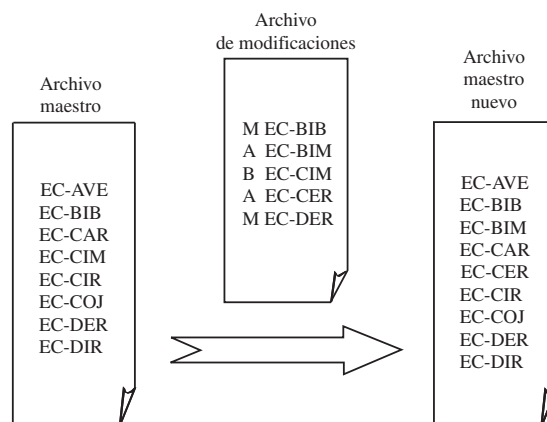
En la Figura 13.3 se muestra cómo se actualizaría el archivo de aviones, con nuevos. Para simplificar la figura sólo se muestran las matrículas de los aviones.

Este tipo de organización es adecuada para archivos en los que no haya procesos interactivos, y en los que haya que acceder a la mayor parte de los registros. Sus ventajas son su sencillez y no ocupar mucho espacio.

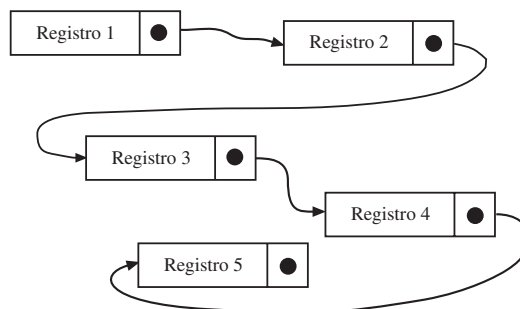
En un archivo con **organización secuencial encadenada** junto a cada registro se almacena un puntero con la dirección del registro siguiente (Figura 13.4). Estos archivos sólo se pueden almacenar en soportes direccionables.

Operaciones que se pueden realizar:

- **Consultar:** Las consultas son secuenciales.
- **Insertar:** Para insertar un registro primero hay que localizar la posición en la que se va a insertar, luego se escribe el registro en una zona libre y por último se cambia el puntero del re-



**Figura 13.3.** Proceso de actualización de un archivo con organización secuencial (A, B y M, indican altas, bajas y modificaciones, respectivamente).



**Figura 13.4.** Organización secuencial encadenada.

registro anterior para que apunte al nuevo registro, y se actualiza el valor del puntero del nuevo registro para que apunte al siguiente.

- **Modificar:** Si la modificación no implica cambio de longitud se reescribe. Si hay cambio de longitud se inserta el registro y se borra el antiguo.
- **Eliminar:** Se asigna al puntero del registro anterior la dirección del registro siguiente del modificado. El espacio del registro borrado puede ser ocupado por otro registro.

Esta organización es útil cuando se van a hacer numerosas inserciones. Es muy flexible pero tiene la limitación de que las consultas son secuenciales.

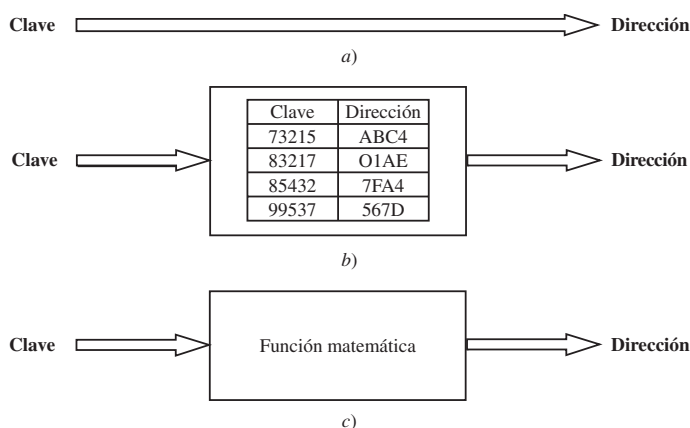
## 13.4. Organización directa

En la organización secuencial, si se desea acceder a un determinado registro, es necesario leer los que le preceden. En una **organización directa o aleatoria** se puede acceder a un registro indicando la posición relativa del mismo dentro del archivo o, más comúnmente, a través de una **clave** (número de identificación fiscal —NIF, en España—, número de la seguridad social, matrícula de coche, etc.) que forma parte del registro como un campo más. Estos archivos deben almacenarse en dispositivos de memoria masiva de acceso directo, tales como discos magnéticos.

Cada registro se guarda en una posición física, que viene determinada por las disponibilidades de espacio en la memoria masiva; por ello, en cierta medida los registros se distribuyen de forma *alea-*

toría dentro del soporte de almacenamiento. A cada posición física se accede especificando su **dirección** o **índice**.

Según se ha indicado anteriormente, en estos archivos el acceso a los registros se hace por su clave, por lo que dada una clave es necesario obtener la dirección del registro correspondiente. En otras palabras, debe existir una transformación perfectamente definida entre clave y dirección. Según la forma de obtener la dirección del registro a partir de su clave se dispone de distintos métodos de acceso (Figura 13.5). En el **acceso directo** la clave coincide con la dirección (Figura 13.5a), teniendo que ser numérica y su rango de valores tiene que ser menor o igual que el rango de direcciones, siendo el método de acceso más rápido. Los otros métodos de acceso directo se describen en las secciones siguientes.



**Figura 13.5.** Modos de acceso a archivos directos: a) directo, b) indexado y c) calculado.

### 13.4.1. ACCESO INDEXADO

En el **acceso indexado** existe una **zona de registros** en la que se encuentran los datos del archivo. Adicionalmente hay una **zona de índices** que contiene una tabla con las claves de los registros y las posiciones donde se encuentran los mismos (Figura 13.5b). Usualmente la tabla de índices se encuentra ordenada según la clave.

Una forma típica de trabajar con un **archivo indexado** es la siguiente. En el dispositivo de memoria masiva se encuentran almacenados los índices y los registros de datos en dos archivos distintos. Al abrir el archivo, el sistema operativo carga en la memoria principal el archivo de índices (la zona de registros suele ser de una gran capacidad y no cabe en la memoria principal). Para acceder a un registro concreto se busca en la tabla de índices la fila correspondiente a la clave del registro, obteniéndose así la dirección donde se encuentra el registro. Esta búsqueda dentro de la tabla de índices es muy rápida por dos motivos: a) la tabla está en memoria principal y b) se puede realizar una **búsqueda binaria** (Problema 11.12) al estar la tabla ordenada según las claves. Una vez establecida la dirección del registro se obtiene éste rápidamente del disco utilizando acceso directo.

#### EJEMPLO 13.5

En la Figura 13.6 se muestra un archivo indexado de los 23.742 alumnos de una universidad. Obviamente todo el archivo (registros) no cabe en memoria. Al abrir el archivo, el sistema operativo carga en la memoria principal la zona de índices. Como clave se utiliza el número de identificación del

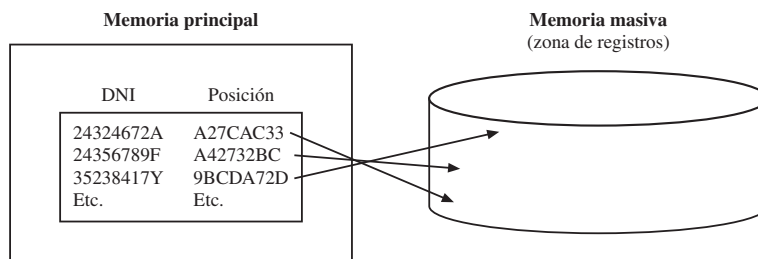


Figura 13.6. Organización indexada.

alumno (DNI)<sup>1</sup>. Para acceder a un alumno hay que proporcionar su DNI, a continuación, por medio de una búsqueda dicotómica se busca el DNI en la tabla de índices, obteniéndose así la posición del registro de ese alumno en la memoria secundaria. Una vez conocida la dirección, por acceso directo se obtiene la información del registro: nombre y apellidos, domicilio, edad, curso y asignaturas en que se encuentra matriculado, calificaciones de las asignaturas cursadas previamente, etc. Si, por ejemplo, se quiere acceder al alumno de DNI 35238417F, la tabla de índices nos proporcionará la dirección de disco donde se encuentra el registro del alumno: 9BCDA72D.

Las operaciones que se pueden realizar en archivos indexados son:

- **Consultar o leer:** Se busca en la tabla de índices la clave, si es posible con una búsqueda dicotómica, obteniendo así la dirección física del inicio del registro. Conocida la dirección, se lee por acceso directo del dispositivo de memoria masiva el registro buscado.
- **Insertar:** Al introducir un registro, primero debe comprobarse si está ya en la memoria masiva; caso de que no sea así, debe ubicarse en algún espacio que esté libre. Una vez almacenado se actualiza la tabla de índices con la clave y dirección del nuevo registro. Una vez hecho esto debe reordenarse la tabla de índices.
- **Modificar:** Se lee del dispositivo de memoria masiva el registro (como una consulta), se actualiza, y se vuelve a grabar en la memoria masiva.
- **Eliminar:** Se hace un borrado lógico, consistente en suprimir de la tabla de índices la información del registro en cuestión, y liberando el espacio de memoria masiva que ocupaba el registro.

### 13.4.2. ACCESO PARCIALMENTE INDEXADO

Otra técnica muy usada de almacenamiento de archivos es el **acceso parcialmente indexado**, también denominado **secuencial indexado**. Como en la organización indexada, existe una zona de índices y otra de registros de datos, pero esta última se encuentra dividida en **segmentos** (bloques de registros) ordenados. En la tabla de índices, cada fila hace referencia a cada uno de los segmentos: la clave corresponde al último registro y el índice apunta al registro inicial. Una vez que se accede al primer registro del segmento, dentro de él se localiza (de forma secuencial, por ejemplo) el registro buscado.

#### EJEMPLO 13.6

En la Figura 13.7 se muestra un esquema de un archivo secuencial indexado. Si, por ejemplo, se deseara acceder al registro de clave *E*, primero se busca en la tabla de índices la dirección de inicio del segmento de registros. En este caso, como  $C < E < F$  el bloque donde se encuentra *E* será el asocia-

<sup>1</sup> El número de DNI en España es un código formado con 8 cifras decimales, eventualmente seguido de una letra de detección de errores.

Zona de índices		Zona de registros		
Clave	Dirección de segmento	Dirección	Registro	
C	A371	A371	A	otros campos
F	CD38	A372	B	otros campos
I	DE48	A373	C	otros campos
J	F73D			
Etc.	Etc.	CD38	D	otros campos
		CD39	E	otros campos
		CD3A	F	otros campos
		DE48	G	otros campos
		DE49	H	otros campos
		DEA	I	otros campos
		F73D	J	otros campos

**Figura 13.7.** Organización parcialmente indexada.

do a la clave *F*; es decir, según indica la tabla de índices, dicho bloque empieza en la dirección CD38. Una vez que se ha accedido al inicio del bloque, dentro de él se busca secuencialmente el registro  $D \rightarrow E$ . Fácilmente se deduce que es mucho más rápido buscar secuencialmente dentro de un segmento que dentro del archivo completo, por lo que los archivos secuenciales indexados son más eficientes que los secuenciales puros en cuanto al tiempo de acceso.

### 13.4.3. ACCESO CALCULADO (HASH)

La organización indexada presenta el inconveniente de tener que consultar una tabla para obtener la dirección de almacenamiento a partir de la clave. Existe otra técnica que permite accesos más rápidos; en efecto, con el **acceso calculado** o *hash*, en lugar de consultar una tabla, se utiliza una transformación matemática conocida que a partir de la clave genera la dirección de cada registro del archivo (Figura 13.5c). Si la clave es alfanumérica, previamente se transforma en un número. Existen diversos métodos para calcular la dirección; por ejemplo:

- *Módulo*: la dirección se hace igual al resto de la división entera entre la clave y el número de registros<sup>2</sup>.
- *Extracción*: la dirección se forma extrayendo ciertas cifras de la clave.
- *Elevación al cuadrado*: la dirección se eleva al cuadrado y se toman como dirección los dígitos centrales.

Con el acceso calculado puede ocurrir que a partir de distintas claves se obtenga la misma dirección, es decir, que a dos o más registros se les asigne la misma dirección. A este problema se le lla-

<sup>2</sup> Desde un punto de vista práctico, se demuestra que se obtienen menos sinónimos si la función módulo se hace con el número primo inmediatamente superior al número máximo de registros (ver, por ejemplo, la referencia bibliográfica [1]).

ma **colisión**, y las claves que generan la misma dirección se denominan **sinónimos**. Existen diversos métodos para resolver este problema, como dejar al final del archivo un bloque de excedentes, denominada **zona de sinónimos**, o incluso crear un **archivo de sinónimos** asociado al archivo principal: si no se encuentra el registro en su sitio, se busca en la zona o en el archivo de sinónimos. Obviamente el tiempo medio de acceso a los registros crece notablemente si el número de sinónimos y su probabilidad de acceso son grandes.

Con frecuencia la dirección generada con el acceso calculado corresponde, en lugar de la ubicación exacta del registro buscado, a un segmento donde se encuentra éste. Cada **segmento** contiene los sinónimos correspondientes a una dirección. De la misma forma que en el direccionamiento parcialmente indexado, una vez que se accede al segmento se busca dentro de él el registro solicitado. Los registros de cada segmento se pueden organizar, por ejemplo, como un archivo secuencial encadenado, de esta forma el tamaño de los segmentos puede ser variable. Para más detalles puede verse el Problema 13.7.

### EJEMPLO 13.7

Suponga que el número de empleados máximo de una empresa es de 4.000, reservando en un disco 4.555 registros para almacenarlos. Supongamos que se utiliza como clave el número del documento nacional de identidad (DNI), que, supongamos tiene valores entre 0 y 50.000.000. Caso de que el acceso se hiciese con una transformación módulo:

- Si la clave fuese 77.429, la dirección correspondiente sería:

$$\text{dirección} = \text{módulo}(77.429; 4.555) + 1 = 4.549 + 1 = 4.950$$

- Si la clave fuese 2.324, la dirección correspondiente sería:

$$\text{dirección} = \text{módulo}(2.324; 4.555) + 1 = 0 + 1 = 1$$

- Si la clave fuese 4.015, la dirección correspondiente sería:

$$\text{dirección} = \text{módulo}(4.015; 4.555) + 1 = 0 + 1 = 1$$

es decir, las claves 4.015 y 2.324 son sinónimas por generar la misma dirección.

Obsérvese que si las 4.000 claves (DNI) de los empleados no se encontrasen distribuidas uniformemente en el rango de posibles valores (de 0 a 50.000.000) se producirían numerosos sinónimos.

Entre las operaciones básicas, además de las de abrir y cerrar archivo, que se pueden hacer con un archivo con direccionamiento calculado se encuentran:

- **Consultar:** Para realizar una consulta se obtiene la dirección asociada por medio de la transformación matemática, se lee el registro en esa dirección y se comprueba si las claves coinciden; si es así, ese será el registro buscado, sino se buscará en la zona de sinónimos.
- **Insertar:** Para introducir un nuevo registro en el archivo, se aplica la transformación a la clave; si la posición de memoria masiva está libre, se escribe allí el registro, y si, por el contrario, está ocupado por otro registro con distinta clave, se trata de un sinónimo que debe guardarse en la zona o archivo de sinónimos.
- **Modificar:** Para modificar un registro previamente almacenado en el archivo, primero se lee mediante una consulta y se realiza una operación de inserción, sobrescribiéndose en la posición que previamente ocupaba.
- **Eliminar:** Se realiza un borrado lógico; es decir, se hace una operación de modificación, poniendo una marca predeterminada en uno de sus campos.



## 13.5. Parámetros de utilización de un archivo

El tipo de organización que tenga un archivo depende del uso que se vaya a hacer de él. Existen una serie de parámetros que se utilizan para determinar el uso de un archivo: volumen, actividad, volatilidad y crecimiento.

- **Capacidad o volumen:** es el espacio, en caracteres, que ocupa el archivo.

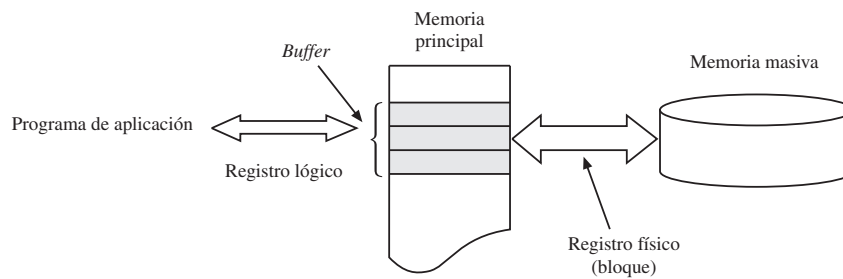
$$\text{Capacidad} = n.^{\circ} \text{previsto de registros} \times \text{longitud media de registro}$$

- **Actividad:** representa la cantidad de las consultas y modificaciones que se hacen en un archivo y se especifica con:
  - **Tasa de consulta o modificación:** porcentaje de registros consultados o modificados en cada tratamiento del archivo, respecto al número total de registros del archivo.
  - **Frecuencia de consulta o modificación:** número de veces que se accede al archivo para hacer una consulta o modificación en un período de tiempo fijo.
- **Volatilidad:** es una medida de la cantidad de inserciones y borrados que se efectúan en un archivo. Se especifica con:
  - **Tasa de renovación:** porcentaje de registros renovados en cada tratamiento del archivo, respecto al número total de registros del archivo.
  - **Frecuencia de renovación:** número de veces que se accede al archivo para renovarlo en un período de tiempo fijo.
- **Crecimiento:** es la variación de la capacidad del archivo y se mide con la **tasa de crecimiento**, que es el porcentaje de registros en que aumenta el archivo en cada tratamiento.

## 13.6. Ejemplo: gestión de archivos en Windows y UNIX

Según indicamos en la Sección 13.1, los archivos se organizan en registros lógicos, formados por campos. Los registros lógicos se definen en función de la aplicación o aplicaciones que los vayan a utilizar, no estando, por tanto, su tamaño definido a priori.

Por otra parte, los archivos se almacenan en los dispositivos de memoria masiva en forma de unidades, denominadas **registros físicos o bloques** (Secciones 5.3.2 y 9.6). Un registro físico es la cantidad mínima de información que puede transferirse en una operación de entrada/salida, y su tamaño viene determinado por el dispositivo físico y el sistema operativo. Así, en las últimas versiones de Windows de Microsoft utilizan tamaños de 2 o 4 KBytes para los bloques (que denominan **unidades de ubicación o clusters**). En consecuencia, no existe a priori ninguna relación entre el tamaño de un registro lógico y el de un bloque: un registro lógico por lo general ocupa parte de un bloque, aunque puede ocupar varios de ellos. Se denomina **factor de blocaje** al número de registros lógicos que se pueden almacenar en un registro físico. El sistema operativo se encarga de gestionar el almacenamiento de los registros lógicos. Cuando un programa quiere acceder al contenido de un archivo genera una llamada al sistema operativo y éste lee los registros físicos necesarios, los almacena en una zona de la memoria principal asociada a la entradas/salidas del archivo denominada **buffer** y el programa de aplicación accede a ese **buffer** para leer el registro lógico o datos que haya solicitado. De forma similar, cuando una aplicación quiere almacenar información en un archivo, pasa los datos al sistema operativo, que los almacena en el **buffer** y cuando se completa un registro físico lo transfiere al dispositivo de memoria masiva (Figura 13.8).

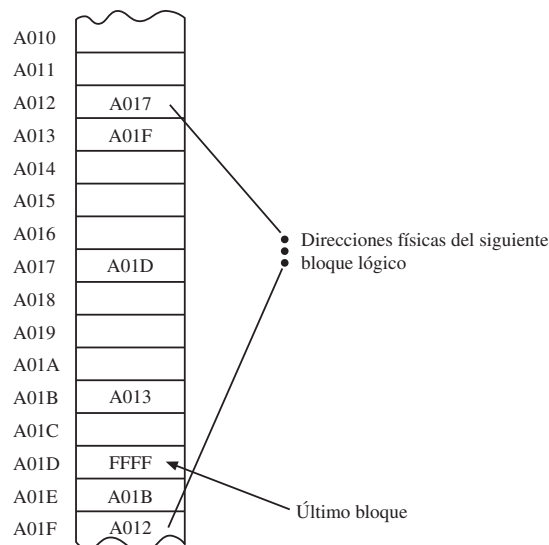


**Figura 13.8.** Gestión del sistema operativo en la gestión de archivos.

Para poder llevar a cabo estas operaciones el sistema operativo tiene que conocer cierta información sobre el archivo, como su nombre, en qué dispositivo está almacenado, el *buffer* utilizado, etc. Esta información se guarda en una tabla en memoria principal llamada **descriptor de archivos**. Esta tabla se crea cuando una aplicación le comunica al sistema operativo que quiere acceder a un archivo (**abrir el archivo**), y se elimina cuando la aplicación le dice al sistema operativo que ya no va a necesitar más ese archivo (**cerrar el archivo**).

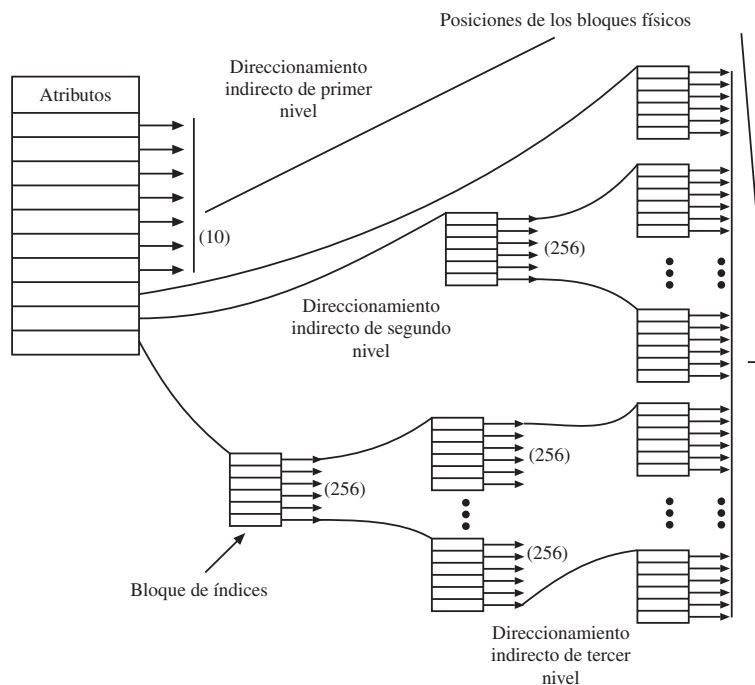
Cuando damos al sistema operativo la orden de almacenar un archivo de un programa, de un texto, vídeo o música, por ejemplo, utiliza una de las siguientes formas:

- **Contigua:** los bloques se almacenan consecutivamente uno tras otro. Esta forma es la usual en los CD, que suelen emplear un acceso de tipo secuencial indexado. En efecto en una tabla se ubican los nombres y direcciones de inicio de determinados bloques (una canción, un pasaje determinado de una película, etc.), y dentro de cada bloque la información se lee de forma secuencial.
- **Lista encadenada:** los bloques se almacenan en **unidades de ubicación** (“clusters”) no necesariamente consecutivos, encadenándose un bloque con otro por medio de un puntero. Se accede al archivo conociendo la dirección de la unidad de ubicación inicial.
- **Lista de enlaces:** cada disco dispone de una **tabla de ubicación de archivos** con tantos elementos como bloques físicos, la posición de cada elemento se corresponde biunívocamente con cada bloque, y contiene el puntero al lugar donde se encuentra el siguiente bloque del archivo (Figura 13.9).



**Figura 13.9.** Grabación y localización de las unidades de ubicación con una lista de enlaces. El archivo comenzaría en el bloque A01E, y continuaría en: A01B, A013, A01F, A012, A017 y A01D.

- **Archivo de índices o i-nodos:** corresponde a la forma de gestionar los archivos por el sistema operativo UNIX. Los parámetros que se dan a continuación corresponden a UNIX versión 7. Cada archivo tiene asociado un **nudo de información o i-nodo**, que es una pequeña tabla de tamaño fijo (64 Bytes) conteniendo los atributos del archivo y 13 direcciones de 3 bytes. Las 10 primeras direcciones indican directamente las posiciones de las 10 primeras unidades de ubicación donde se encuentran los datos del archivo, mientras que las 3 últimas indican las siguientes direcciones de los bloques del archivo, pero de forma indirecta a través de bloques de índices: con simple indirección, doble indirección y triple indirección, respectivamente, según se indica en la Figura 13.10. Cada bloque de índices puede tener, por ejemplo, 256 direcciones a otros bloques de índices o a unidades de ubicación de datos del archivo. En UNIX Sistema V el tamaño de las unidades de ubicación es de 1 KByte, con lo que, con el esquema anterior, la capacidad de un archivo puede llegar a 16 GBytes.



**Figura 13.10.** Grabación y localización de las unidades de ubicación con un archivo de índices.

### EJEMPLO 13.8

El sistema operativo **MS-DOS**, **Windows** utilizan para grabación de la información una lista de enlaces denominada **FAT** (tabla de localización de archivos, Figura 13.9). Existen dos versiones FAT16 y FAT32:

- La versión **FAT16** sirve para discos de hasta 2 GBytes (unidades de ubicación de 32 KB), cada dirección de bloque se da con 2 Bytes. Como cada elemento de la FAT corresponde a un bloque, se puede grabar en él información alternativa al puntero; por ejemplo, indicar si el bloque está deteriorado (H'FFF7, en MS-DOS), si está libre (H'0000) y por tanto disponible para su uso, o si es el último del archivo (H'FFFF).
- En la versión **FAT32** (últimas versiones de Windows 95 y Windows 98), sirve para discos de hasta 1TB/2TB, siendo, sin embargo, las unidades de ubicación menores (2 KB/4 KB), lo que implica un mejor aprovechamiento del disco. Cada elemento de la FAT tiene 32 bits, de los que 29 especifican el número de la unidad de asignación.

Los sistemas operativos actuales de Microsoft (tales como **Windows NT** y **Windows XP**) utilizan el denominado **sistema de archivos de nueva tecnología (NTFS, “NT File System”)**, que admite varias formas de archivos, como FAT16, FAT32 y OS/2. Las unidades de almacenamiento que se consideran en NTFS son:

- *Sector*: unidad de almacenamiento más pequeña (usualmente de 250 Bytes).
- *Unidad de ubicación*: uno o más sectores contiguos, en la misma pista (de 512 Bytes a 64 KBytes).
- *Volumen*: porción lógica de un disco compuesta de una o más unidades de ubicación (de 1 a 128 sectores/volumen).

Con NTFS se pueden tener archivos de hasta  $2^{64}$  Bytes. Las unidades de ubicación, dentro de un determinado volumen, se especifican con 64 bits. Cada volumen contiene:

- *Región de arranque de la partición*: con información para arranque del volumen y sobre el propio volumen.
- *Tabla maestra de archivos* (MFT, “Master File Table”): con información acerca de los archivos y carpetas dentro del volumen e incluso los propios datos del archivo.
- *Archivos del sistema*: entre los que se encuentran archivos redundantes (“espejos”) para recuperación de fallos, lista de transacciones realizadas, tabla de definición de tipos de atributos admitidos por el volumen, mapa que especifica las unidades de ubicación que están ocupadas y libres.
- *Zona de expansión* para datos de archivos.

El elemento fundamental del volumen es la tabla maestra de archivos, que suele tener un elemento (registro) de longitud variable por archivo o carpeta del volumen. Contiene, entre otras, la siguiente información:

- *Información estándar*, que consiste en los atributos del sistema.
- *Nombre(s) del archivo*, que puede ser de hasta 255 caracteres Unicode. Admite varias denominaciones simultáneas para un mismo archivo, en particular las MS-DOS.
- *Seguridad*, puntero a la zona referente al archivo, que se encuentra en un archivo centralizado que contiene toda la información de seguridad.
- *Datos*, si el archivo es pequeño, su contenido se encuentra en el propio registro de la tabla (**archivos inmediatos**), si no contiene punteros a las unidades de ubicación de la zona de expansión donde se encuentran los datos sucesivos, o, usualmente, las direcciones iniciales de las unidades de ubicación donde realmente se encuentran los datos, junto con la longitud de los datos del archivo.

## 13.7. Conclusión

En este capítulo se ha analizado cómo se almacena la información en soportes de memoria masiva. La información referente a un mismo tema se organiza en un **archivo**, y éste se estructura en **registros lógicos**, cada uno de los cuales se refiere a un elemento o individuo distinto. Las técnicas de organización de archivos tratan de cubrir dos objetivos básicos:

1. Acceder con rapidez a los registros, bien sea para consultarlos, modificarlos o darlos de baja.
2. Conseguir que el archivo ocupe el menor espacio posible.

Después de analizar el concepto de archivo (Sección 13.1), hemos descrito los distintos tipos de archivos desde el punto de vista de la función que van a desempeñar (Sección 13.2). Posteriormente hemos estudiado las dos principales organizaciones de archivos: secuencial (Sección 13.3) y directa (Sección 13.4); esta última con las variantes de acceso directo, indexado y calculado. La Sección 13.5 la hemos dedicado a definir parámetros que sirven para medir el volumen, actividad, volatilidad y crecimiento de un archivo.

El capítulo ha concluido con la presentación de algunos conceptos sobre cómo los sistemas operativos ayudan a la gestión de archivos, trasladando la visión lógica o virtual del programador a las peculiaridades físicas de los soportes de memoria masiva, fundamentalmente discos magnéticos.



## Test

**T13.1.** En Informática, un conjunto de información sobre un mismo tema tratado como una unidad de almacenamiento y organizada de forma estructurada constituye:

- a) Una base de datos.
- b) Un sistema de gestión de archivos.
- c) Un bloque físico de información.
- d) Un archivo.

**T13.2.** Un *archivo*:

- a) Es externo al programa que lo utiliza (se almacena en memoria masiva independientemente del programa que lo utiliza, sin conexión con el programa).
- b) Forma parte del programa que lo usa.
- c) Es independiente del programa que lo usa, pero se almacena en memoria masiva ligado a éste.
- d) Se carga en la memoria principal conjuntamente con el programa que lo usa, formando un todo.

**T13.3.** Un *registro* de un archivo es:

- a) El archivo, cuando está en memoria principal.
- b) Un dato individual o ítem del mismo (el DNI de un alumno o la matrícula de un coche).
- c) La información almacenada en memoria principal o disco correspondiente a una línea de pantalla.
- d) Una unidad o elemento constitutivo del archivo (información sobre un alumno, por ejemplo).

**T13.4.** Un *campo* de un archivo es:

- a) Una unidad o elemento constitutivo de un archivo (información sobre un alumno, por ejemplo).
- b) El propio archivo, cuando está en memoria principal.
- c) Un dato individual o ítem de un archivo (el DNI de un alumno o la matrícula de un coche).
- d) La información almacenada en memoria principal o disco correspondiente a una línea de pantalla.

**T13.5.** Una *clave*, en un archivo, es:

- a) Un campo (o varios) que identifica a cada registro del archivo.

- b) La dirección en disco donde comienza el archivo.
- c) Una identificación (*password*) para permitir el acceso al archivo.
- d) Un carácter especial que delimita o separa registros.

**T13.6.** Un *archivo maestro*:

- a) Contiene información que no cambia nunca, y por tanto tiene una vida larga.
- b) Contiene información relevante para una aplicación, que cambia a menudo pero tiene una vida larga.
- c) Contiene información relevante para un determinado proceso, pero no para toda la aplicación; tiene una vida larga.
- d) Tiene una vida larga, aunque sus datos no sean necesarios para el funcionamiento de una aplicación (almacena datos históricos).

**T13.7.** Un *archivo temporal* es:

- a) Aquel cuya información cambia a menudo, pero su vida es larga.
- b) Aquel cuya información no cambia nunca, pero su vida es corta.
- c) Aquel que contiene información relevante para un determinado proceso, pero no para toda la aplicación. Su vida es corta.
- d) Aquel que contiene información relevante para una aplicación y su vida es corta.

**T13.8.** Dentro de los archivos permanentes se incluyen sólo los archivos:

- a) Maestros y constantes.
- b) Maestros e históricos.
- c) Constantes e históricos.
- d) Maestros, constantes e históricos.

**T13.9.** Dentro de los archivos temporales se incluyen sólo los archivos:

- a) Intermedios, de maniobras y de resultados.
- b) Intermedios y de maniobras.
- c) Maniobras y de resultados.
- d) Intermedios y de resultados.

**T13.10.** Un archivo histórico contiene datos:

- a) Referentes a hechos históricos (ejemplo: archivo de una enciclopedia).
- b) Que fueron actuales anteriormente (ejemplo: clientes dados de baja de un banco).
- c) Que llevan mucho tiempo dentro del ordenador (ejemplo: clientes muy antiguos de un banco).
- d) Generados por el sistema operativo sobre los antecedentes del ordenador (ejemplo: averías, ampliaciones, diagnósticos, etc.).

**T13.11.** Los archivos en los que los registros se encuentran yuxtapuestos consecutivamente son los de organización de tipo:

- a) Secuencial.
- b) Indexada.
- c) Encadenada.
- d) Directa.

**T13.12.** Los archivos en los que se consulta una tabla de índices para localizar un registro determinado son los de organización de tipo:

- a) Secuencial.
- b) Indexada.
- c) Encadenada.
- d) Directa.

**T13.13.** Los archivos en los que cada registro contiene un puntero que indica donde se encuentra el siguiente son los de organización de tipo:

- a) Secuencial.
- b) Indexada.
- c) Encadenada.
- d) Directa.

**T13.14.** La organización de archivos que necesita crear un archivo maestro nuevo cuando se hace una actualización es la:

- a) Secuencial.
- b) Indexada.
- c) Encadenada.
- d) Directa.

**T13.15.** La organización de archivos que necesita localizar el registro anterior (rA) a la posición lógica donde se desea insertar el nuevo registro (rN) y modificar rA es la:

- a) Secuencial.
- b) Indexada.
- c) Encadenada.
- d) Directa.

**T13.16.** La organización de archivos que para localizar un registro necesita disponer de una tabla con las claves ordenadas y las direcciones de los registros correspondientes es la:

- a) Secuencial.
- b) Indexada.
- c) Encadenada.
- d) Directa.

**T13.17.** El tipo de organización de archivos que a través de una transformación conocida (cálculo, por ejemplo) obtiene la dirección para localizar un registro es:

- a) Secuencial.
- b) Secuencial indexada.
- c) Encadenada.
- d) Directa.

**T13.18.** En una organización secuencial de archivos:

- a) Hay una zona de registros, que contiene los registros ordenados según una clave, y una zona de índices, que contiene la clave y la dirección de cada registro.
- b) Los registros se almacenan de forma contigua siguiendo la secuencia lógica del archivo.
- c) Junto a cada registro hay un puntero que contiene la dirección del registro siguiente.
- d) Existe una transformación conocida que genera la dirección de cada registro a partir de una clave numérica conocida.

**T13.19.** En una organización indexada de archivos:

- a) Junto a cada registro hay un puntero que contiene la dirección del registro siguiente.
- b) Los registros se almacenan de forma contigua siguiendo la secuencia lógica del archivo.
- c) Hay una zona de registros, que contiene los registros ordenados según una clave, y una zona de índices, que contiene la clave y la dirección de cada registro.
- d) Existe una transformación conocida que genera la dirección de cada registro a partir de una clave numérica conocida.

**T13.20.** La única organización de archivos susceptible de ser gestionada directamente en un dispositivo de acceso secuencial (cinta magnética) es la:

- a) Secuencial.
- b) Indexada.
- c) Encadenada.
- d) Directa.

**T13.21.** En una organización encadenada de archivos:

- a) Hay una zona de registros, que contiene los registros ordenados según una clave, y una zona de índices, que contiene la clave y la dirección de cada registro.
- b) Los registros se almacenan de forma contigua siguiendo la secuencia lógica del archivo.
- c) Existe una transformación conocida que genera la dirección de cada registro a partir de una clave numérica conocida.
- d) Junto a cada registro hay un puntero que contiene la dirección del registro siguiente.

**T13.22.** En una organización directa de archivos:

- a) La dirección de cada registro se obtiene a partir de su clave.
- b) Hay una zona de registros, que contiene los registros ordenados según una clave, y una zona de índices, que contiene la clave y la dirección de cada registro.

- c) Los registros se almacenan de forma contigua siguiendo la secuencia lógica del archivo.
- d) Junto a cada registro hay un puntero que contiene la dirección del registro siguiente.

**T13.23.** Cuando se actualiza (se inserta, modifica o elimina algún registro/s) un archivo con organización secuencial hay que actualizar:

- a) El archivo maestro.
- b) El archivo maestro nuevo.
- c) El archivo de modificaciones.
- d) El archivo de errores.

**T13.24.** Cuando se actualiza (se inserta, modifica o elimina algún registro/s) un archivo con organización secuencial los cambios se almacenan en:

- a) El archivo maestro.
- b) El archivo maestro nuevo.
- c) El archivo de modificaciones.
- d) El archivo de errores.

**T13.25.** En un archivo con organización directa con acceso directo:

- a) Se utiliza la dirección como clave.
- b) Hay una tabla en la que está cada clave con la dirección del registro correspondiente.
- c) La dirección es el resto de dividir la clave entre el tamaño del archivo más 1.
- d) La dirección se extrae de la clave.

**T13.26.** En un archivo con organización aleatoria con direccionamiento calculado (*hash*):

- a) Se utiliza la dirección como clave.
- b) Hay una tabla en la que está cada clave con la dirección del registro correspondiente.
- c) La dirección se obtiene a partir de la clave mediante un algoritmo.
- d) Las claves se asignan aleatoriamente.

**T13.27.** El volumen de un archivo es:

- a) El número de registros que contiene.
- b) El espacio en caracteres que ocupa en el soporte donde está almacenado.
- c) El espacio que ocupa en disco o cinta, teniendo en cuenta los caracteres de control y de detección de errores.
- d) El número total de campos (datos individuales) que contiene el archivo.

**T13.28.** La actividad de un archivo es un parámetro que:

- a) Indica la cantidad de veces que se abre o cierra el archivo.
- b) Indica si el archivo es utilizado por muchos o pocos usuarios.

- c) Caracteriza el volumen de las consultas y modificaciones del archivo.
- d) Indica el peso de los procesos de inserción y borrado del archivo.

**T13.29.** La volatilidad de un archivo es un parámetro que indica:

- a) La variación del volumen del archivo.
- b) La cantidad de veces que se abre o cierra el archivo.
- c) Si se utiliza mucho o poco.
- d) El peso de los procesos de inserción y borrado en dicho archivo (frecuencia de renovación).

**T13.30.** El crecimiento de un archivo es un parámetro que:

- a) Caracteriza la variación del volumen del archivo.
- b) Indica el peso de los procesos de inserción en el archivo.
- c) Indica si el archivo cambia mucho o poco.
- d) Indica la edad del archivo.

**T13.31.** El factor de bloqueo indica:

- a) El número de bloques (registros físicos) que caben en un registro lógico.
- b) El número de registros lógicos que caben en un registro físico.
- c) El número de unidades de ubicación que caben en un disco.
- d) El número de unidades de ubicación que caben en un archivo determinado.

**T13.32.** El cierre de un archivo se produce cuando:

- a) Se llega a la marca EOF.
- b) Se elimina de la memoria principal el descriptor del archivo.
- c) Se escribe en el dispositivo de memoria masiva.
- d) Alcanza su capacidad máxima de almacenamiento.

**T13.33.** ¿Cuál de las siguientes afirmaciones es cierta?

- a) La versión FAT16 del sistema operativo MS-DOS utiliza unidades de ubicación mayores que la versión FAT32, aunque las capacidades de disco que admite son menores.
- b) La versión FAT32 del sistema operativo MS-DOS utiliza unidades de ubicación mayores que la versión FAT16, y por tanto las capacidades de disco que admite son mayores.
- c) La versión FAT16 del sistema operativo MS-DOS utiliza unidades de ubicación mayores que la versión FAT32, por lo que la capacidad de disco que admite son mayores.
- d) La versión FAT32 del sistema operativo MS-DOS utiliza unidades de ubicación mayores que la versión FAT16, aunque las capacidades de disco son menores.



## Problemas resueltos



## ORGANIZACIÓN DE ARCHIVOS

**P13.1.** Describir los pasos a seguir para actualizar un archivo secuencial denominado EMPLEADOS, que tiene información sobre el personal de una empresa.

## SOLUCIÓN

El fichero de EMPLEADOS será el fichero maestro de la aplicación, ya que contiene los datos más importantes de la misma. Lo lógico es que los registros de este archivo estén ordenados por DNI, que se utilizará como clave.

El conjunto de alteraciones deben incluirse en un archivo de ALTERACIONES, que contendrá los registros a modificar, los registros a dar de baja y los registros a dar de alta. Estos registros incluirán un campo adicional (al final, por ejemplo), indicando si son altas (*i*), bajas (*b*) o modificaciones (*m*). Estos registros también se ordenarán según el DNI.

Las actualizaciones las realizará un programa (ACTUALIZACIONES) que leerá secuencialmente los registros de los archivos ALTERACIONES y EMPLEADOS, y generará un nuevo archivo EMPLEADOS\_vn; todo ello de la siguiente forma:

1. Leer registro de ALTERACIONES.
2. Leer registro de EMPLEADOS.
3. Si DNI\_EMPLEADO es menor que DNI\_ALTERACIONES:
  - a) Escribir registro de EMPLEADOS en EMPLEADOS\_vn.
  - b) Ir al paso 2.
4. Si DNI\_EMPLEADO es igual a DNI\_ALTERACIONES:
  - a) Si campo de alteración es baja, ir a paso 1.
  - b) Si campo de alteración es modificación:
    - i. Escribir registro de ALTERACIONES en EMPLEADOS\_vn.
    - ii. Ir a paso 1.
5. Si DNI\_EMPLEADO es mayor de DNI\_ALTERACIONES:
  - a) Si campo de alteración no es alta, dar mensaje de error.
  - b) Si campo de alteración es alta:
    - i. Escribir registro de ALTERACIONES en EMPLEADOS\_vn.
    - ii. Escribir registro de EMPLEADOS en EMPLEADOS\_vn.
    - iii. Ir a paso 1.

**P13.2.** Dados los archivos maestro y de modificaciones de las Tablas 13.3 y 13.4, que representa una lista de las veinte canciones más populares del 18 de octubre de 2004, y cuyos registros contienen clave, el nombre del cantante, la canción y el orden de popularidad, obtener el archivo maestro nuevo.

## SOLUCIÓN

Al ser un archivo secuencial, el programa de actualización va generando un archivo nuevo de la siguiente manera: se lee el primer registro y si no tiene ninguna alteración, se graba en el nuevo, y así sucesivamente.

- Cuando se lee 4654, como es una eliminación, deja de escribirse en el nuevo archivo.
- Al llegar al registro 4658, como es una modificación, se escribe el registro nuevo tal como está en el archivo de alteraciones, en lugar de copiar el registro leído del archivo maestro original.
- Al llegar al 5643, se produce una nueva baja.



0056	<b>Orozco, Antonio</b> “Estoy hecho de pedacitos de ti”	15
0224	<b>Marc Anthony</b> “Valió la pena”	10
1143	<b>Keane</b> “Everybody’s changing”	11
2357	<b>Bisbal, David</b> “Camina y ven”	8
2543	<b>Demaría, David</b> “Precisamente ahora”	1
2733	<b>Green Day</b> “American idiot”	7
2733	<b>Vives, Carlos</b> “Como tú”	13
3456	<b>Anastacia</b> “Sick and tired”	9
4224	<b>Juanes</b> “Nada valgo sin tu amor”	3
4527	<b>Belinda</b> “No entiendo”	18
4654	<b>Erentxun, Mikel</b> “Esos días”	17
4658	<b>Bebe</b> “Ella”	5
4673	<b>O-Zone</b> “Despre tine”	6
4832	<b>García, Manolo</b> “Para que no se duerman mis sentidos”	4
5643	<b>Ubagó, Alex</b> “Fantasía o realidad”	12
6637	<b>R.E.M.</b> “Leaving New York”	20
6852	<b>Mojinos Escosíos</b> “Al carajo”	14
7354	<b>Melendi</b> “Con la luna llena”	2
7354	<b>Mürfila</b> “Mi guitarra quiere rock”	16
8329	<b>Canto del Loco, El</b> “Una foto en blanco y negro”	19

Tabla 13.3. Archivo maestro original.

4654			b
4658	<b>Bebe</b> “Ella”	2	m
5643			b
5656	<b>Suff, Hilary</b> “Fly”	20	i
6637	<b>R.E.M.</b> “Leaving New York”	12	m
7354	<b>Melendi</b> “Con la luna llena”	5	m
7427	<b>Nelly</b> “My place”	17	i

i: Insertar registro.

b: Borrar registro.

m: Modificar registro.

Tabla 13.4. Archivo de alteraciones

- El próximo registro leído del archivo maestro original es el 6637, y como en el archivo de alteraciones figura el 5656 (anterior al 6637), y es un alta, se inserta este último registro en el maestro nuevo.
- Etcétera.

El archivo resultante es el que se indica en la Tabla 13.5.

0056	<b>Orozco, Antonio</b> “Estoy hecho de pedacitos de ti”	15
0224	<b>Marc Anthony</b> “Valió la pena”	10
1143	<b>Keane</b> “Everybody’s changing”	11
2357	<b>Bisbal, David</b> “Camina y ven”	8
2543	<b>Demaría, David</b> “Precisamente ahora”	1
2733	<b>Green Day</b> “American idiot”	7
2733	<b>Vives, Carlos</b> “Como tú”	13
3456	<b>Anastacia</b> “Sick and tired”	9
4224	<b>Juanes</b> “Nada valgo sin tu amor”	3
4527	<b>Belinda</b> “No entiendo”	18
4658	<b>Bebe</b> “Ella”	2

(continúa)

Tabla 13.5. Archivo maestro nuevo.

4673	<b>O-Zone</b> “Despre tine”	6
4832	<b>García, Manolo</b> “Para que no se duerman mis sentidos”	4
5656	<b>Suff, Hilary</b> “Fly”	20
6637	<b>R.E.M.</b> “Leaving New York”	12
6852	<b>Mojinos Escocíos</b> “Al carajo”	14
7354	<b>Melendi</b> “Con la luna llena”	5
7354	<b>Mürfila</b> “Mi guitarra quiere rock”	16
7427	<b>Nelly</b> “My place”	17
8329	<b>Canto del Loco, El</b> “Una foto en blanco y negro”	19

**Tabla 13.5.** Archivo maestro nuevo (*continuación*).

**P13.3.** Suponga que en un archivo se prevén como máximo 10.000 registros y las claves están comprendidas entre 0 y 9999999. Se desea acceder en un archivo a los registros con las siguientes claves:

1123456  
1735332  
5987623  
2754482  
8564239  
6666732  
7543210  
9999999

Indicar las direcciones de los registros que se obtendrían con:

- a) Organización directa, acceso directo.
- b) Organización directa, acceso calculado con la función módulo.
- c) Organización directa, acceso calculado extrayendo las cifras 1.<sup>a</sup>, 3.<sup>a</sup>, 5.<sup>a</sup> y 7.<sup>a</sup>
- d) Organización directa, acceso calculado elevando al cuadrado.

**SOLUCIÓN**

- a) Organización directa, acceso directo.

En este caso la dirección coincide con la clave:

Clave	Dirección
1123456	1123456
1735332	1735332
5987623	5987623
2754482	2754482
8564239	8564239
6666732	6666732
7543210	7543210
9999999	9999999

Obsérvese que, como puede accederse a cualquier dirección comprendida entre 0 y 9.999.999, este sistema de acceso obligaría a tener espacio en disco para 10.000.000 de registros, a pesar de que sólo habría como máximo 10.000 registros.

- b) Organización directa, acceso calculado con la función módulo.

Como se prevé como máximo 10.000 registros realizamos la función módulo con 10.007 (que es el número primo inmediato superior al número de registros), con lo que las direcciones son:

Clave	Dirección
1123456	2673
1735332	4122
5987623	3438
2754482	2558
8564239	8255
6666732	2071
7543210	7940
9999999	3007

c) Organización directa, acceso calculado extrayendo las cifras 1.<sup>a</sup>, 3.<sup>a</sup>, 5.<sup>a</sup> y 7.<sup>a</sup>

Clave	Dirección
1123456	1246
1735332	1332
5987623	5863
2754482	2542
8564239	8629
6666732	6672
7543210	7420
9999999	9999

d) Organización directa, acceso calculado elevando al cuadrado.

Clave	Cuadrado	Dirección
1123456	1262153384000	1533
1735332	3011377150000	3771
5987623	35851629190000	6291
2754482	7587171088000	1710
8564239	73346189650000	1896
6666732	44445315560000	3155
7543210	56900017100000	3155
9999999	99999980000001	9800

**P13.4.** ¿Por qué el archivo del Ejemplo 13.4 que contiene información de los 23.742 alumnos de una universidad y que utiliza como clave el número de DNI no se debe gestionar por organización directa con acceso directo? ¿Cómo se podría modificar el diseño del archivo para poderlo gestionar con acceso directo?

#### SOLUCIÓN

Hay cien millones (100.000.000) de números de DNI posibles, con lo que si utilizamos el DNI como dirección de acceso, tendríamos que reservar espacio para cien millones de registros, lo cual es extraordinariamente excesivo, sobre todo cuando de todo ese espacio sólo utilizaríamos el de 23.000 registros (ocuparíamos sólo un 0,023 por 100 del espacio reservado).

Si se desea utilizar direccionamiento directo, podríamos asignar un número correlativo de identificación a cada alumno (que figuraría, por ejemplo, en un carné universitario) y utilizar este número como clave.

**P13.5.** Se desea disponer de un archivo para localizar en España el nombre y dirección de una persona o entidad a partir de su número de teléfono<sup>3</sup>. Suponiendo que hay 50 millones de habitantes y que, por término medio, una de cada 43,75 personas tiene teléfono, indicar:

<sup>3</sup> En España los números de teléfono se componen de 9 cifras decimales, correspondiendo las dos o tres primeras a la localidad. Así, por ejemplo, los teléfonos que comienzan por 91 son de Madrid, los que comienzan por 93 son de Barcelona, los que lo hacen por 923 son de Salamanca y los que lo hacen por 958 de Granada, etc.

- a) La capacidad estimada del archivo.
- b) El tipo de organización que considera más adecuado para el archivo.
- c) Por qué no es buena idea utilizar acceso calculado, utilizando como dirección los tres primeros dígitos.

*Sugerencias:* Suponer un tiempo de acceso al registro de 10 ms, y unidades de ubicación en disco de 512 Bytes.

#### SOLUCIÓN

- a) Tipo de organización.

El archivo contendrá alrededor de 875.000 registros. La estructura de los registros puede ser la siguiente:

- Número de teléfono: 9 decimales.
- Nombre y apellidos: 25 letras.
- Domicilio: 24 letras.
- Ciudad: 2 letras.

Es decir, cada registro ocupará 60 caracteres.

El tamaño total del archivo será:  $875.000 \cdot 60 = 50$  MBytes.

Una organización secuencial no sería útil, ya que, por término medio, para localizar un registro se tardaría (suponemos el tiempo de acceso a un registro 10 ms):

$$t_{\text{acceso}} = \frac{875.000}{2} \cdot 10 \cdot 10^{-3} = 4.375 \text{ s} \approx 1 \text{ hora } 10 \text{ minutos}$$

tiempo totalmente inaceptable.

Una solución posible es organizar el archivo con direccionamiento calculado, y, por ejemplo, se podría obtener la dirección del segmento, extrayendo las 5 últimas cifras decimales. De esta forma se tendrían cien mil segmentos, y, por término medio, en cada uno de ellos se ubicarían  $875.000/100.000 = 8,75$  registros. Conviene hacer notar lo siguiente:

- Los registros asociados a los números de teléfono se deben dispersar entre todos los segmentos de la forma más homogénea posible. La elección de las últimas cifras decimales para identificar el segmento resulta adecuada, ya que sin duda los números así formados tendrán una distribución uniforme.
- Como cada segmento va a ocupar, por término medio,  $8,75 \cdot 60 = 525$  Bytes, cada segmento ocupará más de una unidad de ubicación. El sistema funcionaría más rápidamente si cupiese cada segmento en una unidad de ubicación, para ello debemos incluir como máximo en cada segmento  $512/60 \leq 8,53$  registros; con lo que el número de segmentos debería ser:  $875.000/8 = 109.375$  segmentos, utilizando un bloque de 8. El número primo inmediato superior a 109.375 que es **109.379**, luego definitivamente utilizamos este último número como número de segmentos.

En definitiva se sugiere una organización directa con acceso calculado por extracción. Dado un número de teléfono, por ejemplo 958123695, el segmento donde estaría el registro correspondiente sería el 23695, y en una tabla cargada en la memoria principal podríamos obtener la dirección física de la unidad de ubicación donde se encuentra el registro, y cargar la información de toda esta unidad de ubicación en la memoria principal. Aquí, con acceso secuencial, obtendríamos el registro buscado.

- b) Problema de direccionar con los tres primeros dígitos.

Tal y como se asignan los números de teléfono, los 3 primeros dígitos corresponden a la provincia donde reside el teléfono; como la distribución de población no es uniforme entre provincias, la dispersión de claves entre segmentos no sería uniforme: algunos segmentos estarían poco ocupados y en otros habría una cantidad muy grande de sinónimos, aumentando notablemente los tiempos en las búsquedas secuenciales a realizar.

**P13.6.** Se dispone de un archivo con organización directa y acceso calculado, y el segmento donde se debe ubicar cada registro se obtiene con la función módulo.

- a) Suponiendo que hay 8 segmentos, indicar en cuál de ellos deben situarse los registros cuyas claves son: 24; 38; 40; 16; 24; 30; 32; 46.

- b) ¿Qué problemas se presentan a causa del número de segmentos elegidos?
- c) Determinar los segmentos donde deben ubicarse los registros mencionados anteriormente, suponiendo que hay 9 en lugar de 8 segmentos.

#### SOLUCIÓN

- a) Segmentos donde se ubican los registros:

El segmento es el *módulo*(llave,8); es decir, el resto del cociente de la división entera entre la llave y el número de segmentos (8, en este caso).

Llave del registro	Segmento de ubicación
24	0
38	6
40	0
16	0
24	0
30	6
32	0
46	6

- b) ¿Qué problemas se presentan a causa del número de segmentos elegidos?

Puede observarse que la función elegida no es adecuada, dado que no *dispersa* los registros entre todos los segmentos. Ello se debe a que, al no ser 8 primo, se van a encontrar frecuentemente factores comunes entre el numerador y denominador, lo que va a generar restos iguales.

- c) Segmentos donde se ubican los registros, utilizando 9 segmentos.

Llave del registro	Segmento de ubicación
24	6
38	2
40	4
16	7
24	6
30	3
32	5
46	1

Puede observarse que ahora sólo se producen dos colisiones. La reducción de colisiones es debida a que al ser 9 primo, no va a tener nunca denominador factores comunes con el numerador.

**P13.7.** Se desea disponer de un archivo para comprobar el deletreo de textos en castellano que contendrá un total de 83.500 palabras. Suponer que el archivo se almacenará en un disco con unidades de ubicación de 512 Bytes, que se utiliza ASCII-Latín I y que el tiempo medio de acceso y lectura de un registro es de 10 ms.

- a) Sugerir una forma adecuada para estructurar el archivo.
- b) Hacer una estimación de la capacidad del archivo.
- c) ¿Cuál sería el factor de bloqueo?

#### SOLUCIÓN

- a) Sugerir una forma adecuada para estructurar el archivo.

- Una organización secuencial no es adecuada, ya que se tardaría mucho en buscar una palabra. Suponiendo que cada palabra ocupa un registro lógico y un tiempo de acceso al registro de 10 ms, y con la hipótesis de que el acceso a la palabra es equiprobable, el tiempo medio de acceso sería:

$$t = \frac{83.500}{2} \cdot 10 = 417.500 \text{ ms} \approx 7 \text{ minutos}$$

que es un tiempo excesivo.

- Tampoco resulta adecuado un fichero indexado, ya que el tamaño de la tabla de índices (167.000 datos) y el tiempo de consulta a la misma, para, a partir de la palabra, obtener su dirección resultarían excesivos.
- Una posible solución es utilizar acceso parcialmente indexado, en donde cada segmento podría corresponder a la letra de comienzo de la palabra; es decir, habría 27 segmentos y en la tabla de índices se detallarían las direcciones de comienzo de cada segmento.
- Otra solución, la más adecuada, es utilizar acceso calculado con extracción de la primera letra de la palabra. Como en el acceso parcialmente indexado, habría 27 segmentos, cada uno de ellos correspondiente a la letra de inicio. A cada letra se le haría corresponder su valor decimal ( $a \rightarrow 97$ ,  $b \rightarrow 98$ , etcétera) y de hay obtendríamos la dirección donde se encontrase la palabra. No sería necesario, por lo tanto mantener una tabla de índices en la memoria principal y tenerla que consultar para obtener las direcciones de las palabras. Dentro de cada segmento (cada letra inicial) se buscaría la palabra concreta utilizando una organización; por ejemplo, secuencial indexada. Obsérvese que así los segmentos pueden ser de capacidad variable, lo que realmente ocurre con las palabras, ya que no hay el mismo número de ellas que empiecen con distinta letra: hay muchas más que empiezan por “a” que por “b”, y por “b” que por “x”, por ejemplo.

**b)** Hacer una estimación de la capacidad del archivo.

Suponiendo que para cada palabra se reservan 16 caracteres; es decir, 16 Bytes, la ocupación del archivo sería:

$$C = 83.500 \cdot 16 = 1.336.000 \text{ Bytes} = 1,3 \text{ MB}$$

**c)** ¿Cuál sería el factor de bloqueaje?

Como hemos supuesto que cada registro ocupa 16 Bytes, y el tamaño de la unidad de ubicación es de 512 Bytes, el factor de bloqueaje sería:

$$B = \frac{512}{16} = 32$$

Es decir, en cada registro físico se ubicarán 32 registros lógicos.

## PARÁMETROS DE UTILIZACIÓN

**P13.8.** Estimar la capacidad de almacenamiento que requiere un archivo de texto ASCII-Latín-1, de 35 páginas en formato DIN-A4.

### SOLUCIÓN

En un documento MS-Word con la instrucción Herramientas/Contar palabras contamos el número de caracteres de una página y obtenemos 2.900. Entonces, aproximadamente, en 35 páginas habrá unos  $2.900 \cdot 35 = 101.500$  caracteres. Como están almacenados en código Latín 1, cada carácter ocupa 1 Byte; es decir, la capacidad de almacenamiento requerida es:

$$C = 101.500 \text{ Bytes} \approx 100 \text{ KBytes}$$

**P13.9.** Una librería gestiona sus existencias utilizando un archivo cuyos registros contienen la siguiente información: ISBN, autor, título, edición, editorial y precio. Si se prevé que se disponga en existencias un máximo de 5.000 libros, estimar la capacidad que ocuparán los datos del archivo.

### SOLUCIÓN

Suponemos los siguientes tamaños para los campos de cada registro:

- ISBN: 13 caracteres.
- Autor(es): 60 caracteres.

- Edición: 1 carácter.
- Editorial: código de 3 caracteres.
- Precio: 6 caracteres.

Es decir, cada registro ocupará 83 caracteres. Con lo que el número total de caracteres será:

$$5.000 \cdot 83 = 415.000 \text{ caracteres}$$

Si se almacena en ASCII-Latín 1, la capacidad necesaria para almacenar todo el archivo será:

$$C = 415.000 \text{ Bytes} = 406 \text{ KBytes}$$

En Unicode ocuparía el doble.

**P13.10.** Se dispone de un archivo secuencial con 80.000 registros; suponiendo que el tiempo medio de acceso a un registro fuese de 6 ms, cuánto tiempo se tardaría en acceder a:

- El primer registro.
- El registro de en medio.
- El último registro.
- Obtener el tiempo de acceso medio.

#### SOLUCIÓN

- Primer registro.

Al abrir el fichero se accede al primer registro, con lo que el tiempo de acceso será:

$$t_{\text{acceso}} = 6 \text{ ms}$$

- Registro de en medio.

Para leer el registro intermedio (el que ocupa la posición 40.000) habrá que haber leído previamente todos los anteriores, con lo que:

$$t_{\text{acceso}} = 40.000 \cdot 6 = 240.000 \text{ ms} = 4 \text{ minutos}$$

- Último registro.

Para leer el último registro, suponiendo que previamente no se ha leído ninguno anterior, habrá que leer todos los anteriores, con lo que:

$$t_{\text{acceso}} = 80.000 \cdot 6 = 8 \text{ minutos}$$

- Tiempo de acceso medio.

Suponiendo que estadísticamente las frecuencias de acceso a los registros son equiprobables, el tiempo de acceso medio será igual al tiempo de acceso del registro de en medio; es decir,

$$t_{\text{acceso}} = 4 \text{ minutos}$$

Este mismo resultado se obtiene, considerando que los tiempos de acceso a los distintos registros son:

1.º registro:	6 ms
2.º registro:	12 ms
3.º registro:	18 ms
.....	.....
40.000 registros:	240.000 ms
41.000 registros:	240.006 ms
.....	.....
79.999 registros:	479.994 ms
80.000 registros:	480.000 ms

Haciendo la media de los tiempos, se obtiene: 240.000 ms = 4 minutos.

$$t_{\text{acceso}} = \frac{6 + 12 + 18 + \dots + 480.000}{80.000} \text{ ms} = 240.000 \text{ ms}$$

**P13.11.** Un fichero maestro de 65.000 registros se somete a una actualización, en la que se producen 1.255 bajas, 7.425 alteraciones de registros y 1.535 altas. Determinar:

- a) La tasa de modificación.
- b) La tasa de renovación.
- c) La tasa de crecimiento.

SOLUCIÓN

a) Tasa de modificación:

$$\text{Tasa de modificación} = \frac{\text{número de registros modificados}}{\text{número total de registros}} = \frac{1.255 + 7.425}{65.000} = \frac{8.680}{65.000} = 0,13$$

b) Tasa de renovación:

$$\text{Tasa de renovación} = \frac{\text{número de registros alterados}}{\text{número total de registros}} = \frac{7.425}{65.000} = 0,11$$

c) Tasa de crecimiento:

$$\text{Tasa de crecimiento} = \frac{\text{incremento de registros}}{\text{número total de registros}} = \frac{1.535 - 1.255}{65.000} = \frac{280}{65.000} = 0,004$$

### EJEMPLO: GESTIÓN DE ARCHIVOS EN WINDOWS Y LINUX

**P13.12.** Calcular el tamaño máximo de un archivo de una versión de UNIX que utiliza el esquema de direccionamiento de bloques mostrado en la Figura 13.10, y suponiendo que el tamaño del bloque de disco es de 2 KB.

SOLUCIÓN

Primero obtenemos el número de bloques que se puede direccionar:

$$N_{\text{bloques}} = 10 + 256 + 256 \cdot 256 + 256 \cdot 256 \cdot 256 = 16.843.018 \text{ bloques}$$

Como cada bloque puede contener 2 KBytes, la capacidad máxima de los archivos será:

$$C_{\text{archivo}} = N_{\text{bloques}} \cdot C_{\text{bloque}} = 16.843.018 \cdot 2 \text{ KB} \approx 32 \text{ GBytes}$$

## Problemas propuestos



### TIPOS DE ARCHIVOS

**P13.13.** Suponga que en una ferretería controlan sus existencias con una aplicación informática que utiliza un archivo de inventario. Este archivo contiene registros con los distintos productos que se comercializan: código, modelo, proveedor, número de unidades de que se dispone, número mínimo de unidades, precio de compra, precio de venta, etc. Todas las ventas se van registrando en un archivo de ventas. Todos los días, al cerrar la ferretería, se actualiza el archivo de inventario, y se genera un archivo de pedidos, con aquellos productos cuyo número sea inferior al mínimo establecido. A partir del

archivo de pedidos se generan automáticamente cartas de pedido para los distintos suministradores. Determinar:

- a) El tipo de cada uno de los archivos que se utiliza.
- b) La estructura de los registros de cada archivo.

### ORGANIZACIÓN DE ARCHIVOS

**P13.14.** Suponga que se dispone de un archivo, denominado EMPLEADOS, que contiene información sobre los trabajadores de una empresa, y está organizado como secuencial enca-



denado, estando ordenado por apellidos del empleado. Describir los pasos a seguir para:

- Dar de baja a un empleado (José Pérez López).
- Insertar el registro dado de baja en el archivo HISTORICO\_EMPLEADOS (con igual organización).

**P13.15.** Obtener un archivo de índices para el archivo de la Tabla 13.6.

Clave	Hotel	Ciudad
453217	Riazor	La Coruña
632004	Parador Reyes Católicos	Santiago
032417	Torremangana	Cuenca
253764	Costabella	Gerona
063227	Albayzin	Granada
542319	Pax	Guadalajara

**Tabla 13.6**

**P13.16.** Suponga que una parte de una tabla de índices de un archivo secuencial indexado es la que se indica en la Tabla 13.7.

Clave	Dirección de segmento
3243	7ABC4
5427	9BA32
7226	A3742
9548	AB451
9957	B1324

**Tabla 13.7.**

Indicar la dirección de los segmentos a los que debe accederse para localizar los registros cuyas claves son:

- 4723.
- 6348.
- 9732.
- 8432.

**P13.17.** Una librería gestiona sus existencias utilizando un archivo cuyos registros contienen la siguiente información: ISBN, autor, título, edición, editorial y precio. Discuta las ventajas e inconvenientes de utilizar cada una de las organizaciones estudiadas en este capítulo. Suponer que como máximo se dispondrá de 5.000 libros y nunca se dispondrá de más de un ejemplar de cada uno de ellos.

**P13.18.** Indicar cómo podrían gestionarse la información de los clientes de un banco con un archivo con organización indexada. ¿Cómo se efectuarían las operaciones de insertar, modificar y borrar registros? En su caso, sugiera una organización más eficiente.

**P13.19.** Suponga que en un archivo se prevén como máximo 900 registros, cada uno de ellos conteniendo un total de 64 Bytes. Como clave se utiliza el número de Documento Nacional de Identidad (DNI).

- Indicar la capacidad máxima que ocupará el archivo (tener sólo en cuenta los datos del usuario).
- Suponiendo que se almacena en una unidad de disco, con unidades de ubicación de 512 Bytes, ¿qué capacidad máxima ocuparía el archivo?
- ¿Qué problemas plantearía utilizar una organización directa con acceso directo?
- Esbozar cómo se podría gestionar el archivo con acceso indexado. ¿Cuánto ocuparía la tabla de índices?

**P13.20.** Suponga que el archivo del problema anterior (P13.19) se gestiona con organización directa, y que se desea acceder a los registros con los siguientes números de DNI:

- 7.742.979.
- 43.317.523.
- 28.432.523.
- 5.324.432.

Indicar las direcciones de los registros que se obtendrían con:

- Acceso directo.
- Acceso calculado con la función módulo.
- Acceso calculado extrayendo las cifras 1.<sup>a</sup>, 3.<sup>a</sup>, 5.<sup>a</sup> y 7.<sup>a</sup>
- Acceso calculado elevando al cuadrado.

**P13.21.** Estimar el número de unidades de ubicación que ocupará en disco un archivo de texto Unicode de 97 páginas, suponiendo que las unidades de asignación son de 2 KBytes.

**P13.22.** Se desea almacenar en un disco un archivo de 40.000 registros, de longitud fija, de 120 caracteres cada uno. Las unidades de ubicación del disco son de 512 Bytes, y el tiempo medio de acceso a un registro es de 6 ms. Los caracteres se almacenan con código ASCII-Latín-1. ¿Cuál es el tiempo medio de localización de un registro para cada una de las organizaciones? ¿Cuál será la capacidad de éste, si cada puntero ocupa dos Bytes, y la clave de búsqueda es de diez caracteres? ¿Cómo variarían los parámetros anteriores si los caracteres se almacenasen en Unicode?

**P13.23.** Se dispone de un archivo de 35.000 registros, cada uno de ellos de 80 caracteres. Se observa que, por término medio, al día se hacen 2.500 consultas, 39 bajas, 55 altas y 325 modificaciones de registros. Determinar:

- La frecuencia de consulta.
- La frecuencia de modificación.
- La frecuencia de renovación.
- La frecuencia de crecimiento.

### EJEMPLO: GESTIÓN DE ARCHIVOS EN WINDOWS Y LINUX

**P13.24.** Calcular el tamaño máximo de un archivo de una versión de UNIX en la que el tamaño del bloque de disco es de 2 KB, en el *i*-nodo se dan directamente las direcciones de 10 bloques, que cada tabla de direccionamiento de segundo y tercer nivel contiene 512 punteros, y la dirección de cada bloque se especifica con 8 Bytes. ¿Cuál sería la capacidad máxima del sistema de direccionamiento descrito?

**P13.25.** En la versión 4.2BSD de UNIX los nodos de índices tienen 12 punteros directos, el tamaño de las unidades de asignación (bloques) es de 4 KB, cada puntero ocupa 4 Bytes. Sabiendo que el tamaño máximo de los archivos reales es de 2 GB:

- a)* ¿Cuál es la capacidad máxima de los archivos que se pueden direccionar directamente?
- b)* ¿Cuántos niveles de direccionamiento sería necesario utilizar?

- c)* ¿Cuáles son las ventajas e inconvenientes de utilizar distintos niveles para el direccionamiento de las unidades de asignación de los datos?

**P13.26.** Comparar la forma de almacenar archivos de 1 Kbytes, 10 KBytes, 100 KBytes, 1 MBytes, 10 MBytes y 100 MBytes en MS-DOS Windows (con FAT16, FAT32) y UNIX. Indicar las ventajas e inconvenientes (fragmentación de memoria) de los distintos sistemas.



# Bases de datos

En el capítulo anterior estudiamos los archivos como herramienta que permite organizar y memorizar conjuntos de datos del mismo tipo o naturaleza con una determinada estructura y que se conciben como un medio para almacenar los datos o resultados de una aplicación determinada. La dependencia de los archivos de las aplicaciones para las que han sido diseñados presenta serios inconvenientes, como la duplicidad innecesaria de información dentro de los archivos de una misma entidad u organismo, incoherencia de datos, falta de seguridad, etc.

Para solucionar los problemas citados surge el concepto de base de datos. La idea básica consiste en unificar toda la información de una empresa o entidad en un único sistema de almacenamiento, de forma independiente de las aplicaciones pero pudiendo todas ellas hacer uso de dicha información. De esta forma los datos adquieren un protagonismo por sí mismo de primera fila, y el desarrollo de aplicaciones resulta más cómodo mejorando notablemente la calidad de la información generada por las mismas.

Este capítulo presenta una introducción al concepto de base de datos, analizando su estructura, los modelos que se han propuesto y los distintos tipos de implementación.

## 14.1. Conceptos generales

Tradicionalmente cuando se desarrolla una aplicación concreta se diseñan los archivos que deben contener los datos de entrada, intermedios y de salida. Recuérdese que los datos introducidos o generados por los usuarios en la memoria principal (memoria RAM) se pierden al desconectar el computador, siendo, por tanto, necesario almacenarlos en la memoria masiva en forma de archivos. En general, un archivo está concebido como un medio para organizar los datos o resultados de una aplicación determinada. De esta forma con frecuencia se plantea en empresas e instituciones el problema de que gran cantidad de la información que contiene en sus archivos de memoria masiva está innecesariamente duplicada, ya que muchas aplicaciones utilizan datos comunes. Por ejemplo, en una empresa es muy probable que exista un archivo para producir la nómina de sus empleados. Este archivo, entre otros datos contendrá, para cada empleado: DNI, nombre, dirección, teléfono, antigüedad, puesto desempeñado, etc. Otro departamento de la misma empresa puede encargarse de asignar los trabajos a los empleados, y, con mucha probabilidad, tendrá un archivo con los mismos datos generales de cada empleado citados anteriormente junto con los más específicos de la aplicación de asignación de trabajos. También puede haber otros archivos, relacionados, por ejemplo, con elecciones sindicales, etc.

Los problemas que presenta el tipo de organización descrito en el párrafo anterior son evidentes: ocupación de memoria masiva innecesaria, mayor carga de trabajo a la hora de cambiar de situación de un empleado (al dar de alta o de baja a un empleado, por ejemplo, habrá que hacerlo en todos los archivos), incoherencia de los datos (puede ser que la información esté más actualizada en unos archivos que en otros), poca seguridad en los datos, etc. El concepto de base de datos se introdujo para solucionar estos problemas: en una organización o empresa se proyecta un sistema (base de datos) que contendrá todos los datos que necesitan las distintas aplicaciones; es decir, se logra independizar los datos de las aplicaciones, con las consiguientes ventajas, como más adelante veremos, en esta misma sección.

Una **base de datos** es un sistema formado por un conjunto de datos y un software para su gestión, de forma que se controle el almacenamiento de datos redundantes, exista independencia entre los datos y los programas que los usan, se almacenen las relaciones entre los datos junto con éstos, y se pueda acceder a los datos con facilidad y de distintas formas. En un sistema de archivos sólo hay datos y sólo se puede acceder a ellos de una manera predeterminada. Las bases de datos son la evolución de los sistemas de archivos y suplen las deficiencias de éstos, ofreciendo las siguientes ventajas:

- **Acceso múltiple.** Diversos usuarios o aplicaciones pueden acceder a la base de datos, sin que se produzcan conflictos ni visiones incoherentes.
- **Utilización múltiple.** Cada usuario o aplicación puede tener una imagen o visión particular de la estructura de la base de datos, accediendo sólo a los datos de su interés.
- **Flexibilidad.** Se pueden usar distintos métodos de acceso, con tiempos de respuesta razonablemente pequeños.
- **Confidencialidad y seguridad.** Se controla el acceso a los datos (a nivel de campo), impidiéndoselo a usuarios no autorizados. Uno en concreto podrá acceder a unos datos y a otros no.
- **Protección contra fallos.** Existen protocolos y mecanismos bien definidos de recuperación en caso de fallo del computador.
- **Independencia física.** Se puede cambiar el soporte físico de la base de datos (modelo de discos, por ejemplo), sin que esto repercuta en la base de datos ni en los programas que la usan.
- **Independencia lógica.** Es posible modificar los datos contenidos en la base de datos, las relaciones existentes entre ellos o incluir nuevos datos, sin afectar a los programas que las usan.
- **Redundancia controlada.** Los datos se almacenan una sola vez, salvo que se prevea lo contrario.
- **Interfaz de alto nivel.** Existe una forma sencilla y cómoda de utilizar la base de datos al menos desde un lenguaje de programación de alto nivel.
- **Consulta directa (*query*).** Se dispone de una utilidad que permite al acceso a los datos de forma conversacional (interactiva).

## 14.2. Estructura de una base de datos

En general se puede decir que una base de datos consta de los siguientes elementos:

- **Entidades:** objetos o elementos que se almacenan en la base de datos. Así en una base de datos académica podrá haber información de las siguientes entidades: alumno, profesor, asignatura, centro, plan de estudios, curso, etc.
- **Atributos:** datos de la entidad. Puede ser atributo de una entidad cualquier característica o propiedad de ésta. Así son atributos de la entidad alumno: DNI, apellidos y nombre, sexo, fecha de nacimiento, nacionalidad, etc.

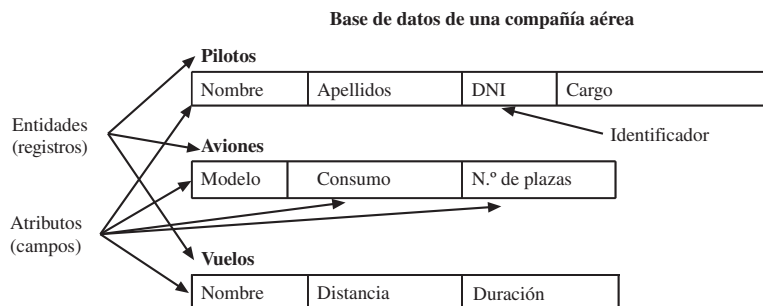
- **Registros:** donde se almacena la información de cada entidad
- **Campos:** donde se almacenan los atributos de cada registro.

En una base de datos hay *diferentes tipos de registros*, uno por entidad. Normalmente se reserva el nombre de “registro” para especificar un “tipo de registro”, usándose otras denominaciones para especificar cada una de las apariciones de ese registro en la base de datos, tales como: *elemento*, *valor actual de registro*, *valor en curso*, *instancia u ocurrencia del registro*.

Normalmente no es necesario conocer los valores de todos los atributos de una entidad para determinar si dos elementos son iguales. Por lo general, es suficiente con conocer el valor de uno o varios atributos para identificar un elemento. Pues bien, diremos que un conjunto de atributos de una entidad es un **identificador** de dicha entidad si el valor de dichos atributos determina de forma unívoca cada uno de los elementos de la misma, y no existe ningún subconjunto de él que sea identificador de la entidad.

### EJEMPLO 14.1

Supóngase que se quiere informatizar la gestión de vuelos de una compañía aérea. En la Figura 14.1 se muestran algunas de las entidades (Pilotos, Aviones y Vuelos) y de los atributos de dichas entidades que compondrían la base de datos para dicha gestión. La entidad Avión tiene como atributos: Modelo, Consumo y Número de plazas. Para la entidad Pilotos, se podría utilizar como identificador el DNI.



**Figura 14.1.** Elementos de una base de datos.

A la hora de hacer una búsqueda de información en la base de datos se suele hacer según una **llave** de búsqueda, que es uno o varios campos cuyos valores permiten localizar rápidamente ocurrencias de un registro. Muchas veces la llave coincide con el identificador.

En las bases de datos también se almacenan las **relaciones** entre entidades que pueden ser más o menos complejas. Estas relaciones se explicitan con punteros que crea el sistema de gestión de la base de datos de forma que son transparentes al usuario.

Al definir una base de datos hay que especificar cada registro y los campos que lo forman, así como las relaciones entre ellos. El **esquema** de una base de datos es la definición lógica de la base de datos. La descripción lógica de una parte de la base de datos se denomina **subesquema**. El subesquema consigue aislar los programas de la estructura lógica de la base de datos, permite que varios usuarios utilicen distintas estructuras de una misma base de datos y limitar el acceso a distintas partes de la base de datos, para realizar sólo determinadas acciones.

### EJEMPLO 14.2

El esquema de una base de datos de una compañía aérea podría describirse como sigue.

Las entidades a considerar son: flotas de aviones, aviones, vuelos y pilotos. Podrían añadirse otras entidades, como azafatas, aeropuertos, etc.; para simplificar, no las vamos a tener en cuenta.

Los atributos de las distintas entidades podrían ser:

- *Flotas de aviones*: modelo de avión, número de asientos, autonomía (millas), revisión (número de horas de vuelo a las que hay que hacer una “gran parada”).
- *Aviones*: matrícula (identifica unívocamente al avión), flota (modelo de avión), número de horas de vuelo que lleva realizadas el avión.
- *Vuelos*: código de vuelo, origen, destino, hora de despegue, hora de aterrizaje, flota (modelo del avión). La hora se da en valores UTC (*Coordinated Universal Time*), sistema también conocido como GMT (*Greenwich Mean Time*).
- *Pilotos*: número de licencia de vuelo, Nombre, DNI, Dirección, Cargo, Flota para la que tiene licencia de vuelo.

Existen diversos tipos de relaciones, como por ejemplo:

- Flota de aviones con aviones; es una relación de *uno a muchos*, ya que cada avión pertenece a una única flota, y dentro de cada flota hay diversos aviones.
- Pilotos con flota de aviones; es una relación *muchos a uno*, ya que distintos pilotos pertenecen a una única flota.
- Pilotos con expedientes de pilotos; es una relación *uno a uno*, debido a que cada piloto está biunívocamente relacionado con su expediente. Suponemos que expediente es una relación compuesta por los datos de los distintos pilotos, con atributos tales como fecha de ingreso en la compañía, domicilio, etc.
- Pilotos con aviones; es una relación *muchos a muchos*, puesto que un piloto puede pilotar distintos aviones (siempre que sean de la misma flota) y un avión de una determinada flota puede ser pilotado por cualquier piloto que tenga licencia para dicha flota.

Se podría establecer un subesquema para determinar la programación mensual de un determinado piloto. En efecto, para ello se utilizarían los registros de pilotos, aviones y vuelos; no siendo necesario utilizar los registros de flotas. Ello es debido a que los vuelos asignados a un piloto no van a depender de los datos de los atributos de la entidad flota (número de asientos, autonomía de vuelo y número de horas de vuelo del avión). Tampoco intervendrían las entidades Azafata y Aeropuerto, si las considerásemos.

## 14.3. Sistemas de gestión de bases de datos

Desde un punto de vista conceptual, una base de datos puede considerarse formada por tres capas (Figura 14.2):

- La primera es la capa de aplicación que representa el software que permite la comunicación con el usuario, determinando las peculiaridades externas del uso de la base de datos e implementando la interfaz con el usuario.
- La siguiente capa corresponde al software de gestión de la base de datos, y se encarga de transformar las peticiones de la capa de aplicación en acciones concretas sobre la base de datos. Esta capa se suele denominar **sistema de gestión de la base de datos** (o **DBMS**, *Data Base Management System*), y, sencillamente, es el software destinado a la creación, control y manipulación de la información de la base de datos. Un DBMS debe permitir la realización de las siguientes tareas:
  - **Acceder** a los datos desde algún lenguaje de alto nivel.
  - **Consultar** (o recuperar información) en modo conversacional.

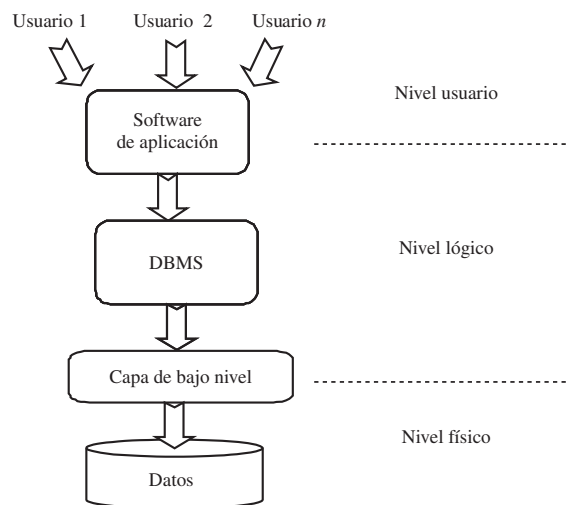


Figura 14.2. Estructura conceptual de una base de datos.

- **Definir** el esquema y subesquemas de la base de datos.
- **Organizar** físicamente la base de datos y recuperar el sistema si falla.
- La última capa, que suele estar integrada en el sistema operativo, es la que interactúa directamente con el hardware, determinando los lugares donde se almacenan físicamente los datos y cómo se transfieren los caracteres hacia y desde el dispositivo de almacenamiento masivo.

Esta estructura de tres capas tiene varias ventajas. El **DBMS** al actuar como intermediario entre los programas de aplicación y el sistema operativo permite que los programas sean independientes de la estructura física de los datos. También permite que el diseño software de estos sistemas sea más sencillo. Y sobre todo hace posible que haya un control de accesos a la base de datos. También posibilita cambiar la organización de la base de datos sin tener que cambiar el software de aplicación, lo que hace que los datos sean independientes de las aplicaciones.

Para definir la estructura lógica de la base de datos, es decir, para crear los esquemas y subesquemas, se utiliza un **lenguaje de descripción de datos (DDL)**. Con este lenguaje se enumeran los registros, los campos que forman cada registro y las relaciones entre ellos.

Para acceder a la base de datos, es decir, para leer, escribir o modificar datos, se utiliza un **lenguaje de manipulación de datos (DML)**. Las sentencias que forman este lenguaje dependerán del sistema de base de datos utilizado.

## 14.4. Modelos de bases de datos

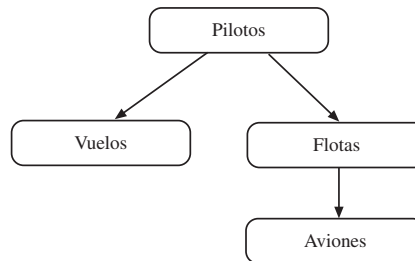
Tradicionalmente las bases de datos se clasifican en jerárquicas, en red y relacionales. Las bases de datos relacionales son las más usadas, y actualmente se están utilizando bajo formas tales como bases de datos distribuidas y bases de datos orientadas a objetos, como se analizará en la Sección 14.5.

### 14.4.1. MODELO JERÁRQUICO

Las bases de datos jerárquicas se organizan en forma de árbol descendente. Cada entidad sólo tiene un padre y puede tener varios hijos. Los registros se unen mediante relaciones uno a uno o uno a mu-



chos (no se permiten relaciones muchos a muchos). En la Figura 14.3 se muestra un posible esquema de base de datos jerárquica para el Ejemplo 14.2.

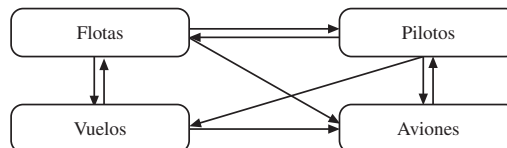


**Figura 14.3.** Ejemplo de base de datos jerárquica.

Las relaciones en una base de datos jerárquica se almacenan como estructuras en árbol (Sección 10.2.7). En efecto, en cada instancia de registro a continuación de los datos del padre se añaden dos campos, uno con un puntero al primer hijo y otro con un puntero al último hijo. Por otra parte, cada hijo también tiene añadido dos punteros, uno al anterior hermano y otro al próximo hermano.

#### 14.4.2. MODELO EN RED

En una base de datos en red se permiten relaciones muchos a muchos, no hay una jerarquía. De una entidad a otra se puede llegar por caminos distintos. Este modelo está obsoleto. En la Figura 14.4 se muestra un posible esquema de base de datos en red para el Ejemplo 14.2.



**Figura 14.4.** Ejemplo de base de datos en red.

#### 14.4.3. MODELO RELACIONAL

Las bases de datos relaciones son las más utilizadas hoy en día. Una base de datos relacional está formada por tablas bidimensionales. Las tablas se pueden tratar como relaciones matemáticas, de ahí que se les llame bases de datos relaciones y a las tablas **relaciones**. Las filas se suelen denominar **tuplas** y las columnas **atributos**, ya que una columna describe las características de una entidad de la tupla. En una tabla:

- Todos sus registros son del mismo tipo.
- No aparecen campos repetidos.
- No hay registros duplicados.
- El orden de los registros no importa.
- Hay una llave que puede ser un campo o varios.

En las Tablas 14.1 a 14.4 se muestran tablas con relaciones de flotas, aviones, vuelos y pilotos de una hipotética compañía aérea.

Modelo de avión	Asientos	Autonomía (millas)	Revisión (gran parada) (horas vuelo)
Airbus-340	249	12.700	1.000
Airbus-321	186	4.000	750
Airbus-320/200	150	3.500	2.500
Airbus-319	126	2.200	2.000
Boeing-747	404	13.000	500
Boeing-757	200	3.700	500
MD-88	150	2.900	750
MD-87	109	2.900	1.000
ATR-72	68	1.722	850
CRJ-200	50	3.045	1.250
Fokker-50	50	2.252	750
Dash-8-Q300	52	1.511	650

Tabla 14.1. Relación de FLOTAS.

Matrícula	Flota	N.º de horas de vuelo
EC-BIM	Airbus-340	3.324
EC-CDE	Airbus-340	2.842
EC-HZI	Airbus-321	27.534
EC-BIM	Airbus-321	32.432
EC-CTC	Airbus-321	36.528
EC-HAI	Airbus-320/200	42.520
EC-CIM	Airbus-320/200	43.527
EC-CDF	Airbus-320/200	38.422
EC-HZF	Airbus-319	25.458
EC-BGM	Airbus-319	45.326
EC-CDR	Boeing-747	48.532
EC-RZI	Boeing-757	52.456
EC-BTM	MD-88	62.355
EC-YDE	MD-88	60.123
EC-UZI	MD-87	63.896
EC-BIM	ATR-72	53.432
EC-CKE	CRJ-200	35.532
EC-HZJ	Fokker-50	27.324
EC-FIN	Fokker-50	43.207
EC-CVE	Dash-8-Q300	2.324

Tabla 14.2. Relación de AVIONES.

Código de vuelo	Origen	Destino	Hora despegue (UTC)	Hora aterrizaje (UTC)	Flota
1257	Granada	Barcelona	11:15	12:30	MD-88
0270	Madrid	Granada	16:30	17:35	Airbus-319
8254	Granada	Melilla	16:10	17:00	Fokker-50
9211	Granada	Mallorca	09:25	10:35	Airbus-319
3100	Madrid	Lisboa	05:45	06:55	Airbus-320/200
0575	Vigo	Madrid	12:55	14:00	Airbus-319
1619	Vigo	Barcelona	05:30	07:00	MD-88
0572	Madrid	Vigo	05:25	06:35	Airbus-319
1608	Barcelona	Vigo	15:10	16:45	MD-88
1245	Granada	Madrid	16:30	17:35	MD-88
0277	Barcelona	Granada	09:10	10:30	Airbus-319

Tabla 14.3. Relación de VUELOS.

Id_piloto	Nombre	DNI	Dirección	Cargo	Flota
1733	Valentín	17432133	Prin, 37	Comandante	Airbus-320/200
1534	Sebastián	56146534	Rua, 45	Copiloto	MD-88
3035	Pablo	32303655	Recogidas, 27	Comandante	Airbus-319
1036	Teresa	75120336	Ronda, 43	Comandante	Fokker-50
4832	Enrique	48321832	Toro, 37	Copiloto	MD-88
1584	Alberto	07742979	Gran Vía, 17	Comandante	MD-88
5095	Juan-Mari	50346795	Canalejas, 137	Comandante	Airbus-319
1636	Matías	16322366	Fuentenueva, 23	Copiloto	Airbus-319
4932	María-Jesús	44953421	Ganivet, 42	Comandante	Airbus-320/200
1234	María-Isabel	31427364	Góngora, 17	Copiloto	Airbus-319
3215	Fernando	45273215	Puerta Real, 27	Copiloto	MD-88
8732	Pablo	56348732	Neptuno, 23	Comandante	Airbus-319

**Tabla 14.4.** Relación de PILOTOS.

El **sistema de gestión de bases de datos relaciones (RDBMS)** proporciona las herramientas necesarias para crear bases de datos y manipularlas. Para definir una base de datos relacional hay que especificar cada tabla, y para ello hay que declarar:

- El nombre de la tabla.
- La descripción de los campos de cada registro.
- Los campos que se van a utilizar como llave.

Una vez creada la base de datos existen sentencias (**DML**) para realizar operaciones con sus datos. El **SQL** (*Structured Query Language*) es el lenguaje normalizado por el American National Standards Institute (**ANSI**) y la Organization for Standardization (**ISO**) que se utiliza en las bases de datos relacionales. Es un lenguaje declarativo, y a continuación vamos a describir las sentencias SQL para realizar las operaciones más usuales:

- **INSERT** (insertar): esta sentencia permite insertar un nuevo registro en una tabla:

**Sintaxis:**

```
insert
into NOMBRE_TABLA
values (... , ... , ...)
```

### EJEMPLO 14.3

Un ejemplo de utilización de esta instrucción se tendría si quisiésemos añadir un nuevo vuelo en la relación de VUELOS:

```
insert into VUELOS
values ('0271', 'Granada', 'Madrid', '07:15', '08:10', 'MD-88')
```

Con lo que la Tabla 14.3 se transformaría en la Tabla 14.5.

- **DELETE** (borrar): permite borrar uno o varios registros de una tabla que cumplan una determinada condición.

**Sintaxis:**

```
delete
from NOMBRE_TABLA
where condición
```

Flota	Origen	Destino	Hora despegue (UTC)	Hora aterrizaje (UTC)	Flota
1257	Granada	Barcelona	11:15	12:30	MD-88
0270	Madrid	Granada	16:30	17:35	Airbus-319
8254	Granada	Melilla	16:10	17:00	Fokker-50
9211	Granada	Mallorca	09:25	10:35	Airbus-319
3100	Madrid	Lisboa	05:45	06:55	Airbus-320/200
0575	Vigo	Madrid	12:55	14:00	Airbus-319
1619	Vigo	Barcelona	05:30	07:00	MD-88
0572	Madrid	Vigo	05:25	06:35	Airbus-319
1608	Barcelona	Vigo	15:10	16:45	MD-88
1245	Granada	Madrid	16:30	17:35	MD-88
0277	Barcelona	Granada	09:10	10:30	Airbus-319
<b>0271</b>	<b>Granada</b>	<b>Madrid</b>	<b>07:15</b>	<b>08:10</b>	<b>MD-88</b>

Tabla 14.5. Relación de VUELOS.

**EJEMPLO 14.4**

Un ejemplo de la instrucción DELETE se tendría si quisiésemos dar de baja uno de los aviones, de la relación de AVIONES. Supongamos, por ejemplo, que hubiese que dar de baja al avión de matrícula EC-FIN; esta operación se efectuaría con la instrucción:

```
delete from AVIONES
where Matricula='EC-FIN'
```

La Tabla 14.2 se convertirá en la Tabla 14.6.

Matrícula	Flota	N.º de horas de vuelo
EC-BIM	Airbus-340	3.324
EC-CDE	Airbus-340	2.842
EC-HZI	Airbus-321	27.534
EC-BIM	Airbus-321	32.432
EC-CTC	Airbus-321	36.528
EC-HAI	Airbus-320/200	42.520
EC-CIM	Airbus-320/200	43.527
EC-CDF	Airbus-320/200	38.422
EC-HZF	Airbus-319	25.458
EC-BGM	Airbus-319	45.326
EC-CDR	Boeing-747	48.532
EC-RZI	Boeing-757	52.456
EC-BTM	MD-88	62.355
EC-YDE	MD-88	60.123
EC-UZI	MD-87	63.896
EC-BIM	ATR-72	53.432
EC-CKE	CRJ-200	35.532
EC-HZJ	Fokker-50	27.324
EC-CVE	Dash-8-Q300	2.324

Tabla 14.6. Relación de AVIONES.

- **UPDATE** (actualizar): actualiza registros que cumplan una condición. Las actualizaciones se pueden realizar con instrucciones de la forma:

*Sintaxis:*

```

update NOMBRE_TABLA
set atributo1 = valor1, atributo2 = valor2, ...
where condición

```

**EJEMPLO 14.5**

Supongamos que hay que modificar la hora de salida y llegada del vuelo 1619. Esto lo realizaríamos de la siguiente forma:

```

update VUELOS
set Hora_despegue = '06:00' and Hora_aterrizaje = 07:30
where Id_Vuelo = '1619'

```

Con lo que la Tabla 14.3 se convertiría en la Tabla 14.7.

Código de vuelo	Origen	Destino	Hora despegue (UTC)	Hora aterrizaje (UTC)	Flota
1257	Granada	Barcelona	11:15	12:30	MD-88
0270	Madrid	Granada	16:30	17:35	Airbus-319
8254	Granada	Melilla	16:10	17:00	Fokker-50
9211	Granada	Mallorca	09:25	10:35	Airbus-319
3100	Madrid	Lisboa	05:45	06:55	Airbus-320/200
0575	Vigo	Madrid	12:55	14:00	Airbus-319
1619	Vigo	Barcelona	<b>06:00</b>	<b>07:30</b>	MD-88
0572	Madrid	Vigo	05:25	06:35	Airbus-319
1608	Barcelona	Vigo	15:10	16:45	MD-88
1245	Granada	Madrid	16:30	17:35	MD-88
0277	Barcelona	Granada	09:10	10:30	Airbus-319
0271	Granada	Madrid	07:15	08:10	MD-88

**Tabla 14.7.** Relación de VUELOS.

- **SELECT** (seleccionar): selecciona registros que cumplan unas condiciones. Una selección se pueden realizar con instrucciones de la forma:

*Sintaxis:*

```

select [*|campo]
from TABLA
where condición

```

(\* significa selección de todos los campos).

**EJEMPLO 14.6**

Un ejemplo de selección consiste en obtener la relación de VUELOS aquellos que tienen su origen en Granada y destino en Madrid, lo que se hace con las siguientes sentencias:

```

select *
from VUELOS
where Origen='Granada' and Destino='Madrid'

```

El resultado sería el siguiente:

Código de vuelo	Origen	Destino	Hora despegue (UTC)	Hora aterrizaje (UTC)	Flota
1245	Granada	Madrid	16:30	17:35	MD-88
0271	Granada	Madrid	07:15	08:10	MD-88

- **PROJECT** (proyectar): crea una tabla a partir de ciertos campos de una relación existente. Con una proyección la tabla resultante tiene las mismas filas pero menos columnas. Una proyección se puede hacer con una sentencia como la que sigue:

*Sintaxis:*

```
select lista_de_campos
from NOMBRE_TABLA
```

### EJEMPLO 14.7

Si deseamos extraer de la relación de PILOTOS (Tabla 14.4) los nombres y direcciones, debemos realizar una proyección de la siguiente forma:

```
select Nombre, Dirección
from PILOTOS
```

El resultado sería el que se muestra en la Tabla 14.8.

Nombre	Dirección
Valentín	Prin, 37
Sebastián	Rua, 45
Pablo	Recogidas, 27
Teresa	Ronda, 43
Enrique	Toro, 37
Alberto	Gran Vía, 17
Juan-Mari	Canalejas, 137
Matías	Fuentenueva, 23
María-Jesús	Ganivet, 42
María-Isabel	Góngora, 17
Fernando	Puerta Real, 27
Pablo	Neptuno, 23

**Tabla 14.8.** Relación de PILOTOS\_DIRECCIONES.

- **JOIN** (juntar): crea una tabla nueva uniendo dos tablas existentes a partir de un campo común.

*Sintaxis:*

```
select lista-atributos
from TABLA1,TABLA2
where criterios
```

### EJEMPLO 14.8

Un ejemplo de la instrucción JOIN se presenta si queremos obtener una relación que muestre el número de asientos disponibles en cada vuelo. Supongamos que deseamos obtener una relación con los siguientes atributos: código de vuelo, origen, destino y número de asientos. Tendremos que juntar información de las tablas de VUELOS y de FLOTA, utilizando como atributo común el campo flota. Las sentencias para lograr este objetivo son las siguientes:

```
select código_vuelo, origen, destino, asientos
from VUELOS, FLOTAS
where flota
```

El resultado de esta operación se muestra en la Tabla 14.9.

Código de vuelo	Origen	Destino	Asientos
1257	Granada	Barcelona	150
0270	Madrid	Granada	126
8254	Granada	Melilla	50
9211	Granada	Mallorca	126
3100	Madrid	Lisboa	150
0575	Vigo	Madrid	126
1619	Vigo	Barcelona	150
0572	Madrid	Vigo	126
1608	Barcelona	Vigo	150
1245	Granada	Madrid	150
0277	Barcelona	Granada	126
0271	Granada	Madrid	150

**Tabla 14.9.** Relación de VUELOS\_PLAZAS.

- **UNION** (unión): crea una tabla nueva uniendo dos tablas existentes, de forma que cada registro de la nueva tabla pertenece a una de las dos tablas o a las dos. Las dos tablas deben tener los mismos atributos. La forma de realizar una unión es utilizando las sentencias que se muestran a continuación:

**Sintaxis:**

```
select [*|campo]
from TABLA1
[where condición]
union
select [*|campo]
from TABLA2
[where condición]
```

(Recuérdese que [ ...] significa instrucción opcional)

**EJEMPLO 14.9**

Supongamos que queremos unir las relaciones indicadas en las Tablas 14.10 y 14.11. Este objetivo se conseguiría con las siguientes sentencias:

```
select *
from AVIONES_IBERIA
union
select *
from AVIONES_AIREEUROPA
```

Id_Avión	Modelo	Asientos	Autonomía (km)	Compañía
A1	Airbus-340	249	12.700	Iberia
A2	Airbus-321	186	4.000	Iberia
A3	Airbus-320/200	150	3.500	Iberia
A4	Airbus-319	126	2.200	Iberia
A5	Boeing-747	404	13.000	Iberia
A6	Boeing-757	200	3.700	Iberia
A7	MD-88	150	2.900	Iberia

**Tabla 14.10.** Relación AVIONES\_IBERIA.

Id_Avión	Modelo	Asientos	Autonomía (km)	Compañía
A13	Boeing-737/300	142	3.700	AirEuropa
A14	Boeing-737/400	162	3.140	AirEuropa
A15	Boeing-737/600	116	2.950	AirEuropa
A16	Boeing-737/800	186	5.000	AirEuropa
A17	Boeing-767/300	263	9.260	AirEuropa

Tabla 14.11. Relación AVIONES\_AIREUROPA.

El resultado sería la relación que se muestra en la Tabla 14.12.

Id_Avión	Modelo	Asientos	Autonomía (km)	Compañía
A1	Airbus-340	249	12.700	Iberia
A2	Airbus-321	186	4.000	Iberia
A3	Airbus-320/200	150	3.500	Iberia
A4	Airbus-319	126	2.200	Iberia
A5	Boeing-747	404	13.000	Iberia
A6	Boeing-757	200	3.700	Iberia
A7	MD-88	150	2.900	Iberia
A13	Boeing-737/300	142	3.700	AirEuropa
A14	Boeing-737/400	162	3.140	AirEuropa
A15	Boeing-737/600	116	2.950	AirEuropa
A16	Boeing-737/800	186	5.000	AirEuropa
A17	Boeing-767/300	263	9.260	AirEuropa

Tabla 14.12. Relación AVIONES.

- **INTERSECTION** (intersección): crea una nueva tabla en la que cada registro está en las dos tablas.

*Sintaxis:*

```
select [*|campo]
from tabla1
[where condición]
intersection
select [*|campo]
from tabla2
[where condición]
```

#### EJEMPLO 14.10

Supongamos que deseamos conocer los vuelos cuyos orígenes y destinos se repiten por la mañana y por la tarde y con el mismo tipo de avión, partiendo de las relaciones de VUELOS\_MÑANA (Tabla 14.13) y VUELOS\_TARDE (Tabla 14.14).

Código de vuelo	Origen	Destino	Hora despegue (UTC)	Hora aterrizaje (UTC)	Flota
0572	Madrid	Vigo	05:25	06:35	Airbus-319
3100	Madrid	Granada	05:45	06:55	Airbus-319
1619	Vigo	Barcelona	06:00	07:30	MD-88
0271	Granada	Madrid	07:15	08:10	MD-88
0277	Barcelona	Granada	09:10	10:30	Airbus-319
9211	Granada	Mallorca	09:25	10:35	Airbus-319
1257	Granada	Barcelona	11:15	12:30	MD-88

Tabla 14.13. Relación de VUELOS\_MÑANA.



Código de vuelo	Origen	Destino	Hora despegue (UTC)	Hora aterrizaje (UTC)	Flota
0575	Vigo	Madrid	12:55	14:00	Airbus-319
1608	Madrid	Vigo	15:10	16:45	Airbus-319
8254	Granada	Melilla	16:10	17:00	Fokker-50
0270	Madrid	Granada	16:30	17:35	Airbus-319
1245	Granada	Madrid	16:30	17:35	MD-88

**Tabla 14.14.** Relación de VUELOS\_TARDE.

Las sentencias a utilizar serían las siguientes:

**Sintaxis:**

```
select origen, destino, flota
from VUELOS_MAÑANA
intersection
select origen, destino, flota
from VUELOS_TARDE
```

El resultado sería el que se muestra en la Tabla 14.15.

Origen	Destino	Flota
Madrid	Vigo	Airbus-319
Granada	Madrid	MD-88

**Tabla 14.15.** VUELOS\_INTERSECCIÓN.

- **DIFERENCE** (diferencia): crea una nueva tabla que contiene los registros de la primera tabla que no están en la segunda.

**Sintaxis:**

```
select [*|campo]
from tabla1
[where condición]
except
select [*|campo] from tabla2
[where condición]
```

### EJEMPLO 14.11

Supongamos que deseamos conocer los vuelos de la tarde cuyos orígenes y destinos no se repiten por la mañana, y con el mismo tipo de avión, partiendo de las relaciones de VUELOS\_MAÑANA (Tabla 14.13) y VUELOS\_TARDE (Tabla 14.14).

**Sintaxis:**

```
select origen, destino, flota
from VUELOS_TARDE
except
select origen, destino, flota
from VUELOS_MAÑANA
```

La relación que se obtendría es la que se muestra en la Tabla 14.16.

Código de vuelo	Origen	Destino	Hora despegue (UTC)	Hora aterrizaje (UTC)	Flota
0575	Vigo	Madrid	12:55	14:00	Airbus-319
8254	Granada	Melilla	16:10	17:00	Fokker-50
0270	Madrid	Granada	16:30	17:35	Airbus-319

**Tabla 14.16.** Relación de VUELOS\_DIFERENCIA.

El lenguaje SQL también permite combinar las sentencias para manipular información más concreta (ver, por ejemplo, Problema 14.10).

## 14.5. Tipos de bases de datos

En la sección anterior (Sección 14.4) hemos estudiado los modelos básicos de bases de datos; ahora bien, estos modelos se pueden implementar de distintas formas. Una base de datos sencilla suele ubicarse físicamente en un computador central o en un servidor de archivos, accediendo los usuarios a ella a través de terminales conectados en una red local o de otro tipo e incluso a través de Internet. Estas bases de datos se denominan centralizadas y como alternativa a ellas se implementan bases de datos distribuidas, a las que dedicaremos la Sección 14.5.1.

Por otra parte, en la Sección 12.5.1 se analizó brevemente el concepto de programación orientada a objetos, y las ventajas que ésta ofrecía para el desarrollo de aplicaciones. Muchas de las técnicas de este tipo de programación son aplicables a las bases de datos, existiendo bases de datos orientadas a objetos que son objeto de estudio en la Sección 14.5.2.

### 14.5.1. BASES DE DATOS DISTRIBUIDAS

Las bases de datos distribuidas están basadas en el modelo relacional. Los datos están almacenados en distintos computadores conectados a través de una red (Internet o red de área local, por ejemplo). Cada computador se encarga de mantener toda o parte de la base de datos en ella almacenada. En las **bases de datos distribuidas con fragmentación**, en cada computador se almacenan datos distintos, pero todos tienen acceso a todos los datos. En las **bases de datos distribuidas con duplicación** hay una copia de todos los datos en cada computador, de forma que cuando algo se actualiza se modifica en todas las copias y cuando se produce algún fallo en algún equipo se pueden recuperar los datos desde cualquier otro.

### 14.5.2. BASES DE DATOS ORIENTADAS A OBJETOS

Las bases de datos orientadas a objetos se han creado para satisfacer las necesidades de algunas aplicaciones nuevas que no se pueden llevar a cabo con bases de datos tradicionales, y también pensando en su integración directa con aplicaciones desarrolladas con lenguajes orientados a objetos (Sección 12.5.1).

En este tipo de bases de datos se definen objetos y relaciones entre ellos, permitiéndose acceder a datos estructurados. Un objeto está formado por entidades que tienen una serie de características, atributos y procedimientos asociados. Un objeto puede ser cualquier cosa: un artículo, un evento, una compra, etc.; sus características o atributos pueden ser texto, sonido, gráficos, etc., y un procedimiento es cualquier procesamiento relacionado con el objeto.

En el caso de la compañía aérea, se pueden definir cinco tipos de objetos o clases: Avión, Flota, Piloto, Azafata y Aeropuerto. Un objeto de la clase Avión puede contener los siguientes elementos: matrícula, flota y número de horas de vuelo. Otra clase que se define es Asignación, que representa la asignación o relación que un objeto de una clase tiene con otro objeto de otra clase; así existirá un objeto de la clase Asignación que asocia Avión con Piloto (los aviones que puede conducir cada piloto). Por lo general, en las bases de datos orientadas a objetos los enlaces (relaciones) entre objetos son mantenidos por el DBMS a través de punteros.

Como en los lenguajes orientados a objetos, aquí también los objetos están encapsulados, comunicándose unos con otros a través de mensajes que se intercambian a través de las interfaces de los objetos. Debido al sistema de comunicación entre objetos se tiene una imagen homogénea de todos ellos, independientemente de su naturaleza (sonido, vídeo, texto, etc.). Hay objetos que pueden mostrar a través de su interfaz los datos interiores en formatos diversos o desde distintas perspectivas en función de su contenido y relaciones. A este tipo de entidades se les suele denominar **objetos inteligentes**. En un objeto inteligente, además de los datos, se almacenan los procedimientos que describen como suministrar los datos a través de la interfaz de salida, en función del mensaje recibido.

## 14.6. Conclusiones

En este capítulo hemos presentado los conceptos básicos de bases de datos. Las bases de datos surgen con la idea de generar sistemas de almacenamiento de los datos de una empresa o una entidad de la forma más eficiente posible, y buscando el acceso fácil y controlado a dichos datos. Una vez diseñada una base de datos, las distintas aplicaciones acceden a ella para obtener los datos necesarios para lograr los resultados buscados. Entre las cualidades de las bases de datos se encuentran: acceso múltiple por diversos usuarios o aplicaciones sin que se produzcan conflictos, utilización múltiple de forma tal que cada usuario o aplicación sólo accede a los datos que necesita y está autorizado, distintos métodos de acceso con tiempos de respuesta razonables, confidencialidad y seguridad, métodos de protección claramente definidos, redundancia controlada, interfaz cómoda para el acceso de usuarios y aplicaciones, posibilidad de acceso directo a los datos, etc.

En la Sección 14.1 se ha definido el concepto de base de datos y se han descrito las características que se buscan con este tipo de sistemas. Posteriormente, en la Sección 14.2, se han presentado las definiciones básicas para entender adecuadamente la estructura y forma de funcionamiento de bases de datos. La Sección 14.3 describe distintos niveles o capas que intervienen en un sistema de gestión de base de datos: usuario (aplicaciones), lógico (Sistema de Gestión de Base de Datos o DBMS) y físico (soporte de almacenamiento controlado por las rutinas de bajo nivel integradas en el sistema operativo).

Las Secciones 14.4 y 14.5 se dedican a presentar distintos modelos y tipos de bases de datos, concretamente los que se relacionan en la Tabla 14.17.

<b>Modelos:</b> <ul style="list-style-type: none"><li>• Modelo jerárquico.</li><li>• Modelo en red.</li><li>• Modelo relacional.</li></ul>	<b>Ubicación física:</b> <ul style="list-style-type: none"><li>• Base de datos centralizada.</li><li>• Base de datos distribuidas con fragmentación.</li><li>• Base de datos distribuidas con duplicidad.</li></ul> <b>Orientación:</b> <ul style="list-style-type: none"><li>• Base de datos clásica.</li><li>• Base de datos orientada a objetos.</li></ul>
--	---

Tabla 14.17. Modelos y tipos de bases de datos.

## Test



**T14.1.** En Informática, un conjunto de información sobre un mismo tema tratado como una unidad de almacenamiento y organizada de forma estructurada, constituye:

- a) Una base de datos.
- b) Un sistema de gestión de archivos.
- c) Un bloque físico de información.
- d) Un archivo.

**T14.2.** Una base de datos está formada por:

- a) Un conjunto de archivos.
- b) Conjuntos de datos de distinta naturaleza, pero relacionados unos con otros, y el software necesario para su gestión.
- c) Un sistema completo (programa + dispositivos de memoria masiva) para gestionar datos.
- d) El conjunto de archivos de una empresa o institución.

**T14.3.** Una base de datos se dice que cumple el requisito de acceso múltiple si:

- a) Diversos usuarios pueden acceder a la base de datos, sin visiones incoherentes y sin conflictos.
- b) Cada usuario puede tener una imagen o visión particular de la estructura de la base de datos.
- c) Si se pueden usar múltiples métodos de acceso con tiempos de respuesta razonables.
- d) Si se controla el acceso a los datos de múltiples usuarios, impidiéndoselo a los no autorizados, y con distintos niveles de privilegio a distintos tipos de datos.

**T14.4.** Una base de datos se dice que cumple el requisito de utilización múltiple si:

- a) Cada usuario puede tener una imagen o visión particular de la estructura de la base de datos.
- b) Si se pueden usar múltiples métodos de acceso con tiempos de respuesta razonables.
- c) Si se controla el acceso a los datos de múltiples usuarios, impidiéndoselo a los no autorizados, y con distintos niveles de privilegio a distintos tipos de datos.
- d) Diversos usuarios pueden acceder a la base de datos, sin visiones incoherentes y sin conflictos.

**T14.5.** Una base de datos se dice que cumple el requisito de flexibilidad si:

- a) Cada usuario puede tener una imagen o visión particular de la estructura de la base de datos.
- b) Si se pueden usar múltiples métodos de acceso con tiempos de respuesta razonables.
- c) Si se controla el acceso a los datos de múltiples usuarios, impidiéndoselo a los no autorizados, y con distintos niveles de privilegio a distintos tipos de datos.
- d) Diversos usuarios pueden acceder a la base de datos, sin visiones incoherentes y sin conflictos.

**T14.6.** Una base de datos se dice que cumple el requisito de confidencialidad y seguridad si:

- a) Cada usuario puede tener una imagen o visión particular de la estructura de la base de datos.
- b) Si se pueden usar múltiples métodos de acceso con tiempos de respuesta razonables.
- c) Si se controla el acceso a los datos de múltiples usuarios, impidiéndoselo a los no autorizados, y con distintos niveles de privilegio a distintos tipos de datos.
- d) Diversos usuarios pueden acceder a la base de datos, sin visiones incoherentes y sin conflictos.

**T14.7.** Una base de datos es:

- a) Un programa para gestionar archivos muy grandes.
- b) El conjunto de datos de los usuarios almacenados en un único disco duro.
- c) Un archivo que excede la capacidad de un disco duro (necesita más de uno para almacenarse).
- d) Conjunto de datos de distinto tipo relacionados entre sí, junto con un programa de gestión de dichos datos.

**T14.8.** Los objetos o elementos que se almacenan en una base de datos se denominan:

- a) Entidades.
- b) Atributos.
- c) Registros.
- d) Campos.

**T14.9.** Los objetos o elementos de una base de datos se almacenan en:

- a) Entidades.
- b) Atributos.
- c) Registros.
- d) Campos.

**T14.10.** Las propiedades o datos de los objetos que se almacenan en una base de datos se denominan:

- a) Entidades.
- b) Atributos.
- c) Registros.
- d) Campos.

**T14.11.** Las propiedades de los objetos de en una base de datos se almacenan en:

- a) Entidades.
- b) Atributos.
- c) Registros.
- d) Campos.

**T14.12.** Un DBMS es un conjunto de programas para:

- a) La creación de la información de una base de datos.
- b) El control de la información de una base de datos.

- c) Acceso e interrogación de la información de una base de datos.
- d) Tanto la creación, como el control y manipulación de la información de una base de datos.

**T14.13.** El DDL de una base de datos sirve para:

- a) Definir la estructura lógica de la base de datos.
- b) La introducción de los datos en una base de datos.
- c) Interrogar a la base de datos (consultar la información de dicha base).
- d) Tanto la creación, como el control y manipulación de la información de una base de datos.

**T14.14.** El DML de una base de datos sirve para:

- a) Definir la estructura lógica de la base de datos.
- b) El control de la información de una base de datos.
- c) La introducción de los datos en una base de datos.
- d) Acceder a la base de datos para leer, escribir o modificar información.

**T14.15.** El modelo de base de datos menos utilizado en la actualidad es el:

- a) Jerárquico.
- b) En red.
- c) Relacional.
- d) Distribuido.

**T14.16.** El modelo de base de datos en el que no se pueden definir relaciones muchos a muchos es el:

- a) Jerárquico.
- b) En red.
- c) Relacional.
- d) Distribuido.

**T14.17.** El modelo de base de datos en el que se establecen las relaciones entre los datos con una estructura de árbol descendente es el:

- a) Jerárquico.
- b) En red.
- c) Relacional.
- d) Distribuida.

**T14.18.** El modelo de base de datos en el que los datos se encuentran en varios computadores conectados en red es el:

- a) Jerárquico.
- b) En red.
- c) Relacional.
- d) Distribuido.

**T14.19.** En una base de datos relacional, las relaciones se representan lógicamente por medio de tablas de:

- a) Una dimensión.
- b) Dos dimensiones.
- c) Tres dimensiones.
- d) Cualquier dimensión.

**T14.20.** En una tabla en una base de datos relacional, el número de atributos (columnas o campos) como máximo puede ser:

- a) Uno.
- b) Dos.
- c) Tres.
- d) Cualquiera.

**T14.21.** Si se desea cambiar el valor de un atributo (campo) en un registro (tupla) debe usarse una operación de:

- a) Proyección (project).
- b) Composición (union, join).
- c) Actualización (update).
- d) Selección (select).

**T14.22.** Si se tienen una tabla con información de los vuelos de una determinada compañía aérea, y se desea extraer sólo los que se hacen con Jumbos (Boeing 747), debe usarse una operación de:

- a) Proyección (project).
- b) Composición (union, join).
- c) Actualización (update).
- d) Selección (select).

**T14.23.** Si se desea obtener una nueva relación a partir de una ya existente, debe usarse una operación de:

- a) Proyección (project).
- b) Composición (union, join).
- c) Actualización (update).
- d) Selección (select).

**T14.24.** Si se desea obtener una nueva relación a partir de dos ya existentes, debe usarse una operación de:

- a) Proyección (project).
- b) Composición (union, join).
- c) Actualización (update).
- d) Selección (select).

**T14.25.** Si se desea añadir un registro (tupla) nuevo en una relación, debe usarse una operación de:

- a) Inserción (insert).
- b) Eliminación (delete).
- c) Intersección (intersection).
- d) Diferencia (except).

**T14.26.** Si se desea obtener una nueva relación que contenga los registros (tuplas) comunes a dos relaciones, debe usarse una operación de:

- a) Inserción (insert).
- b) Eliminación (delete).
- c) Intersección (intersection).
- d) Diferencia (except).

**T14.27.** Si se desea obtener una nueva relación que contenga los registros (tuplas) de la primera relación que no estén en la segunda, debe usarse una operación de:

- a) Inserción (insert).
- b) Eliminación (delete).
- c) Intersección (intersection).
- d) Diferencia (except).

**T14.28.** Si se desea eliminar un registro (tupla) existente en una relación, debe usarse una operación de:

- a) Inserción (insert).
- b) Eliminación (delete).
- c) Intersección (intersection).
- d) Diferencia (except).

**T14.29.** Una base de datos en la que se mantienen copias de los mismos datos en distintos computadores interconectados en red se dice que:

- a) Utiliza un modelo jerárquica.

- b) Utiliza un modelo en red.
- c) Es de tipo distribuido con duplicidad.
- d) Es de tipo orientado a objetos.

**T14.30.** Una base de datos almacenada entre distintos computadores conectados en red, de forma que unos tienen acceso a los datos de otros, se dice que:

- a) Utiliza un modelo jerárquica.
- b) Utiliza un modelo en red.
- c) Es de tipo distribuido con fragmentación.
- d) Es de tipo distribuida con duplicidad.

## Problemas resueltos



**P14.1.** Dadas las relaciones de la Tabla 14.18, que pertenecen a una base de datos relacional, escribir la sentencia SQL necesaria para mostrar sólo las frutas de la tabla Vegetales\_Verano.

Relación TIPO_VEGETAL	
Id_tipo	Tipo
T1	Frutas
T2	Hortalizas

Relación VEGETALES_VERANO			
Id_vegetal	Id_tipo	Vegetal	Precio
VV1	T1	Ciruela	2
VV2	T1	Albaricoque	2,5
VV3	T2	Tomate	0,9
VV4	T2	Berenjena	1
VV5	T2	Pepino	0,50
V6	T2	Patata	0,3

Relación VEGETALES_INVIERNO			
Id_vegetal	Id_tipo	Vegetal	Precio
VI1	T1	Naranja	0,9
VI2	T1	Manzana	1,2
VI3	T2	Cardo	1,3
VI4	T2	Puerro	0,6
V6	T2	Patata	0,3
VI5	T2	Brócoli	1,5

**Tabla 14.18.** Relaciones en una base de datos relacional sobre vegetales comestibles.

### SOLUCIÓN

```
select * from VEGETALES_VERANO
where Id_tipo = 'T1'
```

**P14.2.** Para las relaciones de las tablas de la Tabla 14.18, escribir la sentencia SQL necesaria para actualizar el precio de las manzanas de 1,2 € a 1,4 €.

### SOLUCIÓN

```
update VEGETALES_INVIERNO
set Precio='1,4'
where Id_vegetal=VI2'
```

- P14.3.** Dadas las relaciones de la Tabla 14.18, escribir la sentencia SQL necesaria para añadir fresas a 1,5 € en la tabla VEGETALES\_VERANO.

SOLUCIÓN

```
insert
into VEGETALES_VERANO ('VV7', 'T1', 'Fresas', '1,5')
```

- P14.4.** Dadas las relaciones de la Tabla 14.18, escribir la sentencia SQL necesaria para crear una nueva tabla que contenga sólo el nombre y el precio de las frutas de verano.

SOLUCIÓN

```
select Nombre, Precio
from VEGETALES_VERANO
where Id_tipo='T1'
```

- P14.5.** Dadas las relaciones de la tabla 14.18, escribir la sentencia SQL necesaria para crear una nueva tabla los nombres de los tipos de vegetales y los vegetales de verano.

SOLUCIÓN

```
select *
from TIPO_VEGETAL left join VEGATALES_VERANO
on Tipo_VEGETAL.Id_tipo=VEGETALES_VERANO.Id_tipo
```

- P14.6.** Dadas las relaciones de la Tabla 14.18, ¿qué resultado generarían las siguientes sentencias SQL?

- a) **Select \* from VEGETALES\_VERANO union select \* from VEGETALES\_IN-  
VIERNO**  
b) **Select \* from VEGETALES\_VERANO intersection select \* from VEGETA-  
LES\_INVIERNO**  
c) **Select \* from VEGETALES\_VERANO except select \* from VEGETALES\_IN-  
VIERNO**

SOLUCIÓN

- a) **select \* from VEGETALES\_VERANO union select \* from VEGETALES\_IN-  
VIERNO**

Con esta sentencia se obtendría la Tabla 14.19.

Id_vegetal	ID_tipo	Nombre	Precio
VV1	T1	Ciruela	2
VV2	T1	Albaricoque	2,5
VV3	T2	Tomate	0,9
VV4	T2	Berenjena	1
VV5	T2	Pepino	0,50
V6	T2	Patata	0,3
VI1	T1	Naranja	0,9
VI2	T1	Manzana	1,2
VI3	T2	Cardo	1,3
VI4	T2	Puerro	0,6
VI5	T2	Brócoli	1,5

**Tabla 14.19.** Relación VEGETALES.

- b) **select \* from VEGETALES\_VERANO intersection select \* from VEGETA-  
LES\_INVIERNO**

Con esta sentencia se obtendría la Tabla 14.20.

Id_vegetal	ID_tipo	Nombre	Precio
V6	T2	Patata	0,3

Tabla 14.20. Relación Vegetales\_todo\_año.

c) `select * from VEGETALES_VERANO except select * from VEGETALES_IN-  
VIERNO`

Con esta sentencia se obtendría la Tabla 14.21.

Id_vegetal	ID_tipo	Nombre	Precio
VV1	T1	Ciruela	2
VV2	T1	Albaricoque	2,5
VV3	T2	Tomate	0,9
VV4	T2	Berenjena	1
VV5	T2	Pepino	0,50

Tabla 14.21. Vegetales\_verano\_y\_no\_invierno.

**P14.7.** Indicar las sentencias que habría que aplicar a las relaciones de la Tabla 14.18 para obtener la relación de la Tabla 14.22. ¿Qué redundancias se han introducido al juntar las dos tablas?

Id_vegetal	Tipo	Nombre
VV1	Fruta	Ciruela
VV2	Fruta	Albaricoque
VV3	Hortaliza	Tomate
VV4	Hortaliza	Berenjena
VV5	Hortaliza	Pepino
V6	Hortaliza	Patata
VI1	Fruta	Naranja
VI2	Fruta	Manzana
VI3	Hortaliza	Cardo
VI4	Hortaliza	Puerro
VI5	Hortaliza	Brócoli

Tabla 14.22.

## SOLUCIÓN

```
select Id-vegetal, tipo, nombre
from VEGETAL, TIPO_VEGETAL
where VEGETAL.Id_tipo=TIPO_VEGETAL.Id_tipo
```

**P14.8.** Indicar las sentencias que habría que aplicar a las relaciones de la Tabla 14.19 para obtener el precio de los tomates.

## SOLUCIÓN

```
select Precio
from VEGETALES
where VEGETALES.nombre = "tomate"
```

**P14.9.** Considerar las entidades y atributos que se indican en el Ejemplo 14.2, añadiendo dos nuevas entidades: azafatas y aeropuertos. Hacer un diagrama que muestre el esquema de relaciones de la base de datos.



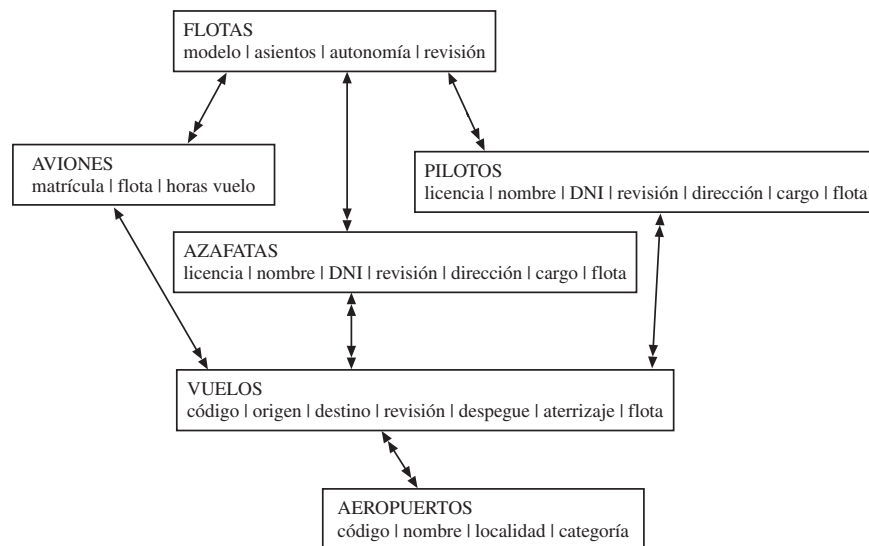
## SOLUCIÓN

Las entidades de la base de datos son:

- **Flotas de aviones:** Modelo de avión, número asientos, autonomía, revisión.
- **Aviones:** matrícula (identifica unívocamente al avión), flota (modelo de avión), número de horas de vuelo que lleva realizadas el avión.
- **Vuelos:** Código de vuelo, origen, destino, hora despegue, hora aterrizaje, flota.
- **Pilotos:** Número licencia, Nombre, DNI, Dirección, Cargo, Flota.
- **Azafatas:** Número licencia, Nombre, DNI, Dirección, Cargo, Flota.
- **Aeropuertos:** Código, Nombre, Localidad, Categoría.

La categoría del aeropuerto está en función de parámetros tales como número de pistas, longitud de pista, ayudas a la navegación, horario de apertura, etc.

Un posible esquema es el que se muestra en la Figura 14.5.



**Figura 14.5.** Esquema de una base de datos para una compañía aérea.

- P14.10.** Obtener las sentencias en SQL necesarias para obtener de la relación de pilotos (Tabla 14.4) dos relaciones distintas: una de comandantes y otra de copilotos.

## SOLUCIÓN

```

select *
from PILOTOS
where PILOTOS.CARGO = "Comandante"
select *
from PILOTOS
where PILOTOS.CARGO = "Copilotos"

```

- P14.11.** Indicar cómo reducir la capacidad que ocupa la relación de PILOTOS (Tabla 14.5), eliminando redundancias.

## SOLUCIÓN

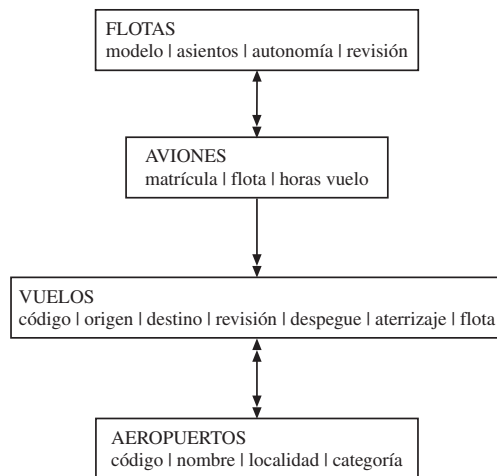
Pueden reducirse la capacidad de la tabla:

1. Codificando los cargos (por ejemplo: 1, Comandante; 2, Copiloto) y creando una tabla de CARGOS, indicando el código del cargo y el nombre del cargo.
2. Codificando los tipos de avión de las distintas flotas. Para ello en la relación de FLOTAS deberíamos añadir un atributo de código de flota, y sustituir en la relación de PILOTOS el nombre de la flota por su código. Esta sustitución también debería de hacerse en la relación de VUELOS.

**P14.12.** Establecer un subesquema de la base de datos de la compañía aérea con objeto de asignar aviones a los vuelos teniendo en cuenta las categorías de los aeropuertos.

#### SOLUCIÓN

Para realizar la asignación requerida sólo son necesarias las entidades FLOTAS, AVIONES, VUELOS y AEROPUERTOS. En consecuencia, el subesquema de la base de datos será el que se muestra en la Figura 14.6.



**Figura 14.6.** Subesquema para asignar aviones a vuelos de la base de datos para una compañía aérea.

**P14.13.** Para el problema anterior, indicar el tipo de relaciones existentes (uno a uno, uno a muchos o muchos a muchos).

#### SOLUCIÓN

- **FLOTAS** con **AVIONES**. Cada avión pertenece a una única flota, pero cada flota tiene múltiples aviones; luego la relación es uno a muchos.
- **AVIONES** con **VUELOS**. Cada vuelo se realiza con un único avión, pero un mismo avión realiza múltiples vuelos; luego es una relación uno a muchos.
- **VUELOS** con **AEROPUERTOS**. Cada vuelo (“salto”) está relacionado con dos aeropuertos (origen y destino) y a cada aeropuerto llegan múltiples aviones; luego es una relación dos a muchos.

## Problemas propuestos



**P14.14.** Determinar el esquema de una **base de datos académica** con las entidades y los atributos siguientes:

- Departamento: Id, nombre, dirección
- Asignaturas: Id, nombre, Id\_departamento, DNI\_profesor
- Profesores: DNI, nombre, departamento
- Alumnos: DNI, nombre, domicilio
- Matrícula alumno: DNI, asignatura
- Expediente alumno: DNI, Id\_asignatura, año, calificación

**P14.15.** Determinar los tipos de relaciones (uno a uno, uno a muchos, o muchos a muchos) existentes en la base de datos académica (Problema 14.14).

**P14.16.** Realizar el subesquema de la base de datos académica (Problema 14.14) que debería establecerse para acceso de los alumnos, suponiendo que únicamente deben poder consultar sus datos personales, su matrícula y su expediente.

**P14.17.** Suponga que la base de datos académica (Problema 14.14) se implementa como base de datos relacional. Ima-

gine que se dispone de las relaciones que se muestran en las Tablas 14.23 a 14.27. Indicar las sentencias SQL necesarias para obtener las asignaturas que imparte el profesor Delgado.

**P14.18.** A partir de las relaciones que se muestran en las Tablas 14.23 a 14.27, indicar las sentencias SQL necesarias para insertar al siguiente nuevo alumno:

DNI	Nombre	Domicilio
53275832	Gómez	Prin, 48
47577747	Martín	Gran Vía, 32
53328975	Pino	Sol, 23
22428734	Anguita	Luna, 48
15487632	Rojas	Toro, 27
44355678	Cañas	Recogidas, 32
34322575	Pelayo	Mirat, 13
45628734	Merelo	Pino, 32
44487632	Ramírez	Murcia, 48
47868678	Olivares	Góngora, 6

**Tabla 14.23.** ALUMNOS.

DNI_alumno	Id_Asignatura
53275832	IS
53275832	EC
53275832	TC
53275832	LP
47577747	SO
47577747	IS
22428734	IS
22428734	VC
22428734	AC
47868678	IS

**Tabla 14.24.** MATRÍCULAS.

Id.	Nombre	Dep.	DNI profesor
IS	Ingeniería del Software	LI	25487632
EC	Estructura de Computadores	AC	47652347
TC	Tecnología de Computadores	AC	23428734
LP	Lenguajes de Programación	LI	54328975
SO	Sistemas Operativos	LI	25487632
AC	Arquitectura de Computadores	AC	47652347
SE	Sistemas Expertos	IA	43275832
VC	Visión por Computador	IA	45345678
MI	Modelos de la I.A.	IA	43275832

**Tabla 14.25.** ASIGNATURAS.

DNI	Nombre	Dep.
43275832	Delgado	IA
47652347	Ortega	AC
54328975	Clares	LI
23428734	Prieto	AC
25487632	Torres	LI
45345678	Molina	IA

**Tabla 14.26.** PROFESORES.

Id.	Nombre	Domicilio
AC	Arquitectura Computadores	ETSII, planta 2
IA	Inteligencia Artificial	ETSII, planta 3
LI	Lenguajes Infomáticos	ETSII, planta 4

**Tabla 14.27.** Departamento.

DNI: 46342725Y Nombre: Pérez Domicilio: Avda. España, 3, que se *matricula* en las asignaturas IS, EC y TP.

**P14.19.** A partir de las relaciones que se muestran en las Tablas 14.23 a 14.27, indicar las sentencias SQL necesarias para modificar la adscripción de la asignatura Sistemas Operativos asignándosela al Departamento ATC, profesor Prieto.

**P14.20.** A partir de las relaciones que se muestran en las Tablas 14.23 a 14.27, indicar las sentencias SQL necesarias para obtener la relación de asignaturas que imparte el Departamento LSI.

**P14.21.** A partir de las relaciones que se muestran en las Tablas 14.23 a 14.27, indicar las sentencias SQL necesarias para

obtener un lista de alumnos conteniendo sólo el DNI y el nombre.

**P14.22.** A partir de las relaciones que se muestran en las Tablas 14.23 a 14.27, indicar las sentencias SQL necesarias para obtener una lista de los nombres de las asignaturas y de los profesores que las imparten.

**P14.23.** A partir de las relaciones que se muestran en las Tablas 14.23 a 14.27, indicar las sentencias SQL necesarias para obtener una lista de los nombres de los alumnos de la asignatura Ingeniería del Software.

**P14.24.** Suponga que en el centro docente se imparten dos titulaciones distintas: Ingeniero Técnico de Informática de Sis-

temas (ITIS) e Ingeniero Técnico de Informática de Gestión (ITIG). En la Tabla 14.25 se mostraron las relaciones de asignaturas de ITIS y en la Tabla 14.28 se muestran las de ITIG. Indicar las sentencias SQL necesarias para obtener una relación con la unión de todas las asignaturas.

**P14.25.** A partir de las relaciones que se muestran en las Tablas 14.25 y 14.28, indicar las sentencias SQL necesarias para obtener las asignaturas comunes a las titulaciones ITIS e ITIG.

**P14.26.** A partir de las relaciones que se muestran en las Tablas 14.25 y 14.28, indicar las sentencias SQL necesarias para

obtener las asignaturas que no son comunes a las titulaciones ITIS e ITIG.

**P14.27.** Diseñar una base de datos relacional para gestionar las existencias, compras y préstamos de los libros de una biblioteca.

**P14.28.** Diseñar una base de datos para gestionar los CD de música clásica de que se disponen en una familia. Considerar información tal como: título de la obra, compositor, intérprete, álbum, género (sinfónico, concierto, ópera, zarzuela, etc.), duración, sello discográfico, etc.

Id.	Nombre	Dep.	DNI profesor
IS	Ingeniería del Software	LI	25487632
EC	Estructura de Computadores	AC	47652347
EP	Evaluación de Prestaciones	AC	23428734
LP	Lenguajes de Programación	LI	54328975
SO	Sistemas Operativos	LI	25487632
PE	Periféricos	AC	47652347
SE	Sistemas Expertos	IA	43275832
BD	Bases de datos	IA	45345678

**Tabla 14.28.** Asignaturas ITIG.



# Ingeniería del software

Tradicionalmente el desarrollo de aplicaciones y programas informáticos ha sido considerado como una labor creativa e intuitiva, calificándolo con frecuencia como un *arte*. De hecho, uno de los libros que más difusión ha tenido en la historia de la informática lleva por título *El arte de la programación*, fue escrito por Donal Knuth en 1968, sus contenidos son plenamente vigentes<sup>1</sup> y al final de 1999 la Scientific Research Society de Estados Unidos lo eligió como uno de los doce mejores libros científicos del siglo XX.

No obstante lo anterior, la creciente complejidad de las aplicaciones y el incremento notable del volumen de comercialización de algunas de ellas ha obligado a establecer procedimientos y metodologías rigurosas para el desarrollo del software, de forma que la eficiencia y calidad de los productos obtenidos no fuesen tan sólo responsabilidad de la pericia e intuición de los analistas y programadores. Dentro de este contexto surgió la disciplina de **Ingeniería del Software**, que aunque se ha ido desarrollando muy paulatinamente desde que inició su andadura en 1969 (cuando se definió por primera vez) hoy en día es fundamental para construir eficientemente software de calidad. Esta disciplina, en definitiva, trata de aplicar o adaptar técnicas bien conocidas y probadas de las ingenierías convencionales a la construcción de aplicaciones software, considerando a éstas como productos a fabricar, comercializar y mantener.

En este capítulo se presentan los conceptos básicos de la Ingeniería del Software, describiendo aspectos tales como el ciclo de vida del software, las etapas y modelos que deben seguirse para el desarrollo de software y los factores que determinan e influyen en la calidad del software.

## 15.1. Introducción

La **Ingeniería del Software** es la disciplina que trata del establecimiento y uso de métodos y principios sólidos de ingeniería para obtener software fiable que se ejecute en máquinas reales. En definitiva, la ingeniería del software pretende aplicar en lo posible los procedimientos, bien conocidos y bien probados, de otras ingenierías considerando los aspectos tecnológicos y burocráticos relacionados con el diseño, producción y mantenimiento sistemáticos de programas de computadores. Los aspectos burocráticos se refieren a facetas tales como estimación de costes, asignación de personas a la

---

<sup>1</sup> Edición actual: Donald Ervin Knuth, *The art of computer programming*, Addison-Wesley, 2003.

realización de un proyecto, forma de realizar la documentación, mentalización de los usuarios para la aceptación del producto, etc.

La Ingeniería del Software se aplica fundamentalmente al desarrollo de programas o aplicaciones de cierta envergadura o que vayan a tener una amplia comercialización.

En resumen, la ingeniería del software se encarga de planificar y estimar proyectos, de analizar los requisitos del diseño del software, así como su codificación, prueba y mantenimiento. Existe una serie de diferencias con respecto al resto de las ingenierías, como, por ejemplo, que no hay un proceso de fabricación, o que casi todo el coste es para diseño. También es raro que se puedan volver a utilizar programas creados anteriormente, ya que el diseño depende de la aplicación específica. Otra diferencia es la tolerancia. En un programa las cosas funcionan o no, no se permite un rango de fallos. Y, por último, no existen medidas precisas de la calidad de un software.

En este capítulo vamos a ver los principios básicos de la ingeniería del software, así como las herramientas para el diseño y desarrollo de software.

## 15.2. Planificación y gestión de proyectos

Para desarrollar con rigor un proyecto es necesario establecer de una planificación previa, que determine el modelo de ciclo de vida a seguir, los plazos para el desarrollo de cada etapa y los medios necesarios en cada una de ellas. El propósito último de la planificación es realizar una estimación de tiempo y coste del desarrollo, determinando en detalle el proceso a seguir. Es conveniente escribir todos los detalles en un documento, que se suele conocer con el nombre de **plan de desarrollo del software**. El plan es un elemento fundamental para poder gestionar el proyecto, ya que permite prever los recursos que se van a utilizar, detectar posibles desviaciones de tiempo y presupuesto, y tomar medidas para corregir estas desviaciones. En muchos casos los proyectos no continúan más allá de la planificación, por ser el coste de desarrollo excesivo o el plazo de realización demasiado largo.

Para realizar la planificación es conveniente disponer de una información lo más detallada posible sobre el proyecto. Es decir, antes de iniciar la planificación se debe haber realizado, al menos, un avance del análisis.

En la realización de un proyecto se usan tres tipos de recursos: humanos, software y hardware.

- Los **recursos humanos** incluyen al personal que interviene en el proyecto. Normalmente el esfuerzo humano se mide en *personas × unidad de tiempo* (habitualmente en personas por mes), que representa la suma del tiempo empleado por todas las personas que intervienen en el proyecto.
- Los **recursos hardware** son los computadores en los que se ha de desarrollar el proyecto, los computadores en los que se usará el software (que pueden no ser los mismos) y los periféricos y dispositivos específicos que se necesiten en ambas máquinas.
- Los **recursos software** incluyen los paquetes y **herramientas software** usados como ayuda durante la realización del proyecto, tanto en la codificación (editores, compiladores, encuadernadores, depuradores simbólicos, bibliotecas, etc.) como en el diseño, especificación y planificación del proyecto.

Muchas veces es difícil y crítico estimar el número de personas que deben intervenir en el proyecto para acabarlo en una determinada fecha. Para poder resolver este problema se necesita cuantificar el esfuerzo necesario para completar el proyecto.

Por lo general, el establecimiento de medidas cuantitativas del software, que puedan ser estimadas en las fases iniciales del proyecto, es complejo. Sin embargo, se han propuesto diversas métricas para hacer una estimación rápida y aproximada del presupuesto del desarrollo de un proyecto informático. De entre éstas, cabe destacar la utilización como medida del **número de líneas de código** generadas, que si bien no es simple de estimar en las primeras etapas, ya que todavía no hay código que

medir, es fácil de medir a posteriori. La única forma de realizar estimaciones del tamaño del software es adquirir experiencia, y para ello la mejor estrategia es recoger datos históricos de todas las aplicaciones y los programas desarrollados (tanto personalmente como en la empresa).

## 15.3. Ciclo de vida del software

El **ciclo de vida** del software está constituido por las etapas o fases por las que pasa un programa o aplicación software a lo largo de su existencia. Estas etapas son (Figura 15.1):

1. **Desarrollo:** etapa en la que se definen los requisitos del sistema, se diseñan las estructuras de datos y de los programas, se implementan y se prueban.
2. **Uso:** etapa en la que se utiliza el software.
3. **Mantenimiento:** etapa en la que el software se somete a modificaciones debidas a fallos, actualizaciones o cambios necesarios a lo largo de su uso.

En resumen, los analistas y programadores desarrollan un software, éste se utiliza y si surgen errores o si hay que hacer actualizaciones, se modifica. Las etapas de uso y modificación se repiten hasta que el sistema quede obsoleto, ya sea porque ya no sea eficiente o quede anticuado o por otros factores.

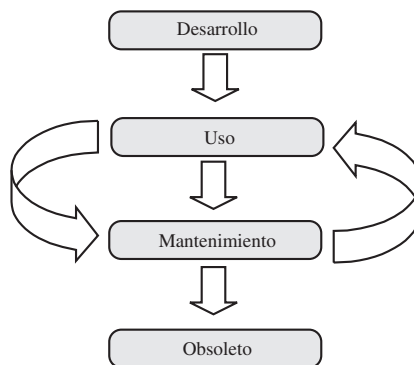


Figura 15.1. Ciclo de vida del software.

## 15.4. Etapas y modelos de desarrollo de software

Tradicionalmente las etapas que se consideran para el desarrollo de software de cierta envergadura<sup>2</sup> son: análisis, diseño, implementación y prueba.

- **Análisis:** el principal objetivo de esta fase es identificar lo que debe realizar el sistema a desarrollar.
- **Diseño:** una vez determinado lo que tiene que hacer el sistema, hay que decidir cómo se va a llevar a cabo; es decir, definir la estructura del software. Normalmente el problema se suele di-

<sup>2</sup> Se suele considerar software de envergadura aquel que necesita de varias personas para su desarrollo, o de una sola dedicada a ello más de tres meses.



vidir en otros más sencillos de manera que el proyecto queda descompuesto en módulos, facilitándose así su realización y posterior mantenimiento.

- **Implementación:** en esta fase se crea el sistema, es decir, se redactan los programas, se generan los ficheros de datos, se desarrollan bases de datos, etc.
- **Prueba:** esta fase está muy ligada a la implementación, ya que normalmente cada módulo se va implementando y probando de forma independiente.

En las Secciones 15.5 a 15.8 describiremos con más detalles las funciones a realizar en cada una de estas fases.

Para el desarrollo de software suelen considerarse dos modelos básicos: en cascada e incremental.

- **Modelo en cascada** (Figura 15.2): el proceso de desarrollo se hace en una sola dirección, es decir, una fase no se lleva a cabo hasta que no haya concluido la anterior. Las ventajas de este modelo son que cada fase tiene que estar totalmente terminada antes de empezar la siguiente, de forma que las personas que han efectuado cada una de ellas la conocen perfectamente, ya que la han realizado de principio a fin y la han probado antes de concluirla. Como cada fase queda perfectamente definida y documentada, cada nueva fase puede ser realizada por personas o equipos distintos. La desventaja fundamental de este modelo es que si al final surge algún problema es difícil localizarlo, ya que habría que examinar el proyecto entero reabriendo etapas anteriores.

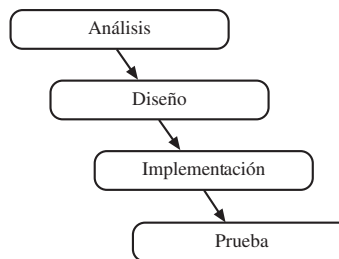


Figura 15.2. Modelo en cascada.

- **Modelo incremental** (Figura 15.3): el software se va desarrollando paso a paso, haciendo primero una versión simplificada de todo el proyecto, que suele incluir los módulos principales. Luego éstos se van completando poco a poco con submódulos, y así hasta concluir el proyecto. Es decir, se va añadiendo poco a poco funcionalidad. Si en el proceso se detectan errores se sabe perfectamente qué submódulo está fallando. Además, no se añaden módulos nuevos hasta que no funcionen los desarrollados previamente.

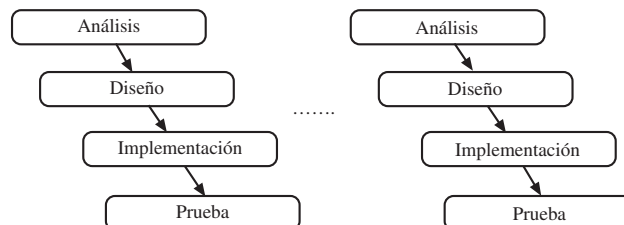


Figura 15.3. Modelo incremental.

El modelo incremental también se conoce con el nombre de **prototipado**, que se basa en la construcción de un **prototipo** o modelo “a escala reducida” del sistema que se va a construir.

Éste incorpora sólo los aspectos relevantes del sistema y se utiliza como ayuda en la especificación del software, sirviendo como base tangible sobre la que discutir con el cliente. Este modelo se suele utilizar cuando en la fase de análisis es difícil establecer los requisitos software a priori.

Un aspecto muy importante del diseño, uso y mantenimiento del software es la documentación. El conjunto de información generada en las distintas fases se suele denominar **configuración del software**. La documentación es imprescindible y un factor que incide notablemente en la calidad del software. Con la configuración del software se pretende que un programa o aplicación pueda ser desarrollado por distintos grupos y personas, e incluso que al cabo del tiempo pueda ser actualizado o modificado con facilidad y por personas distintas a las que desarrollaron la versión original. Normalmente un software va acompañado de dos tipos de documentación:

- **Documentación para el usuario:** se le suele denominar manual, y no es más que una explicación de todos los pasos a realizar para poder utilizar el software y una explicación del funcionamiento de cada parte del sistema.
- **Documentación técnica:** sirve para que los propios creadores del software puedan en un futuro modificar y mantener eficientemente el sistema. Cada etapa del desarrollo del software (análisis, diseño, implementación y prueba) debe concluir con un documento en el que se detallen con precisión las opciones consideradas e implementadas. En la fase de análisis también debe incluirse un **documento de definición de requisitos** que debe ser firmado tanto por el analista de la aplicación como por el usuario.

En cada una de las fases de desarrollo de un proyecto se suelen utilizar tablas y diagramas o esquemas gráficos específicos que ayudan a su realización. Así, por ejemplo, para la **planificación temporal** del proyecto se puede utilizar un *diagrama de Gantt* en la que se indican las tareas que se deben realizar y las fechas previstas de inicio y fin de cada una de estas tareas. A lo largo de las Secciones 15.5 y 15.6 se describirán otros diagramas.

### EJEMPLO 15.1

En la Figura 15.4 se muestra un diagrama de Gantt asociado a la temporización del desarrollo de un hipotético proyecto.

Tarea	Abril	Mayo	Junio	Julio	Agosto	Septiembre
Análisis de necesidades	■	■				
Análisis de requisitos		■	■			
Análisis de especificaciones			■	■		
Diseño: modularización			■	■		
Diseño: descripción modular				■	■	
Codificación: programas				■	■	
Pruebas					■	■
Correcciones						■
Medida de prestaciones						■
Documentación						■

Figura 15.4. Diagrama de Gantt.

En otro orden de cosas, se han desarrollado herramientas informáticas sofisticadas para el desarrollo de software. Este tipo de aplicaciones se engloban bajo la denominación de **ingeniería del software con ayuda de computador** (CASE, *computer-aided software engineering*). Estos sistemas facilitan notablemente la planificación y gestión de proyectos, el desarrollo de documentación, la realización de prototipos, la simulación y el diseño de interfaces.

## 15.5. Análisis

Según se ha comentado anteriormente, el principal objetivo del análisis es identificar lo que debe realizar el sistema a desarrollar. En esta fase se definen los **requisitos del software** o conjunto de funciones y prestaciones que debe incluir el sistema para llevar a cabo su objetivo. La definición de requisitos del software implica que se tenga en cuenta a quien va dirigido, que el cliente defina con claridad lo que debe hacer el sistema, que el analista determine los medios que va a necesitar y por último establecer los métodos a utilizar en el desarrollo del sistema. La definición de requisitos concluye con la redacción de un documento, comprensible por el cliente, y que debe ser firmado por éste y el analista de la aplicación. Una vez identificados los requisitos hay que convertirlos en **especificaciones técnicas** orientadas al diseño de la aplicación.

En resumen, la fase de análisis considera las siguientes cuestiones:

- Identificación del ámbito de utilización de la aplicación (uso genérico o de un determinado grupo de personas con una formación específica).
- Descripción de necesidades e intereses del cliente.
- Definición de requisitos.
- Definición de especificaciones y métodos.

### EJEMPLO 15.2

En una determinada aplicación se pueden establecer los siguientes **requisitos**: el acceso debe ser restringido a los usuarios autorizados, la interfaz debe ser muy cómoda para usuarios no especialistas en informática, los precios y contabilidad debe hacerse en euros, etc.

Los requisitos anteriores pueden dar lugar a las siguientes **especificaciones**:

- El acceso debe realizarse con un nombre de usuario (seis caracteres) y una contraseña (ocho caracteres).
- Debe implementarse una interfaz gráfica basada en iconos, similar a la de entornos MS-Windows.
- Para las variables contables utilizar números reales, simple precisión, y las visualizaciones hacerlas con dos cifras decimales, etc.

En la fase de análisis se pueden utilizar herramientas que faciliten la notación para expresar los requisitos del sistema, como por ejemplo:

- **Tablas de decisión.** Muestran las acciones que debe realizar el sistema ante determinadas entradas (Figura 15.5).
- **Diagramas de estado.** Un diagrama de estado es un grafo en el que cada nodo representa un estado del sistema, y los arcos son transiciones entre estados provocadas por determinadas condiciones ya sean externas, internas o espontáneas (Figura 15.6).
- **Diagramas de flujo de datos (DFD).** Un DFD representa la información que se transmite de la entrada a la salida (mediante arcos etiquetados) y las transformaciones que se realizan con ella (círculos) (Figura 15.7).

		Reglas			
		1	2	3	4
Condiciones	Campos obligatorios rellenos	N	S	S	S
	Campos correctos	—	N	S	S
	Envío de datos correcto	—	—	S	N
	Envío de datos fallido	—	—	—	S
Acciones	Rellenar campos obligatorios	X			
	Corregir campos erróneos		X		
	Mensaje de confirmación			X	
	Mensaje de error				X

Figura 15.5. Tabla de decisión.

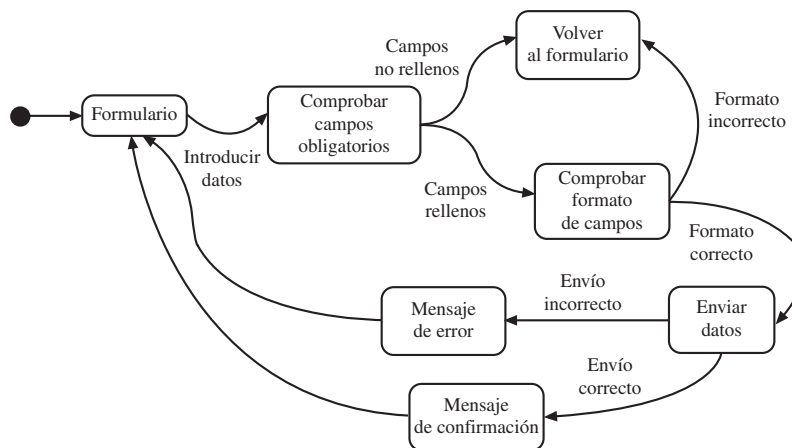


Figura 15.6. Diagrama de estados.

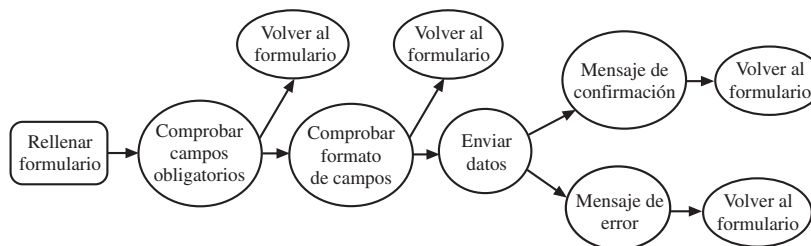


Figura 15.7. Diagrama de flujo de datos.

**EJEMPLO 15.3**

Supóngase que se desea realizar un programa para cumplimentar un formulario en la web y que al pulsar un botón de enviar transmita por correo electrónico los datos recopilados en dicho formulario. Para poder enviar los datos el programa hay que comprobar previamente si el usuario ha rellenado los campos obligatorios, y si éstos se han rellenado, comprobar si tienen el formato adecuado (por ejemplo, un código postal son 5 números). Una vez se haya pulsado el botón de envío debe aparecer una página

con un mensaje de confirmación junto con los datos del formulario. Si se ha producido algún error en el envío de los datos debe visualizarse una página con un mensaje de error y un enlace al formulario, en lugar del mensaje de confirmación. Las Figuras 15.6 y 15.7, respectivamente, muestran diagramas de estados y de flujo de datos para este programa.

## 15.6. Diseño

En el diseño se crea una estructura para el software que satisfaga las especificaciones dadas en la fase de análisis. El diseño determina “cómo” se va a construir el sistema. El resultado del trabajo de diseño es la **documentación del diseño** que forma parte de la configuración del software. En el diseño se establece la estructura de los programas y de los datos usados por éstos. Una de las tareas más relevantes del diseño es la descomposición modular del problema, de forma que cada módulo realice una tarea sencilla determinada a implementar (en una fase posterior) en forma de un programa. En determinadas aplicaciones, como es el caso de los sistemas embebidos, en esta etapa se decide incluso el tipo de implementación más adecuada para cada módulo: software o hardware.

En la **fase de diseño** se puede utilizar una gran variedad de herramientas. Una de las más conocidas para representar la estructura del software son las cartas de estructura. Una **carta de estructura** es un diagrama jerárquico en el que aparecen los módulos o procedimientos que forman el sistema (cajas o rectángulos) y las relaciones entre éstos (flechas de interconexión de cajas). Un procedimiento que utiliza o llama a otro se une por medio de una flecha que va del primero al segundo. Esta carta, por tanto, refleja las dependencias de control entre los componentes del software.

### EJEMPLO 15.4

Para el Ejemplo 15.3 una posible carta de estructura es la que se indica en la Figura 15.8. Cada rectángulo representa un módulo independiente, y las flechas las dependencias de unos con otros.

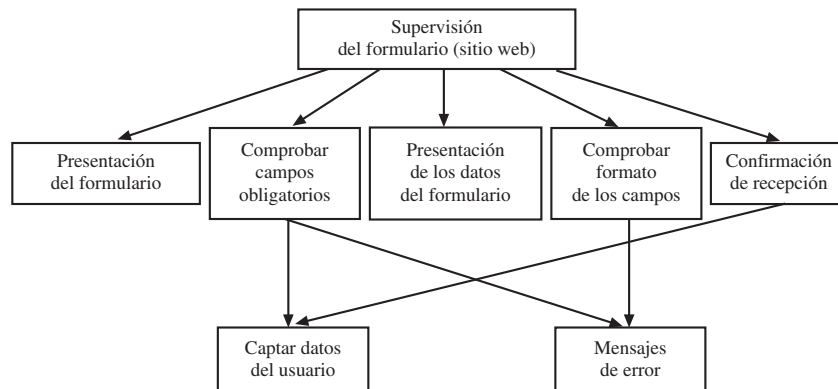


Figura 15.8. Carta de estructura.

En el diseño de sistemas orientados a objetos se suelen utilizar **diagramas de clases**, que representan con rectángulos las clases de objetos y con flechas las relaciones entre ellos.

### EJEMPLO 15.5

En el Ejemplo 15.3 podemos identificar dos objetos: usuario y formulario. La relación que hay es uno a uno, ya que cada usuario rellena un único formulario y cada formulario corresponde a un único usuario. El diagrama de clases se representa en la Figura 15.9a).

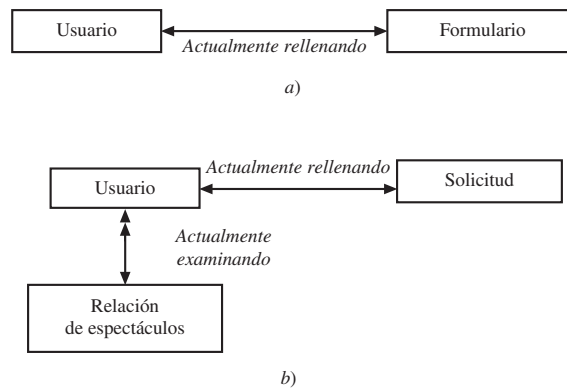


Figura 15.9. Ejemplos de diagramas de clases.

### EJEMPLO 15.6

Supongamos ahora que se desea realizar una aplicación para reserva de entradas de espectáculos por Internet. Podemos identificar tres tipos de objetos: tipo usuario o cliente, tipo solicitud de reserva y tipo catálogo o relación de espectáculos ofrecidos. La relación solicitud-usuario es uno a uno (cada usuario rellena un único formulario); en cambio la relación espectáculos-usuario es una a muchos, ya que en un momento dado muchos clientes pueden estar accediendo simultáneamente a la relación de espectáculos. En la Figura 15.9b) se muestra el diagrama de clases correspondiente.

#### 15.6.1. MODELOS DE DISEÑO

Según se ha indicado anteriormente, el diseño trata de establecer la forma de cómo realizar el sistema proyectado. Se puede realizar siguiendo diversos modelos; los más conocidos son:

- **Modelo de arriba a abajo:** consiste en dividir el proyecto en problemas más sencillos y más fáciles de manejar. Estos problemas se pueden volver a fragmentar a su vez en otros más sencillos, de manera que el conjunto de subproblemas de menor complejidad resuelvan el problema general. Es decir, consiste en una descomposición jerárquica en el que los últimos subproblemas obtenidos se puede traducir en módulo fácilmente programables.
- **Modelo de abajo a arriba:** primero se identifican tareas individuales dentro del sistema y cómo encontrar soluciones a ellas, de forma que se puedan utilizar para solucionar problemas más complejos.
- **Modelo suscriptor-editor:** consta de un módulo (editor) que envía copias de sus “publicaciones” a otros módulos (suscriptores).
- **Modelo envase-contenido:** consiste en tener módulos que a su vez contienen otros módulos, y así sucesivamente (como la estructura de directorios de un computador).
- **Modelo de marcos:** se crean programas que implementan ciertas características de la solución pero no se especifican los detalles. Cada programa va acompañado de una documentación en la que se indica cómo completar el programa para que implemente una aplicación en particular.
- **Software abierto:** es otra forma de desarrollar software. El autor original desarrolla una versión inicial de un software y lo publica en Internet (tanto el código como la documentación). Cualquier usuario tiene acceso a esta versión, que puede modificar o ampliar. Estas modificaciones o ampliaciones se envían al autor original, que vuelve a publicar la nueva versión, y así sucesivamente. El ejemplo más notable del uso de este modelo se encuentra en la creación de Linux.

En la actualidad la Ingeniería del Software usa con gran frecuencia el **lenguaje unificado de modelado, UML** (*Unified Modelling Language*), que es un lenguaje gráfico normalizado para mostrar de forma visual la especificación y el diseño del software. UML permite describir gráficamente distintos aspectos del software, de un modo equivalente a como un arquitecto representa la estructura de un edificio. La descripción de un sistema en UML incluye una serie de modelos, referidos a distintos aspectos del sistema, cada uno de los cuales se representa utilizando un conjunto de diagramas específico. Los modelos pueden agruparse en los siguientes:

- **Modelo de comportamiento.** Describe el funcionamiento del sistema, detallando las acciones de interacción entre usuario y sistema, y entre objetos del sistema.
- **Modelo estructural.** Describe la estructura de clases y objetos del sistema.
- **Modelo arquitectónico.** Describe y determina la forma de implementar los distintos módulos: por hardware específico o por software, y en este último caso las plataformas a usar.

La forma más habitual de realizar un diseño es descomponiendo el sistema completo en módulos (diseño de arriba abajo). Debido a la importancia de la descomposición modular le dedicamos la próxima sección.

### 15.6.2. MODULARIDAD

Hay proyectos tan grandes que sería imposible tratarlos si no se descompusieran en partes. Así que normalmente un proyecto se fragmenta, de forma que en vez de tratar un gran problema se tienen pequeños problemas que se comunican entre ellos. Esto da lugar a la modularidad. Cada módulo realiza una función concreta y presenta una interfaz sencilla, pero a su vez con conexión con el resto de los módulos. La modularidad se lleva a cabo según el método utilizado en la fase de diseño (Sección 15.6.1).

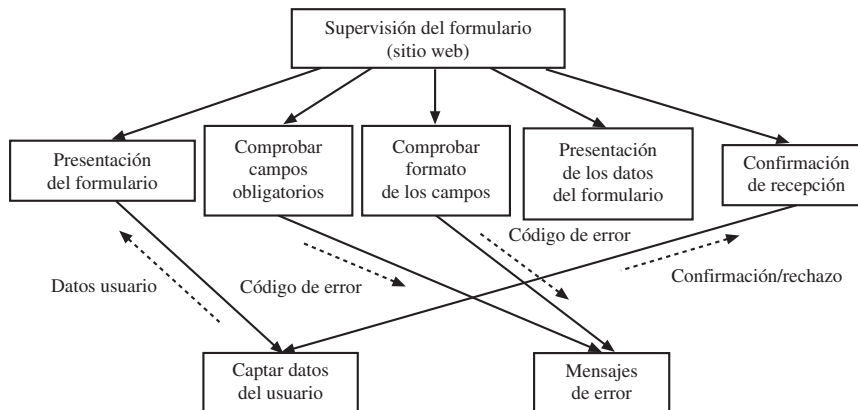
Para determinar las relaciones entre módulos se definen los conceptos de acoplamiento y cohesión.

- El **acoplamiento** es una medida de la dependencia entre dos módulos. Siempre hay que procurar que el acoplamiento entre dos módulos sea mínimo, es decir, que la independencia entre los módulos sea máxima. Al aumentar el nivel de acoplamiento, aumenta la dependencia entre los módulos y, por tanto, la dificultad de prueba y mantenimiento. Un error en un módulo puede propagarse y afectar a otro, y una modificación en un módulo puede obligar a modificar otro. El acoplo entre módulos puede ser de distinta naturaleza. El **acoplamiento de control** se produce cuando un módulo pasa el control a otro, por ejemplo llamando el primero al segundo como un subprograma. El acoplamiento de control entre módulos se puede visualizar con la carta de estructura (Figura 15.8). Otra forma de interacción entre módulos se produce con el **acoplamiento de datos**, que tiene lugar cuando se comparten datos entre distintos módulos. Para visualizar el acoplamiento de datos se puede incluir en la carta de estructura flechas adicionales mostrando la información que se transfiere entre los procesos. También en los diseños orientados a objetos el acoplamiento de datos se visualiza añadiendo a los diagramas de clases flechas indicando las transferencias de información entre clases. Estos últimos diagramas se conocen como **diagramas de colaboración**.
- La **cohesión** es una medida de la interdependencia de las distintas partes de un módulo. El nivel de cohesión de cada módulo debe ser lo más alto posible. Un módulo con baja cohesión realizará funciones poco relacionadas y deberá descomponerse en varios módulos de tal forma que cada uno realice una única función. La cohesión se puede considerar bajo dos puntos de vista: la **cohesión lógica**, que se produce cuando los elementos internos de un módulo realizan actividades lógicamente similares, y la **cohesión funcional**, que se obtiene cuando todas las partes de un módulo están diseñadas para llevar a cabo una única actividad.

**EJEMPLO 15.7**

El acoplamiento de datos de la aplicación planteada en el Ejemplo 15.3 se puede visualizar con el diagrama de la Figura 15.10. Se incluye el intercambio entre los siguientes procesos:

- Entre los procesos *captar datos de usuario* y *presentación de formulario*: datos de usuario (los que incluye en el formulario).
- Entre los procesos *captar datos de usuario* y *confirmación de recepción*: aceptación o rechazo del usuario.
- Entre los procesos *comprobar campos obligatorios* y *mensajes de error*: código de error.
- Entre los procesos *comprobar formato de los campos* y *mensajes de error*: código de error.



**Figura 15.10.** Carta de estructura mostrando el acoplamiento de datos.

## 15.7. Implementación

En la **fase de implementación** se construye el sistema; es decir, se escriben los programas, se crean los archivos de datos, se desarrollan las bases de datos, etc. Esta fase se puede llevar a cabo en varios pasos. Primero, con la ayuda de herramientas como organigramas (Sección 11.2.2) se describe lo que hacen los programas, luego se definen en **pseudocódigo** (Sección 11.2.1) y por último se codifican (escriben) los programas en un lenguaje de alto nivel adecuado para la aplicación de que se trata (Sección 12.5).

En las Secciones 11.6 y 12.4 se describieron los pasos a seguir para el desarrollo y ejecución de programas, respectivamente.

## 15.8. Prueba

Una de las fases más importantes y complejas en el desarrollo de cualquier software es su prueba. No se puede decir que un sistema software es correcto, ya que es imposible realizar una **prueba exhaustiva**, analizando todas y cada una de las situaciones a las que se va a someter en un futuro. Debido a ello, los ingenieros de software desarrollan series de pruebas para poder localizar los errores más probables. Conviene tener en cuenta que para probar un sistema no basta con hacerlo con sus módulos individuales, sino que después de hacerlo así hay que realizar **pruebas de integración** con todos los módulos acoplados.



Los fallos pueden consistir en errores o deficiencias en la lógica del programa, en errores de precisión, en las prestaciones (velocidad, etc.), en la documentación, o incluso en los procedimientos seguidos durante el desarrollo que dificultarán el mantenimiento del sistema. Los tipos de pruebas suelen ser de caja de cristal o de caja negra.

La metodología de tipo **test de caja de cristal** supone que el interior del software es visible; es decir, el programador, que conoce los programas a fondo, puede comprobar que el sistema funciona correctamente mediante el examen y desarrollo de datos de prueba. En definitiva, la detección de fallos se hace por **inspección**, consistente en la revisión personal de cada módulo del sistema. La experiencia ha demostrado que identificando los módulos problemáticos y probándolos cuidadosamente se encuentran más errores que probando el sistema en su conjunto (este aserto se conoce con el nombre de **principio de Pareto**).

En el **test de caja negra** se realizan ensayos que tratan de determinar no sólo si la funcionalidad del sistema es la correcta, sino también de evaluar los requisitos no funcionales. El ingeniero que prueba el sistema utiliza sus conocimientos sobre éste y el entorno de trabajo que va a tener el usuario para comprobar que se cumplen los requisitos. Por lo general, lo que se hace es desarrollar conjuntos **de datos de prueba** de forma que nos aseguremos que se van a ejecutar todas las bifurcaciones posibles o líneas de código del sistema. Así por lo menos se asegura que todas las líneas se han probado al menos una vez. El objetivo es elegir datos de prueba de forma que analizando las salidas obtenidas se pueda detectar el mayor número de fallos posible. Existen métodos para obtener sistemáticamente los casos de prueba a partir de las especificaciones.

Independientemente del proceso de prueba seguido, no es posible garantizar que el sistema esté libre de fallos. Por este motivo es importante establecer criterios que nos den una idea de la probabilidad de que queden fallos sin detectar en el sistema. Una técnica simple para conseguir este objetivo es aplicar **redundancias**, utilizando, por ejemplo, dos grupos independientes (incluso de distintas empresas) para desarrollar el mismo producto software partiendo de las mismas especificaciones. Una vez contruidos los dos sistemas se les somete a ensayo con los mismos conjuntos de pruebas; en los casos de que haya discrepancias en las salidas es que hay errores en uno u otro sistema.

Otra técnica para ensayar software nuevo, y que se puede utilizar conjuntamente con las anteriormente descritas, se denomina **prueba beta**, y consiste en, una vez probada la aplicación por sus desarrolladores y programadores, cederla de forma gratuita a determinados usuarios a cambio de que informen al fabricante de todos los errores y deficiencias encontradas. Una vez que ha transcurrido un período razonable de tiempo, el fabricante modifica la versión preliminar (denominada **versión beta**) corrigiendo las anomalías detectadas por los distintos usuarios.

## 15.9. Calidad

En el ámbito de la Ingeniería del Software, el **software de calidad** es aquel que satisface los requisitos explícitos e implícitos dados por el cliente, que está bien documentado, y que funciona eficientemente en el hardware para el que ha sido diseñado.

Para evaluar la calidad del software se utilizan por una parte *medidas cuantitativas* (por ejemplo, medida estadística del tanto por ciento de fallos por línea de código) y por otra parte medidas *cualitativas* (por ejemplo, medida de la flexibilidad de la aplicación desarrollada). Podemos dividir estas medidas en tres grupos:

- **Medidas relacionadas con el funcionamiento.** Tratan de determinar si el software cumple los requisitos de funcionamiento para los que fue creado, si el entorno es cómodo de utilizar, etc. Hay una serie de factores que se pueden utilizar para medir la calidad del funcionamiento como, por ejemplo, la *precisión* (tiempo medio entre fallos, tanto por ciento de errores por línea de código, etc.), *facilidad de uso* (¿su apariencia es agradable?, ¿su uso es cómodo e intuitivo?), *eficiencia* (tiempo de respuesta), *fiabilidad* (¿los usuarios confían en él?), *seguridad* (di-

ficultad para que usuarios no autorizados accedan a la aplicación), *grado de utilización* (¿se utiliza mucho o poco?).

- **Medidas relacionadas con el mantenimiento.** Versan sobre la facilidad de mantener el software. La necesidad del mantenimiento de un sistema es debida a alguno o varios de los siguientes motivos: corrección de errores, mejora de prestaciones (optimización de los algoritmos, por ejemplo) y adaptación a nuevas circunstancias (cambios legislativos, por ejemplo). Se pretende que el software funcione de la mejor forma a lo largo del mayor tiempo posible. Hay programas que permiten medir la **complejidad de un sistema** desde el punto de vista de su actualización; cuanto más complejo es, más difícil será hacer cualquier actualización, teniendo un mantenimiento más complicado, lento y costoso. También se pueden estimar: la **capacidad de corrección de un sistema** o tiempo medio que se tardaría en arreglarlo si se produjera algún fallo, la **flexibilidad** (facilidad para hacer cambios) y la **testeabilidad** (facilidad para hacer pruebas en el sistema).
- **Medidas relacionadas con la compatibilidad.** Determinan la facilidad de intercambiar datos con otros sistemas, de cambiar de plataforma y de reutilizar el código. Los buenos programadores hacen bibliotecas de funciones de forma que en un futuro se pueda **reutilizar el código**; es decir, desarrollan los subprogramas pensando que puedan ser aprovechados por otras nuevas aplicaciones. También influye en la calidad de un paquete o aplicación software su **interoperabilidad** o capacidad de interactuar con otros sistemas (como por ejemplo intercambiar datos), y su **portabilidad** o posibilidad de cambiar de plataforma (Windows, Macintosh, etc.).

## 15.10. Conclusiones

La Ingeniería del Software trata de aplicar técnicas de las ingenierías convencionales para desarrollar aplicaciones o paquetes informáticos, considerados éstos como productos a fabricar, comercializar y mantener.

El objetivo es obtener programas de calidad, en el sentido de que: *a*) funcionen eficientemente y de acuerdo con los requisitos y prestaciones proyectadas; *b*) sean fáciles de mantener y actualizar, y *c*) sean compatibles con otras aplicaciones.

En este capítulo hemos presentado los conceptos básicos de la Ingeniería del Software. Así, después de hacer algunas consideraciones sobre la planificación y gestión de proyectos (Sección 15.2), hemos descrito aspectos tales como el ciclo de vida del software (Sección 15.3) y las etapas y modelos que deben seguirse para el desarrollo de software (Sección 15.4). Las siguientes secciones las hemos dedicado a estudiar las cuatro etapas que se suelen utilizar para desarrollar software, a saber: análisis (Sección 15.5), diseño (Sección 15.6), implementación (Sección 15.7) y prueba (Sección 15.8). El capítulo concluye con una sección (Sección 15.9) dedicada a analizar los principales factores que determinan e influyen en la obtención de software de calidad.

### Test



**T15.1.** La ingeniería del software es la disciplina que trata de los aspectos tecnológicos y burocráticos relacionados con:

- a*) El desarrollo de sistemas operativos, como interfaz software entre hombre y máquina.
- b*) La implementación de software residente en la memoria ROM del computador.

- c*) El diseño, producción y mantenimiento sistemáticos de programas de computadores.
- d*) La construcción (fabricación) de software.

**T15.2.** Las principales diferencias entre la ingeniería del software y cualquier otra ingeniería residen en las fases de:

- a) Análisis del sistema y especificación de requisitos.
- b) Diseño e implementación.
- c) Diseño y prueba.
- d) Fabricación y mantenimiento.

**T15.3.** Para realizar una estimación rápida y aproximada del presupuesto del desarrollo de un proyecto informático se suele considerar:

- a) El valor de la amortización de las plataformas hardware y software utilizadas más el coste de las nóminas del personal involucrado.
- b) El número de líneas de los programas.
- c) El número de personas  $\times$  unidad de tiempo (personas por mes).
- d) Un diagrama de Gantt.

**T15.4.** La fase en la que se definen los requisitos y métodos que se van a usar en el desarrollo del sistema es:

- a) Análisis.
- b) Diseño.
- c) Implementación.
- d) Prueba.

**T15.5.** La fase en la que se define la estructura del software, es decir, cómo se va a llevar a cabo el sistema (qué es lo que hay que hacer), es:

- a) Análisis.
- b) Diseño.
- c) Implementación.
- d) Prueba.

**T15.6.** La fase en la que se escriben los programas que van a constituir el sistema es:

- a) Análisis.
- b) Diseño.
- c) Implementación.
- d) Prueba.

**T15.7.** La fase en la que se examinan y depuran los programas que constituyen el sistema es:

- a) Análisis.
- b) Diseño.
- c) Implementación.
- d) Prueba.

**T15.8.** La fase del desarrollo del software en la que se diseñan las estructuras de los datos y de los programas es la:

- a) Fase de análisis.
- b) Fase de diseño.
- c) Fase de implementación.
- d) Fase de prueba.

**T15.9.** La fase del desarrollo del software en la que se debe realizar la documentan del sistema en construcción es:

- a) Fase de análisis.
- b) Fase de implementación.
- c) Fase de diseño.
- d) En todas.

**T15.10.** Dentro del contexto de la ingeniería del software, la configuración del software es el conjunto de:

- a) Información generada en todas las fases.
- b) Parámetros que particularizan una instalación de software en un computador concreto.
- c) Programas de una aplicación.
- d) Requisitos que debe cumplir una instalación hardware para que funcionen correctamente una aplicación software.

**T15.11.** La fase de desarrollo del software en la que se especifica qué módulos deben implementarse con hardware y cuáles con software es la de:

- a) Análisis.
- b) Diseño.
- c) Implementación.
- d) Prueba.

**T15.12.** La fase de desarrollo del software en la que se determinan las interfaces que debe presentar, el rendimiento y la utilización de recursos es la de:

- a) Análisis.
- b) Diseño.
- c) Implementación.
- d) Prueba.

**T15.13.** La fase del desarrollo del software en la que se prueba y depura el sistema es:

- a) Análisis del sistema.
- b) Diseño.
- c) Implementación.
- d) Prueba.

**T15.14.** El ciclo de vida de prototipos se suele utilizar cuando es difícil de realizar la fase de:

- a) Análisis del sistema.
- b) Diseño.
- c) Implementación.
- d) Prueba.

**T15.15.** El modelo de ingeniería del software en el que no se realiza una fase hasta que no se haya terminado la anterior es el:

- a) Modelo en cascada.
- b) Modelo incremental.
- c) De ciclo de vida de prototipos.
- d) De ingeniería del software asistido por ordenador (CASE).

**T15.16.** El modelo de ingeniería del software en el que primero se hace un modelo a escala reducida y poco a poco se va complicando el modelo hasta obtener el sistema proyectado es el:

- a) Modelo en cascada.
- b) Modelo incremental.
- c) De ciclo de vida de prototipos.
- d) De ingeniería del software asistido por ordenador (CASE).

**T15.17.** El modelo de ingeniería del software en el que el software se va desarrollando paso a paso y luego se va completando cada fase es el:

- a) Modelo en cascada.
- b) De ciclo de vida de prototipos.
- c) De ingeniería del software asistido por ordenador (CASE).
- d) Modelo incremental.

**T15.18.** Un diagrama de Gantt se utiliza para:

- a) Realizar una planificación temporal del proyecto.
- b) Estimar el esfuerzo de desarrollo de una aplicación (personas-mes).
- c) Mostrar las acciones que debe realizar el sistema ante determinadas condiciones de entrada.
- d) Determinar las dependencias entre procesos.

**T15.19.** Para determinar las dependencias entre tareas y el tiempo de las mismas debe usarse:

- a) Un diagrama de Gantt.
- b) Un esquema del ciclo de vida.
- c) Una carta de estructura.
- d) Una tabla de decisión.

**T15.20.** Una tabla de decisión se utiliza para:

- a) Realizar una planificación temporal del proyecto.
- b) Estimar el esfuerzo de desarrollo de una aplicación (personas-mes).
- c) Mostrar las acciones que debe realizar el sistema ante determinadas condiciones de entrada.
- d) Determinar las dependencias entre tareas y tiempo de las mismas.

**T15.21.** Para mostrar las acciones que debe realizar el sistema ante determinadas condiciones de entrada debe usarse:

- a) Un diagrama de Gantt.
- b) Un diagrama de estado.
- c) Un diagrama de flujo de datos.
- d) Una tabla de decisión.

**T15.22.** Si deseamos modelar el comportamiento de un sistema ante condiciones externas, internas o espontáneas debemos utilizar:

- a) Un diagrama de estados.
- b) Un diagrama de flujo de datos (DFD).
- c) Una tabla de decisión.
- d) Un diagrama Gantt.

**T15.23.** Si deseamos modelar el comportamiento de un sistema representando la información que se transmite en el sistema, de la entrada a la salida, y las transformaciones que se realizan con ella, debemos utilizar:

- a) Un diagrama de estados.
- b) Un diagrama de flujo de datos (DFD).
- c) Una tabla de decisión.
- d) Un diagrama Gantt.

**T15.24.** Un diagrama jerárquico en el que aparecen los módulos que forman el sistema y las relaciones de utilización que aparecen entre éstos se denomina:

- a) Diagrama de estados.
- b) Diagrama de flujo de datos (DFD).
- c) Carta de estructura.
- d) Diagrama de colaboración.

**T15.25.** En el diseño de sistemas orientados a objetos se suelen utilizar:

- a) Diagramas de estados.
- b) Diagramas de flujo de datos (DFD).
- c) Diagramas de clases.
- d) Cartas de estructura.

**T15.26.** Cuando un proyecto se jerarquiza en subproblemas de forma que cada subproblema se puede traducir en un módulo, se está utilizando el modelo:

- a) De arriba abajo.
- b) De abajo a arriba.
- c) Suscriptor-editor.
- d) Envase-contenido.

**T15.27.** Cuando en un proyecto se identifican tareas individuales dentro del sistema y cómo encontrar las soluciones a estas tareas de forma que se pueden utilizar para solucionar problemas más complejos se está utilizando el modelo:

- a) De arriba abajo.
- b) De abajo a arriba.
- c) Suscriptor-editor.
- d) Envase-contenido.

**T15.28.** Cuando un proyecto consta de un módulo que envía copias de sus “publicaciones” a otros módulos se está utilizando el modelo:

- a) De arriba abajo.
- b) De abajo a arriba.
- c) Suscriptor-editor.
- d) Envase-contenido.

**T15.29.** Cuando un proyecto está formado por módulos que a su vez contienen otros módulos, y así sucesivamente (como la estructura de directorios de un computador), se está utilizando el modelo:

- a) De arriba abajo.
- b) De marcos.
- c) Suscriptor-editor.
- d) Envase-contenido.

**T15.30.** Cuando en un proyecto se crean programas que implementan ciertas características de la solución pero no se especifican los detalles, se está utilizando el modelo:

- a) De arriba abajo.
- b) De marcos.
- c) Suscriptor-editor.
- d) Envase-contenido.

**T15.31.** El Lenguaje Unificado de Modelado (UML) es un lenguaje ideado para:

- a) Redactar adecuadamente la documentación de desarrollo del software.
- b) Descomponer el sistema en módulos.
- c) Expresar de forma visual las especificaciones y el diseño del software.
- d) Seleccionar el modelo más adecuado de implementación.

**T15.32.** ¿Qué concepto propone que un problema complejo se descomponga en problemas más sencillos?

- a) Cohesión.
- b) Acoplamiento.
- c) Modularidad.
- d) Funcionalidad.

**T15.33.** La medida de la dependencia entre dos módulos viene dada por:

- a) La cohesión.
- b) El acoplamiento.
- c) La modularidad.
- d) La funcionalidad.

**T15.34.** La medida de la interdependencia entre las distintas partes de un módulo viene dada por:

- a) La cohesión.
- b) El acoplamiento.
- c) La modularidad.
- d) La funcionalidad.

**T15.35.** La medida de la dependencia entre dos módulos se denomina:

- a) Acoplamiento.
- b) Cohesión.
- c) Ocultación.
- d) Grado de abstracción.

**T15.36.** La medida de la concreción funcional de un módulo se denomina:

- a) Acoplamiento.
- b) Cohesión.
- c) Ocultación.
- d) Grado de abstracción.

**T15.37.** Un diseño debe pretender que:

- a) El acoplo entre módulos y la cohesión sean mínimos.
- b) El acoplo entre módulos sean mínimo y la cohesión máxima.
- c) El acoplo entre módulos sean máximo y la cohesión mínima.
- d) El acoplo entre módulos y la cohesión sean máximos.

**T15.38.** Se utiliza la descomposición del sistema en una serie de clases en el:

- a) Diseño orientado a flujo de datos.
- b) Diseño orientado a las estructuras de datos.
- c) Diseño orientado a objetos.
- d) Establecimiento de las pruebas de integración.

**T15.39.** La verificación que un conjunto de módulos funciona correctamente, después de haberlos testado individualmente, se denomina:

- a) Inspección DFD.
- b) Inspección orientada a estructuras de datos.
- c) Inspección orientada a módulos-objetos.
- d) Pruebas de integración.

**T15.40.** Es test de la caja de cristal lo realiza:

- a) El usuario.
- b) El programador.
- c) Un software de prueba.
- d) El analista.

**T15.41.** Es test de la caja negra lo realiza:

- a) El usuario.
- b) El técnico de mantenimiento.
- c) Un software de prueba.
- d) El ingeniero.

**T15.42.** La precisión, eficiencia, fiabilidad, seguridad, velocidad de respuesta y utilización son parámetros que permiten medir:

- a) El funcionamiento del sistema.
- b) El mantenimiento del sistema.
- c) La compatibilidad del sistema.
- d) La longevidad del sistema.

**T15.43.** La reutilización del código, la capacidad de interactuar con otros sistemas y la portabilidad son parámetros que permiten medir:

- a) El funcionamiento del sistema.
- b) El mantenimiento del sistema.
- c) La compatibilidad del sistema.
- d) La longevidad del sistema.

**T15.44.** La complejidad del sistema, el tiempo medio en solventar un fallo y la testeabilidad son parámetros que permiten medir:

- a) El funcionamiento del sistema.
- b) El mantenimiento del sistema.
- c) La compatibilidad del sistema.
- d) La longevidad de un sistema.

## Problemas resueltos



## CONCEPTOS BÁSICOS

- P15.1.** Teniendo en cuenta que, en general, la ingeniería puede considerarse la habilidad de construir sistemas a partir de componentes genéricos prefabricados, indicar, razonando la respuesta, las peculiaridades de la ingeniería del software en este sentido.

## SOLUCIÓN

Las aplicaciones informáticas no se realizan a partir de componentes genéricos prefabricados, que en este caso serían programas. Por lo general, para cada aplicación hay que hacer componentes (programas) nuevos.

La reutilización del software pretende aprovechar programas realizados para una aplicación A para otra B, pero esta concepción está muy lejos de tener colecciones genéricas de componentes.

- P15.2.** En las ingenierías tradicionales los productos fabricados pueden tener un determinado margen de tolerancia. Por ejemplo, en un aparato de aire acondicionado doméstico se admite que la temperatura obtenida tenga un determinado rango de error con respecto a la temperatura de consigna. Indicar, razonando la respuesta, si esta tolerancia es admisible en la ingeniería del software.

## SOLUCIÓN

En la ingeniería del software no se admiten márgenes de tolerancia: el programa funciona correctamente o no. Los resultados deben ser los buscados en todo caso: en una aplicación para obtener las nóminas de los empleados de una empresa no se admite ningún margen de tolerancia para las cantidades que debe cobrar cada uno de ellos.

- P15.3.** Enumerar las diferencias que existe entre la ingeniería del software y otras ramas tradicionales de la ingeniería.

## SOLUCIÓN

En ingeniería del software:

- Los sistemas no se construyen con componentes genéricos prefabricados.
- No se admiten márgenes de tolerancia en los resultados, éstos siempre deben ser exactos.
- Las medidas de prestaciones y de calidad suelen ser más complejas.
- El proceso de producción es normalmente mucho más económico y sencillo que en otras ramas de la ingeniería: así, una vez desarrollado un prototipo, es más fácil obtener copias de programas y de sus manuales (fabricar software) que construir coches (fabricar coches).

## PLANIFICACIÓN Y GESTIÓN DE PROYECTOS

- P15.4.** Indique los inconvenientes de utilizar el número de líneas de un programa como medida de su complejidad.

## SOLUCIÓN

- Existen muchos tipos de instrucciones, con complejidad muy diversa, con lo que un programa puede tener pocas instrucciones y muy complejas y otro puede tener muchas de poca complejidad (con muchas declaraciones, por ejemplo), pudiendo, por tanto, ser el primero más complejo que el segundo.
- Dado un problema, la complejidad del algoritmo que lo resuelve y la de su codificación en un lenguaje puede depender notablemente de la habilidad o experiencia del programador.
- El número de líneas de un programa depende del lenguaje de programación utilizado.

No obstante lo anterior, el número de líneas estimado para un programa da una idea aproximada para realizar una planificación y presupuesto iniciales.

## ANÁLISIS

**P15.5.** Suponga la aplicación de adquisición de entradas de espectáculos a través de Internet (Ejemplo 15.6), incluyendo los procesos de entrada y salida en el servidor web. Realice un análisis del sistema identificando los posibles procedimientos o módulos a desarrollar.

## SOLUCIÓN

Podrían considerarse los siguientes procesos:

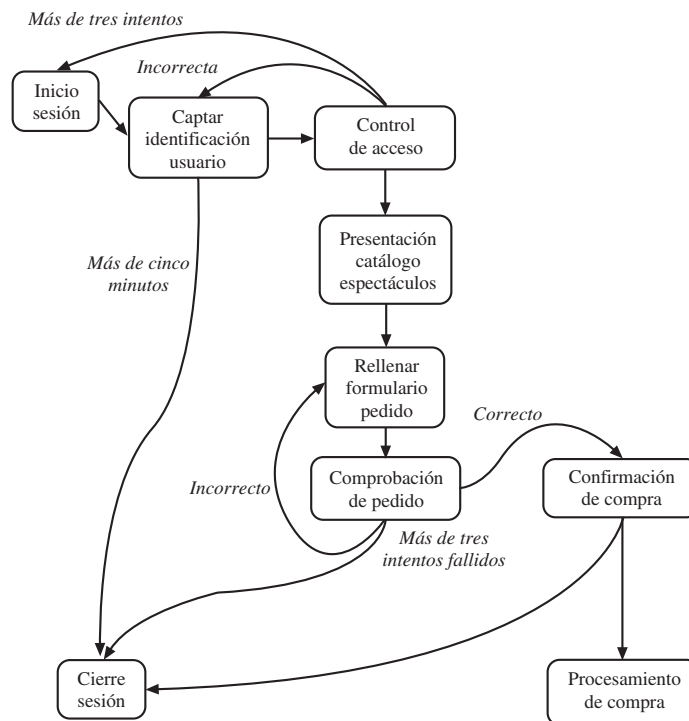
- Supervisor del servidor web.
- Inicio de sesión.
- Captación de nombre de usuario y contraseña.
- Control de acceso.
- Conexión con la aplicación de reserva de espectáculos.
- Supervisor del navegador a través del catálogo de espectáculos.
- Proceso de selección de espectáculo y rellenado de formulario de pedido (con detección de errores de formato).
- Proceso de recepción y comprobación de datos del formulario.
- Proceso de confirmación de compra.
- Procesamiento de reserva y compra.
- Proceso de cierre de sesión.

**P15.6.** Suponga la aplicación de adquisición de entradas de espectáculos a través de Internet (Ejemplo 15.6), incluyendo los procesos de entrada y salida en el servidor web. Esbozar, para este sistema:

- Un diagrama de estados.
- Una carta de estructura.
- Un diagrama de flujo de datos.

## SOLUCIÓN

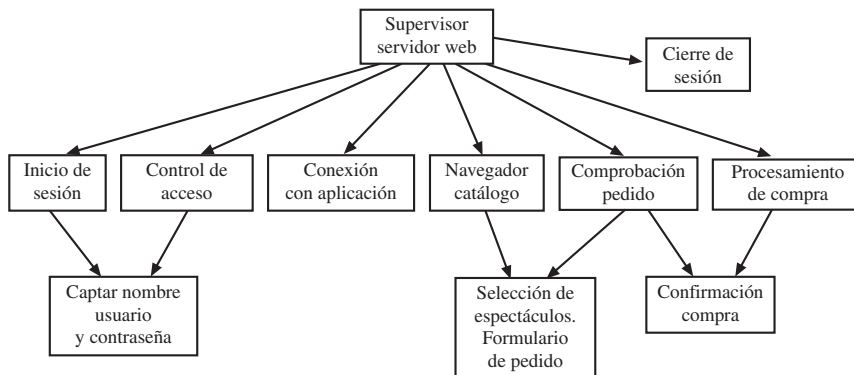
- Un diagrama de estados (véase Figura 15.11).



**Figura 15.11.** Diagrama de estados de la aplicación de reserva y compra de entradas de espectáculos.

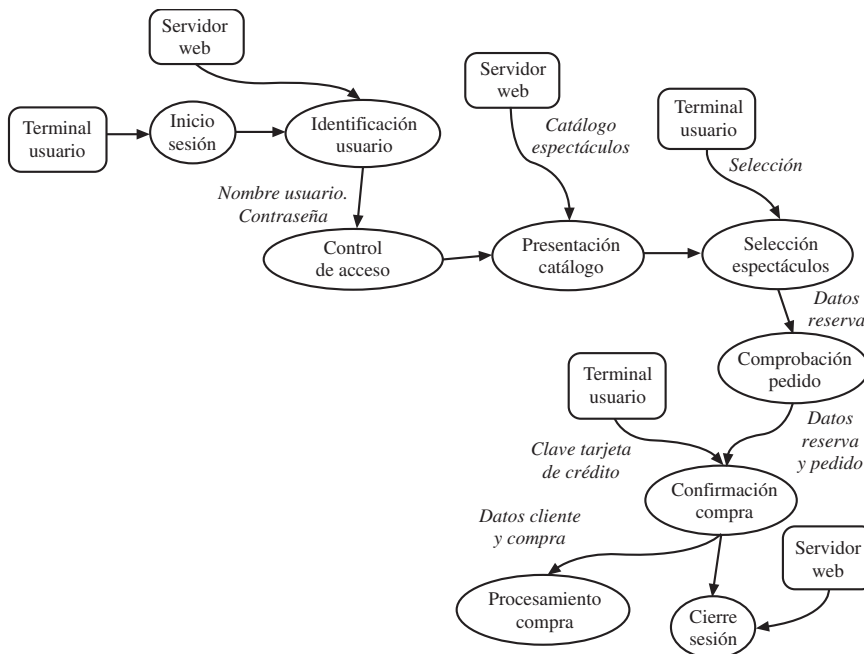


b) Una carta de estructura (véase Figura 15.12).



**Figura 15.12.** Carta de estructura de la aplicación de reserva y compra de entradas de espectáculos.

c) Un diagrama de flujo de datos (véase Figura 15.13).



**Figura 15.13.** Diagrama de flujo de datos de la aplicación de reserva y compra de entradas de espectáculos.

**P15.7.** Considere la aplicación de adquisición de entradas de espectáculos a través de Internet (Ejemplo 15.6), incluyendo los procesos de entrada y salida en el servidor web. Para este sistema:

- Identificar los tipos de objetos.
- Realizar un diagrama de clases.

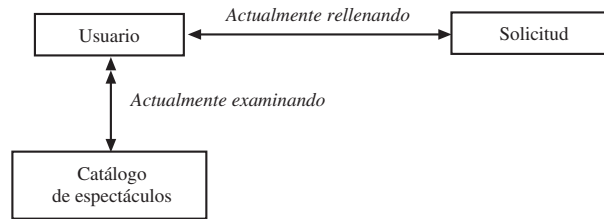
#### SOLUCIÓN

- Identificar los tipos de objetos.

- Cliente.
- Catálogo de espectáculos.
- Pedido de entradas.



b) Realizar un diagrama de clases (véase Figura 15.14).



**Figura 15.14.** Diagrama de clases de la aplicación de reserva y compra de entradas de espectáculos.

## DISEÑO. MODULARIDAD

**P15.8.** Un programa llama a una función que calcula el factorial de un número. ¿Qué tipos de acoplamientos hay entre el programa y la función?

### SOLUCIÓN

Hay acoplamiento de control, ya que el programa pasa el control (“llama”) a la función, y acoplamiento de datos, ya que el valor obtenido por la función factorial es pasado al programa que la llama.

**P15.9.** Un programa llama a un subprograma que se limita a imprimir el siguiente mensaje: “Le falta papel a la impresora; si quiere que continúe imprimiendo, por favor, rellene la bandeja”. ¿Qué tipos de acoplamiento hay entre programa y subprograma?

**P15.10.** En un programa hay dos módulos que se relacionan sólo con una variable, y otros dos módulos que se relacionan mediante dos variables. ¿En cuál de los dos grupos de módulos hay menor acoplamiento? ¿Qué situación es la más deseable?

### SOLUCIÓN

Entre los módulos que se relacionan con una sola variable hay menor acoplamiento. Esta es la situación más deseable, ya que siempre hay que procurar que el acoplamiento sea mínimo, es decir, que dependan lo menos posible unos módulos de otros.

**P15.11.** En un programa hay un módulo en el que internamente dos funciones se relacionan mediante una sola variable. ¿Qué se puede decir de la cohesión del módulo?

### SOLUCIÓN

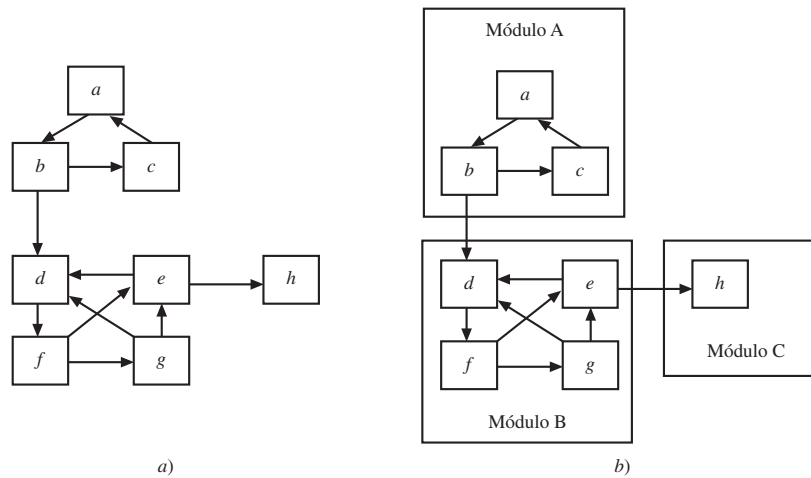
La cohesión es muy baja y en este caso interesaría descomponer ese módulo en dos, ya que siempre hay que procurar que la cohesión dentro de un módulo sea lo mayor posible.

**P15.12.** En la Figura 15.15a) se muestra una carta de estructura, en la que cada rectángulo representa una tarea muy sencilla. ¿Cómo deberían agruparse las distintas tareas formando módulos para conseguir un acoplamiento y cohesión adecuados?

### SOLUCIÓN

Podemos analizar sólo el acoplamiento de control, ya que el enunciado del problema no especifica la interacción de datos entre tareas. Las tareas *a*, *b* y *c* están muy cohesionadas entre ellas y mínimamente con las demás, con lo que debería formar un módulo. Lo mismo ocurre con las tareas *d*, *e*, *f* y *g*, que formarían otro módulo. Por otra parte *h* sólo está acoplada con el segundo módulo, luego podría constituir un tercer módulo, tal y como se indica en la Figura 15.15b).

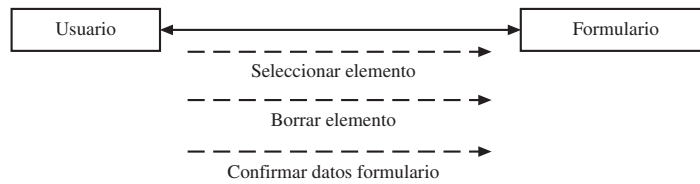
**P15.13.** Suponga la aplicación de obtención por medio de un formulario de datos de un usuario a través de Internet (Ejemplo 15.3), incluyendo los procesos de entrada y salida en el servidor web. Esbozar un diagrama de colaboración para analizar el acoplamiento de datos entre objetos.



**Figura 15.15.** Ejemplo: a) de relaciones entre tareas y b) agrupación de tareas en módulos.

#### SOLUCIÓN

Véase Figura 15.16.



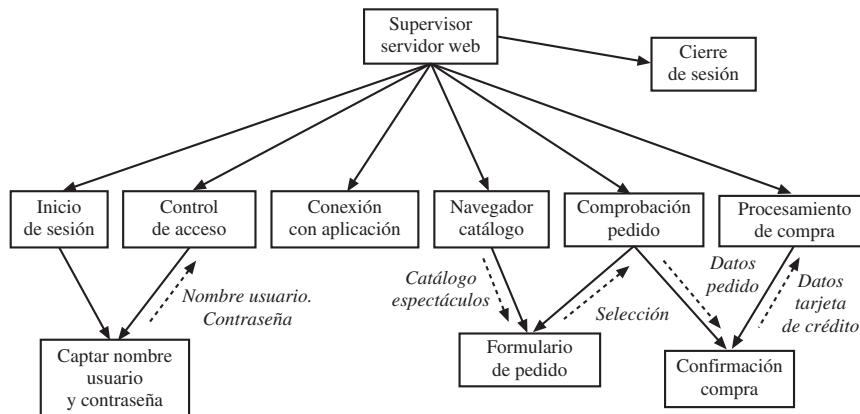
**Figura 15.16.** Diagrama de colaboración para la aplicación de obtención de formulario a través de Internet.

**P15.14.** Considere la aplicación de adquisición de entradas de espectáculos a través de Internet (Ejemplo 15.6), incluyendo los procesos de entrada y salida en el servidor web. Con objeto de estudiar el acoplamiento de datos entre módulos, esbozar:

- Una carta de estructura, incluyendo los intercambios de datos.
- Un diagrama de colaboración.

#### SOLUCIÓN

Las soluciones se dan en las Figuras 15.17 y 15.18.



**Figura 15.17.** Carta de estructura de la aplicación de reserva y compra de entradas de espectáculos mostrando el acoplamiento de datos.

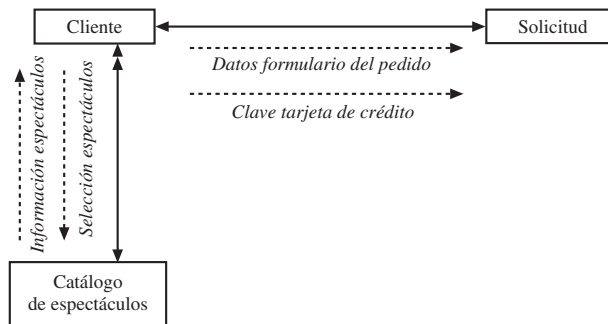


Figura 15.18. Diagrama de colaboración de la aplicación de reserva y compra de entradas de espectáculos.

## CALIDAD

**P15.15.** Una empresa A encarga un programa a una empresa B. Al cabo de un tiempo una empresa C encarga un programa a la empresa B. El ingeniero de la empresa B se da cuenta de que puede utilizar alguno de los módulos del programa que hizo para la empresa A. ¿De qué principio de calidad está haciendo uso?

### SOLUCIÓN

Está reutilizando el software, que es un factor relacionado con la compatibilidad.



## Problemas propuestos

### CONCEPTOS BÁSICOS

**P15.16.** Indicar, razonando la respuesta, las diferencias existentes en el proceso de producción de un paquete software comercializado con el proceso de producción aplicado en otras ingenierías que producen sistemas físicos.

### CICLO DE VIDA DEL SOFTWARE

**P15.17.** Cuando se está usando el modelo de ciclo de vida de prototipos, ¿tiene sentido dejar el prototipo al cliente como producto utilizable? Razonar sobre las ventajas e inconvenientes de utilizar prototipos.

### ETAPAS Y MODELOS DE DESARROLLO DEL SOFTWARE

**P15.18.** Indique, desde un punto de vista económico, los motivos por los que se requiere que una aplicación software esté muy bien documentada.

### ANÁLISIS

**P15.19.** Durante la fase de análisis de una aplicación informática se acuerdan con el cliente, entre otros, los siguientes requisitos:

- La aplicación admitirá hasta como máximo 300.000 artículos, y a todos ellos se les asignará un código (con el número de dígitos menor posible); los precios de compra y de venta de cada artículo en ningún caso serán superiores a 1.000,00 €.
- La aplicación admitirá hasta como máximo 1.000.000 de clientes, y a todos ellos se les asignará un código alfanumérico.
- Los clientes podrán acceder a un subesquema de la base de datos de artículos, pero proporcionando el número de DNI y el código de cliente.

Traducir estos requisitos a especificaciones.

**P15.20.** Suponga que se desea desarrollar una **aplicación de e-comercio**. En resumen, se dispone de un servidor web con una base de datos *catálogo* que contiene todos los productos en venta. El cliente se conecta al servidor a través de Internet (dando su nombre de usuario y contraseña), examina el catálogo y la descripción de los productos que va seleccionando deben ir apareciéndole en un *formulario\_compra*. Una vez que finaliza la selección de productos debe accionar un botón de *compra*, lo que debe provocar el envío de un mensaje al servidor con el contenido del *formulario\_compra*. A continuación el servidor debe presentar un *formulario\_liquidación* indicando el importe total de la compra, y solicitando el número de tarjeta de crédito y la contraseña del cliente. Una vez rellenados estos dos apartados se debe transmitir al servidor

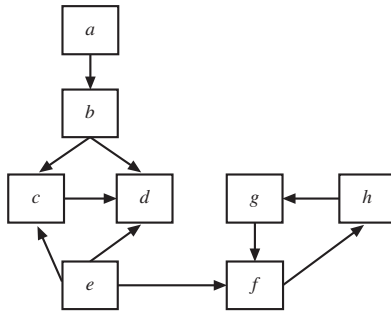
un nuevo mensaje con sus contenidos. Por último, el servidor debe hacer aparecer en la pantalla del cliente un tercer formulario, en esta ocasión de *confirmación\_pedido*, que una vez aceptado por el cliente debe ser devuelto al servidor. Concluida la compra el servidor debe abrir una ventana para que el usuario pueda desconectarse. Realice un análisis del sistema identificando los posibles procedimientos o módulos a desarrollar.

**P15.21.** Suponga la aplicación de e-comercio descrita en el Problema 15.11. Esbozar, para este sistema:

- a) Un diagrama de estados.
- b) Un diagrama de flujo de datos.

### DISEÑO

**P15.22.** En la Figura 15.19 se muestra una carta de estructura, en la que cada rectángulo representa una tarea muy sencilla.



**Figura 15.19.** Ejemplo de relaciones entre tareas.

lla. ¿Cómo deberían agruparse las distintas tareas formando módulos para conseguir unas adecuados acoplamiento y cohesión?

**P15.23.** Para la aplicación de e-comercio descrita en el Problema 15.11 realice una carta de estructura.

**P15.24.** Suponga la aplicación de e-comercio descrita en el Problema 15.11. Para este sistema:

- a) Identificar los tipos de objetos.
- b) Realizar un diagrama de clases.

**P15.25.** Suponga la aplicación de e-comercio descrita en el Problema 15.11. Con objeto de estudiar el acoplamiento de datos entre módulos, esbozar:

- a) Una carta de estructura, incluyendo los intercambios de datos.
- b) Un diagrama de colaboración.

### REPASO GENERAL

**P15.26.** Indicar para qué se utilizan y la información que contienen las siguientes herramientas utilizadas en ingeniería del software:

- a) Diagrama de Gantt.
- b) Tabla de decisión.
- c) Diagrama de estado.
- d) Diagrama de flujo de datos (DFD).
- e) Carta de estructura.
- f) Diagrama de clases.
- g) Diagrama de colaboración.



# Sistemas de numeración usuales en informática

Los computadores suelen efectuar las operaciones aritméticas utilizando una representación para los datos numéricos basada en el **sistema de numeración base dos** (que, por abreviar, denominaremos **binario natural**, o **binario**, sin más).

También se utilizan los sistemas de numeración octal y hexadecimal, para obtener códigos intermedios. Un número expresado en uno de estos dos códigos puede transformarse directa y fácilmente a binario, y viceversa. Con ellos se simplifica la transcripción de números binarios y se está más próximo al sistema que utilizamos usualmente (el sistema decimal), por lo que con gran frecuencia se utilizan como paso intermedio en las transformaciones de decimal a binario, y viceversa. Además la realización electrónica de codificadores/decodificadores entre binario y un código intermedio es mucho más simple que entre binario y decimal.

El objetivo de este apéndice es resumir los aspectos prácticos de mayor interés de los sistemas de numeración desde el punto de vista de su uso en los computadores.

## A1.1. Representación posicional de los números

Un **sistema de numeración en base  $b$**  utiliza para representar los números un alfabeto compuesto por  $b$  símbolos o cifras. Así, todo número se expresa mediante un conjunto de cifras, contribuyendo cada una de ellas con un valor que depende de:

- a) La cifra en sí.
- b) La posición que ocupe dentro del número.

En el **sistema de numeración decimal** (o **sistema en base 10**), que es el que habitualmente se utiliza,  $b = 10$  y el alfabeto está constituido por diez símbolos, denominados también cifras decimales:

$$S_{10} = \{0,1,2,3,4,5,6,7,8,9\} \quad [A1.1]$$

### EJEMPLO A1.1

El número decimal 3.278,52 puede obtenerse como suma de:

$$\begin{array}{r}
 3000 \\
 200 \\
 70 \\
 + \quad 8 \\
 0,5 \\
 0,02 \\
 \hline
 3.278,52
 \end{array}$$

es decir, se verifica:

$$3.278,52 = 3 \cdot 10^3 + 2 \cdot 10^2 + 7 \cdot 10^1 + 8 \cdot 10^0 + 5 \cdot 10^{-1} + 2 \cdot 10^{-2}$$

Cada posición tiene un nombre y peso específicos:

posición 0 peso $b^0 = 1$ , unidades	(en el ejemplo: 8)
posición 1 peso $b^1 = 10$ , decenas	(en el ejemplo: 7)
posición 2 peso $b^2 = 100$ , centenas	(en el ejemplo: 2)
posición 3 peso $b^3 = 1.000$ , millares	(en el ejemplo: 3)
etc.	
posición $-1$ peso $b^{-1} = 0,5$ ; décimas	(en el ejemplo: 5)
posición $-2$ peso $b^{-2} = 0,225$ ; centésimas	(en el ejemplo: 2)
etc.	

Generalizando para cualquier base, se tiene que la representación de un número  $N$  en la base  $b$ :

$$N \equiv \dots n_4 n_3 n_2 n_1 n_0 n_{-1} n_{-2} n_{-3} n_{-4} \quad [\text{A1.2}]$$

es una forma abreviada de expresar su valor, que es:

$$N \equiv \dots n_4 \cdot b^4 + n_3 \cdot b^3 + n_2 \cdot b^2 + n_1 \cdot b^1 + n_0 \cdot b^0 + n_{-1} \cdot b^{-1} + n_{-2} \cdot b^{-2} + n_{-3} \cdot b^{-3} + n_{-4} \cdot b^{-4} + \dots [\text{A1.3}]$$

Para representar una cantidad, por un lado resulta más cómodo que el número de símbolos (cifras) del alfabeto (la base) sea lo menor posible, pero, por otra parte, cuanto menor es la base, mayor es el número de cifras que se necesitan para representar una cantidad dada.

En la Tabla A1.1 se definen los sistemas de numeración usuales en informática; éstos son: el decimal, binario, octal y hexadecimal. Los dos últimos sistemas se conocen con el nombre de códigos intermedios, ya que con mucha frecuencia no se transforma un número directamente de decimal a binario, sino que se hace a través de octal o hexadecimal, como se verá en la Sección A1.4.

Denominación	Base	Símbolos utilizados
Decimal	$b = 10$	$S_{10} = \{0,1,2,3,4,5,6,7,8,9\}$
Binario	$b = 2$	$S_2 = \{0,1\}$
Octal	$b = 8$	$S_8 = \{0,1,2,3,4,5,6,7\}$
Hexadecimal	$b = 16$	$S_{16} = \{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F\}$

**Tabla A1.1.** Sistemas de numeración usuales en informática.

## A1.2. Operaciones aritméticas con variables binarias

Las operaciones aritméticas básicas son la suma, resta, multiplicación y división. Al ser la representación en binario natural una notación ponderada, es decir, verificar la expresión [A1.3], estas operaciones son análogas a las realizadas en decimal pero hay que utilizar para cada posición las Ta-

blas A1.2, A1.3, A1.4 y A1.5, según la operación aritmética a realizar. A continuación se incluyen unos ejemplos.

$a$	$b$	$a + b$
0	0	0
0	1	1
1	0	1
1	1	0 y me llevo 1

**Tabla A1.2.** Suma aritmética con variables binarias.

$a$	$b$	$a - b$
0	0	0
0	1	1 y adeudo 1
1	0	1
1	1	0

**Tabla A1.3.** Resta aritmética con variables binarias.

$a$	$b$	$a \cdot b$
0	0	0
0	1	0
1	0	0
1	1	1

**Tabla A1.4.** Multiplicación aritmética con variables binarias.

$a$	$b$	$a/b$
0	0	Indeterminado
0	1	0
1	0	$\infty$
1	1	1

**Tabla A1.5.** División aritmética con variables binarias.

### EJEMPLO A1.2

Efectuar las siguientes operaciones aritméticas binarias:

$$\begin{array}{r}
 1110101 + 1110110; 1101010 - 010111; \\
 1101010 \times 11; 1010011 \times 10 \text{ y } 1101,01/101
 \end{array}$$

$$\begin{array}{r}
 + \quad 1110101 \\
 \quad 1110110 \\
 \hline
 11101011
 \end{array}
 \quad
 \begin{array}{r}
 - \quad 1101010 \\
 \quad 1010111 \\
 \hline
 0010011
 \end{array}
 \quad
 \begin{array}{r}
 \times \quad 1101010 \\
 \quad \quad 11 \\
 \hline
 1101010 \\
 1101010 \\
 \hline
 100111110
 \end{array}
 \quad
 \begin{array}{r}
 \times \quad 1010011 \\
 \quad \quad 10 \\
 \hline
 0000000 \\
 1010011 \\
 \hline
 10100110
 \end{array}$$

$$\begin{array}{r}
 1101,01 \\
 \hline
 101
 \end{array}
 =
 \begin{array}{r}
 110101 \\
 \hline
 10100
 \end{array}$$

$$\begin{array}{r}
 110101 \quad | \quad 10100 \\
 - 10100 \quad 10,101... \\
 \hline
 0011010 \\
 - 10100 \\
 \hline
 0011000 \\
 - 10100 \\
 \hline
 00010
 \end{array}$$

Se observa que multiplicar por  $10_2$  (es decir, por 2 en decimal) equivale a añadir un cero a la derecha, siendo esto similar a multiplicar por  $10_{10}$  un número decimal. De la misma forma dividir por  $2_{10} = 10_2$  se hace desplazando el punto decimal a la izquierda, o eliminando ceros a la derecha.

### EJEMPLO A1.3

Se verifican las siguientes igualdades

$$\begin{aligned}
 10101,0101_2 \cdot 2 &= 101010,101_2 \\
 1101,1010_2 \cdot 2^5 &= 110110100_2 \\
 1010100_2 / 2 &= 101010_2 \\
 10101,101_2 / 2^{-6} &= 0,010101101_2
 \end{aligned}$$



### A1.3. Representación en complementos

Para representar un número negativo se puede utilizar el complemento de ese número a la base o a la base menos uno del sistema de numeración utilizado. De esta forma, como se verá más adelante, las sumas y restas quedan reducidas a sumas, independientemente de los signos de los operandos. Este sistema de representación es de sumo interés en el caso de los computadores, ya que al utilizarlo se reduce la complejidad de los circuitos de la unidad aritmético-lógica (no son necesarios circuitos específicos para restar).

El **complemento a uno** de un número binario,  $N$ , es el número que resulta de sustituir unos por ceros y ceros por unos en el número original.

Podemos **restar dos números binarios** sumando al minuendo el complemento a uno del sustraendo. La cifra que se arrastra del resultado se descarta y se suma al resultado así obtenido.

#### EJEMPLO A1.4

El complemento a uno del número 10010 es 01101.

El complemento a uno de 101010 es 010101.

El resultado de restar  $1000111 - 10010$  es:

$$\begin{array}{rcl}
 \begin{array}{r}
 1000111 \\
 - 0010010 \\
 \hline
 0110101
 \end{array} & \text{o bien, como el complemento a 1 de} & \begin{array}{r}
 1000111 \\
 + 1101101 \\
 \hline
 (1)0110100 \\
 + \quad \quad \quad 1 \\
 \hline
 0110101
 \end{array} \\
 & 0010010 \text{ es } 1101101 &
 \end{array}$$

El **complemento a dos** de un número binario,  $N$ , es el número que resulta de sumar uno al complemento a uno de dicho número.

Se pueden restar dos números binarios sumando al minuendo el complemento a dos del sustraendo y despreciando, en su caso, el acarreo del resultado.

#### EJEMPLO A1.5

El complemento a 2 del número 10010 es 01110; en efecto:

$$C_2(10010) = C_1(10010) + 1 = 01101 + 1 = 01110$$

El complemento a 2 del número 101010 es 010110; en efecto:

$$C_2(101010) = C_1(101010) + 1 = 010101 + 1 = 010110$$

El resultado de las restas:  $11001 - 10010$  y  $110000 - 101010$  son:

$$\begin{array}{r}
 11001 \\
 - 10010 \\
 \hline
 00111
 \end{array}
 \qquad
 \begin{array}{r}
 110000 \\
 - 101010 \\
 \hline
 001110
 \end{array}$$

O sumando al minuendo el complemento a dos del sustraendo:

$$\begin{array}{r}
 11001 \\
 + 01110 \\
 \hline
 \cancel{1}\cancel{1}00111
 \end{array}
 \qquad
 \begin{array}{r}
 110000 \\
 + 010110 \\
 \hline
 \cancel{1}\cancel{1}001110
 \end{array}$$

## A1.4. Transformaciones entre bases distintas

Como se verá a continuación, las transformaciones de base decimal a binaria, y viceversa, son relativamente complejas y no se pueden hacer directamente en paralelo en un solo paso, lo cual quiere decir que la realización de estas conversiones tanto por medio de un programa como por circuitos específicos son lentas. No ocurre lo mismo con las conversiones entre octal y binario y entre hexadecimal y binario, por lo que con mucha frecuencia se utilizan estos sistemas de numeración. Es más, con frecuencia para pasar de decimal a binario, y viceversa, se utiliza como paso intermedio la base octal o la hexadecimal, en la forma que se indica a continuación:

decimal  $\rightarrow$  hexadecimal  $\rightarrow$  binario  
 binario  $\rightarrow$  hexadecimal  $\rightarrow$  decimal  
  
 decimal  $\rightarrow$  octal  $\rightarrow$  binario  
 binario  $\rightarrow$  octal  $\rightarrow$  decimal

El código octal se suele utilizar cuando el número de bits a representar es múltiplo de 3, y el hexadecimal cuando dicho número es múltiplo de 4.

### A1.4.1. TRANSFORMACIONES CON LA BASE DECIMAL

Podemos transformar un número binario a decimal sin más que tener en cuenta las expresiones [A1.2] y [A1.3], en las que en este caso será  $b = 2$ , según se pone de manifiesto en los siguientes ejemplos:

#### EJEMPLO A1.6

Transformar a decimal los siguientes números binarios:

110100;    0,10100;    10100,001

SOLUCIÓN

$$\begin{aligned} 110100)_2 &= (1 \cdot 2^5) + (1 \cdot 2^4) + (1 \cdot 2^2) = 2^5 + 2^4 + 2^2 = 32 + 16 + 4 = 52)_{10} \\ 0,10100)_2 &= 2^{-1} + 2^{-3} = (1/2) + (1/8) = 0,625)_{10} \\ 10100,001)_2 &= 2^4 + 2^2 + 2^{-3} = 16 + 4 + (1/8) = 20,125)_{10} \end{aligned}$$

Observando el Ejemplo A1.6 se deduce que podemos transformar de binario a decimal sencillamente sumando los pasos de las posiciones en las que hay un 1, como se muestra en el siguiente ejemplo.

#### EJEMPLO A1.7

Obtener los valores decimales correspondientes a los números binarios 1001,001 y 1001101.

<i>Número binario</i>	<i>Equivalente decimal</i>
<u>1 0 0 1, 0 0 1</u>	$= 8 + 1 + 1/8 = 9,125)_{10}$
<i>pasos <math>\rightarrow</math></i>	8 4 2 1, - - 1/8

<i>Número binario</i>	<i>Equivalente decimal</i>
<u>1 0 0 1 1 0 1</u>	$= 64 + 8 + 4 + 1 = 77)_{10}$
<i>pasos <math>\rightarrow</math></i>	64 32 16 8 4 2 1

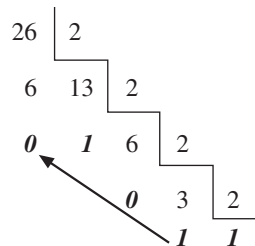
Analicemos ahora cómo se hace la transformación inversa; es decir, la conversión de un número decimal a binario:

- a) La parte entera del nuevo número (binario) se obtiene dividiendo por 2 (sin obtener decimales en el cociente) la parte entera del número decimal de partida, y los cocientes que sucesivamente se vayan obteniendo. Los residuos (restos) de estas divisiones y el último cociente (que serán siempre ceros o unos) son las cifras binarias. El último cociente será el **bit más significativo (MSB)** (*Most Significant Bit*) y el primer residuo será el **bit menos significativo (LSB)** (*Least Significant Bit*).

### EJEMPLO A1.8

Obtener el equivalente binario del número decimal 26.

$26)_{10}$  en binario es  $26)_{10} = 1\ 1\ 0\ 1\ 0)_2$ , ya que:



Con lo que  $26)_{10} = 1\ 1\ 0\ 1\ 0)_2$ .

- b) La parte fraccionaria del número binario se obtiene multiplicando por 2 sucesivamente la parte fraccionaria del número decimal de partida y las partes fraccionarias que se van obteniendo en los productos sucesivos. El número binario se forma con las partes enteras (que serán ceros o unos) de los productos obtenidos, como se hace en el siguiente ejemplo.

### EJEMPLO A1.9

Transformar a binario natural los números decimales 0,1875 y 74,1875.

$$\begin{array}{cccc} \times \frac{0,1875}{2} & \times \frac{0,375}{2} & \times \frac{0,7500}{2} & \times \frac{0,5000}{2} \\ \hline 0,37500 & 0,7500 & 1,5000 & 1,0000 \end{array}$$

Luego:  $0,1875)_{10} = 0,0011)_2$

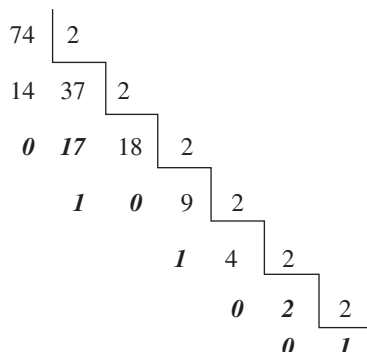
En consecuencia, el número decimal 26,1875 en binario es:

$$26,1875)_{10} = 11010,0011)_2$$

### EJEMPLO A1.10

Transformar a binario el número decimal 74,423.

a) Parte entera:



b) Arte fraccionaria:

$$\begin{array}{r}
 0,423 \\
 \times \quad 2 \\
 \hline
 0,846
 \end{array}
 \quad
 \begin{array}{r}
 0,846 \\
 \times \quad 2 \\
 \hline
 1,692
 \end{array}
 \quad
 \begin{array}{r}
 0,692 \\
 \times \quad 2 \\
 \hline
 1,384
 \end{array}
 \quad
 \begin{array}{r}
 0,384 \\
 \times \quad 2 \\
 \hline
 0,768
 \end{array}
 \quad
 \begin{array}{r}
 0,768 \\
 \times \quad 2 \\
 \hline
 1,536
 \end{array}$$

Es decir:

$$74,423_{10} = 1001010,01101..._2$$

En el último ejercicio del Ejemplo A1.10 se observa que un número decimal con cifras fraccionarias puede dar lugar a un número binario con un número de cifras fraccionarias mucho mayor o incluso infinito. Si el número binario se almacena con un número prefijado de bits se producirá en la representación binaria un error de truncamiento.

## A1.4.2. TRANSFORMACIONES CON LA BASE OCTAL

Para facilitar la comprensión de este apartado, en la Tabla A1.6 incluimos los 8 primeras cifras decimales, que coinciden con las cifras octales, y sus correspondientes valores binarios.

Número decimal	Número binario
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

**Tabla A1.6.** Números decimales del 0 al 7 y su correspondiente binario.

Un número octal puede pasarse a binario aplicando algoritmos análogos a los vistos en la Sección A1.4.1. No obstante, al ser  $b = 8 = 2^3$ , puede hacerse la conversión fácilmente en la forma que se indica a continuación.

Para **transformar un número binario a octal** se forman grupos de tres cifras binarias a partir del punto decimal hacia la izquierda y hacia la derecha. Posteriormente se efectúa directamente la conversión a octal de cada grupo individual.

Como en la práctica sólo es necesario hacer la conversión a octal (o decimal) de cada uno de los grupos de tres cifras, basta con memorizar la Tabla A1.6 para poder realizar rápidamente la conversión.

**EJEMPLO A1.11**

$$\begin{array}{l} \text{Número binario} \rightarrow 10 \quad 001 \quad 101 \quad 100 \quad , \quad 110 \quad 10)_2 = 2.154,64)_8 \\ \text{Número octal} \rightarrow 2 \quad 1 \quad 5 \quad 4 \quad , \quad 6 \quad 4 \end{array}$$

$$\begin{array}{l} \text{Número binario} \rightarrow 01 \quad 101)_2 = 15)_8 \\ \text{Número octal} \rightarrow 1 \quad 5 \end{array}$$

De octal a binario se pasa sin más que convertir individualmente a binario (tres bits) cada cifra octal, manteniendo el orden del número original.

**EJEMPLO A1.12**

$$\begin{array}{l} 537,24)_8 = 101 \quad 011 \quad 111 \quad , \quad 010 \quad 110 = 10101111,01)_2 \\ \text{cifras octales} \rightarrow 5 \quad 3 \quad 7 \quad 2 \quad 6 \end{array}$$

$$\begin{array}{l} 175,22)_8 = 001 \quad 111 \quad 101 \quad , \quad 010 \quad 010 = 1111101,0101)_2 \\ \text{cifras octales} \rightarrow 1 \quad 7 \quad 5 \quad 2 \quad 2 \end{array}$$

Para **transformar un número de octal a decimal** se aplica la expresión [A1.3] con  $b = 8$ . Para **transformar un número entero de decimal a octal** se hacen sucesivas divisiones enteras del número y los subsiguientes cocientes entre ocho (tal como en binario se hacía entre 2). Para transformar la parte fraccionaria de un número decimal a octal se hacen sucesivas multiplicaciones por ocho (como para pasar a binario se hacía por 2).

**EJEMPLO A1.13**

Pasar a decimal el número octal  $1367,25)_8$  y a octal en número decimal  $760,33$ .

$$1367,25)_8 = 759,328125)_{10} \quad ; \quad \text{en efecto:}$$

$$\begin{aligned} 1367,25)_8 &= 1 \cdot 8^3 + 3 \cdot 8^2 + 6 \cdot 8^1 + 7 \cdot 8^0 + 2 \cdot 8^{-1} + 5 \cdot 8^{-2} = \\ &= 512 + 192 + 48 + 7 + 0,25 + 0,078125 = 759,328125)_{10} \end{aligned}$$

$$760,33)_{10} = 1370,2507)_8 \quad ; \quad \text{en efecto:}$$

Parte entera:

$$\begin{array}{r} 760 \overline{) 8} \\ 40 \quad 95 \overline{) 8} \\ 0 \quad 15 \quad 11 \overline{) 8} \\ 7 \quad 3 \quad 1 \end{array}$$

Parte fraccionaria:

$$\begin{array}{r} \times \quad 0,33 \\ \times \quad 8 \\ \hline 2,64 \end{array} \quad \begin{array}{r} \times \quad 0,64 \\ \times \quad 8 \\ \hline 5,12 \end{array} \quad \begin{array}{r} \times \quad 0,12 \\ \times \quad 8 \\ \hline 0,96 \end{array} \quad \begin{array}{r} \times \quad 0,96 \dots \\ \times \quad 8 \\ \hline 7,68 \dots \end{array}$$

### A1.4.3. TRANSFORMACIONES CON LA BASE HEXADECIMAL

Los valores binarios y decimales correspondientes a las cifras hexadecimales se muestran en la Tabla A1.7.

Cifra hexadecimal	Valor decimal	Número binario
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

**Tabla A1.7.** Cifras hexadecimales y sus correspondientes valores decimal y binario.

Al ser  $b = 16 = 2^4$ , podemos hacer las conversiones de binario a hexadecimal, y viceversa, en forma análoga al sistema octal. Ahora bien, aquí utilizaremos grupos de 4 bits en lugar de grupos de 3 bits.

#### EJEMPLO A1.14

Se verifica:

$$10\ 0101\ 1101\ 1111,\ 1011\ 101)_2 = 25DDF,BA)_H$$

ya que:

$$\begin{aligned} \text{Número binario} &\rightarrow 0010\ 0101\ 1101\ 1111,\ 1011\ 1010)_2 = 25DBF,BA)_{16} \\ \text{Número hexadecimal} &\rightarrow 2\ 5\ D\ F,\ B\ A \end{aligned}$$

También:

$$1ABC70,C4)_H = 1\ 1010\ 1011\ 1100\ 0111\ 0000,\ 1100\ 01)_2$$

En efecto:

$$\begin{aligned} \text{Número hexadecimal} &\rightarrow 1\ A\ B\ C\ 7\ 0,\ C\ 4 \\ \text{Número binario} &\rightarrow 0001\ 1010\ 1011\ 1100\ 0111\ 0000,\ 1100\ 0100 \end{aligned}$$

De la misma forma que manualmente es muy fácil convertir números de binario a octal, y viceversa, y de binario a hexadecimal, y viceversa, también resulta sencillo efectuar esta operación elec-

trónicamente o por programa, por lo que a veces el computador utiliza este tipo de notaciones intermedias internamente o como entrada/salida.

Para transformar un número de hexadecimal a decimal se aplica la expresión [A1.3] con  $b = 16$ . Para pasar un número de decimal a hexadecimal se hace de forma análoga a los casos binario y octal: la parte entera se divide entre 16, así como los sucesivos cocientes enteros, y la parte fraccionaria se multiplica por 16, así como las partes fraccionarias de los sucesivos productos.

### EJEMPLO A1.15

Obtener el equivalente decimal del número hexadecimal  $A798C,1E)_H$ .

(Consultar la Tabla A1.7 para ver la correspondencia entre símbolos hexadecimales y valores decimales).

$$\begin{aligned} A798C,1E)_H &= (10) \cdot 16^4 + 7 \cdot 16^3 + 9 \cdot 16^2 + 8 \cdot 16 + 12 + 16^{-1} + (14) \cdot 16^{-2} = \\ &= 655360 + 28672 + 2304 + 128 + 12 + 0,0625 + 0,0546875 = \\ &= 686476.1171)_{10} \end{aligned}$$

Obtener el equivalente hexadecimal del número decimal 4373,79.

$$4373,79)_{10} \approx 11DD,CA3D)_H.$$

En efecto:

Parte entera:

$$\begin{array}{r} 4573 \quad | \quad 16 \\ 137 \quad 285 \quad | \quad 16 \\ 093 \quad 125 \quad | \quad 17 \quad | \quad 16 \\ \hline 13 \quad 13 \quad 1 \quad 1 \end{array}$$

Parte fraccionaria:

$$\begin{array}{cccc} 0,79 & 0,64 & 0,24 & 0,84 \dots \\ \times \frac{16}{16} & \times \frac{16}{16} & \times \frac{16}{16} & \times \frac{16}{16} \\ \hline 12,64 & 10,24 & 3,84 & 13,44 \dots \end{array}$$

# Principales códigos de Entrada/Salida

Conjunto de caracteres ASCII (ANSI-X3.4, 1968) (7 bits)

		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
00	0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
10	16	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
20	32	SP	!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/
30	48	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	64	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	80	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	-
60	96	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	112	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

Ejemplo: El carácter "k" es  $60 + B = 6B)_{16} = 110\ 1011)_2$ . Su valor decimal es  $96 + 11 = 107)_{10}$ .

## Caracteres de control ASCII:

Hex	Nombre	Descripción	Hex	Nombre	Descripción
00	NUL	Nulo	11	DC1	Control de dispositivo 1
01	SOH	Inicio de cabecera	12	DC2	Control de dispositivo 2
02	STX	Inicio de texto	13	DC3	Control de dispositivo 3
03	ETX	Final de texto	14	DC4	Control de dispositivo 24
04	EOT	Fin de transmisión	15	NAK	Acuse de recibo negativo
05	ENQ	Solicitud, petición	16	SYN	Sincronización
06	ACK	Acuse de recibo	17	ETB	Fin de bloque de transmisión
07	BEL	Pitido	18	CAN	Cancelar
08	BS	Retroceso de un espacio	19	EM	Final de soporte (de cinta, etc.)
09	HT	Tabulación horizontal	1A	SUB	Sustituir
0A	LF	Alimentación de línea	1B	ESC	Escape
0B	VT	Tabulación vertical	1C	FS	Separador de archivo
0C	FF	Alimentación de hoja	1D	GS	Separador de grupo
0D	CR	Retorno de carro (línea)	1E	RS	Separador de registro
0E	SO	Fuera de código	1F	US	Separador de unidad
0F	SI	Dentro de código	7F	DEL	Eliminar, borrar, deshabilitar
10	DLE	Escape de enlace de datos			



## Conjunto de caracteres ISO 8859-1 (Latín-1)

		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
00	0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
10	16	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
20	32	SP	!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/
30	48	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	64	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	80	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
60	96	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	112	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL
80	128																
90	144																
A0	160		¡	¢	£	€	¥	¦	§	¨	©	ª	«	¬	®	¯	
B0	176	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
C0	192	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D0	208	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E0	224	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F0	240	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Ejemplo: El carácter "k" es  $60 + B = 6B_{16} = 0110\ 1011_2$ . Su valor decimal es  $96 + 11 = 107_{10}$ .

## Conjunto de caracteres Unicode

Utiliza  $n = 16$  bits para representar 65.536 símbolos posibles, comprendidos en un rango que va de 0000 a FFFF.

## Rango Unicode Se corresponde con

0000 a 007F	Latín Básico (00 a 7F), definidos en la norma ASCII ANSI-X3.4.
0080 a 00FF	Suplemento Latín-1 (ISO 8859-1).
0100 a 017F	Ampliación A de Latín.
0180 a 024F	Ampliación B de Latín.
0250 a 02AF	Ampliación del Alfabeto Fonético Internacional (IPA).
02BF a 02FF	Espaciado de letras modificadoras.
0300 a 036F	Combinación de marcas diacríticas (tilde, acento grave, etc.).
0370 a 03FF	Griego.
0400 a 04FF	Cirílico.
0530 a 058F	Armenio.
0590 a 05FF	Hebreo.
0600 a 06FF	Árabe.
0700 a 074F	Sirio.
Etc.	Etc.

# Algunas medidas de uso común en informática

## Múltiplos de capacidad de información

Kilobyte (o KB)	$2^{10}$ Bytes = 1.024 Bytes $\approx$ 1.000 Bytes.
Megabyte (o MB)	$2^{20}$ Bytes = 1,048.576 Bytes $\approx$ 1,000.000 Bytes.
Gigabyte (o GB)	$2^{30}$ Bytes $\approx$ $10^9$ Bytes.
Terabyte (o TB)	$2^{40}$ Bytes $\approx$ $10^{12}$ Bytes.
Petabyte (o PB)	$2^{50}$ Bytes $\approx$ $10^{15}$ Bytes.
Exabyte (o EB)	$2^{60}$ Bytes $\approx$ $10^{18}$ Bytes.

**Nota:** Estos mismos múltiplos se pueden utilizar para palabras y bits. *Por ejemplo: 5 MBytes = 40 Mbits.*

## Submúltiplos de tiempo (segundos)

milisegundo	ms	$10^{-3}$ s
microsegundo	$\mu$ s	$10^{-6}$ s
nanosegundo	ns	$10^{-9}$ s
picosegundo	ps	$10^{-12}$ s
femtosegundo	fs	$10^{-15}$ s
attosegundo	as	$10^{-18}$ s

## Múltiplos de frecuencia (Hz = ciclo/segundo)

Kilohercio	KHz	$10^3$
Megahercio	MHz	$10^6$
Gigahercio	GHz	$10^9$
Terahercio	THz	$10^{12}$
Petahercio	PHz	$10^{15}$

1 pie (*feet*) = 12 pulgadas = 30,48 cm  
1 pulgada (*inch*) = 1" = 2,54 cm



# Solución a los test

## Soluciones al test del Capítulo 1:

1.1c; 1.2c; 1.3b; 1.4a; 1.5c; 1.6b; 1.7a; 1.8b; 1.9c; 1.10a; 1.11c; 1.12b; 1.13b; 1.14c; 1.15b; 1.16d; 1.17a; 1.18b; 1.19b; 1.20b; 1.21c; 1.22b; 1.23b; 1.24a; 1.25d.

## Soluciones al test del Capítulo 2:

2.1a; 2.2d; 2.3c; 2.4d; 2.5a; 2.6b; 2.7a; 2.8a; 2.9c; 2.10c; 2.11d; 2.12a; 2.13d; 2.14b; 2.15c; 2.16b; 2.17c; 2.18d; 2.19a; 2.20a; 2.21b; 2.22c; 2.23d; 2.24a; 2.25d; 2.26b; 2.27a; 2.28a; 2.29b; 2.30b; 2.31a; 2.32a; 2.33b; 2.34d; 2.35a; 2.36b; 2.37d; 2.38b; 2.39c; 2.40c.

## Soluciones al test del Capítulo 3 :

3.1a; 3.2c; 3.3b; 3.4a; 3.5c; 3.6c; 3.7c; 3.8b; 3.9c; 3.10c; 3.11b; 3.12a; 3.13b; 3.14c; 3.15d; 3.16c; 3.17d; 3.18a; 3.19b; 3.20d; 3.21d; 3.22a; 3.23d; 3.24b; 3.25c; 3.26c; 3.27a; 3.28d; 3.29d; 3.30a; 3.31c; 3.32b; 3.33c; 3.34d; 3.35b; 3.36d; 3.37d; 3.38d.

## Soluciones al test del Capítulo 4:

4.1b; 4.2b; 4.3b; 4.4c; 4.5b; 4.6a; 4.7d; 4.8a; 4.9a; 4.10b; 4.11c; 4.12c; 4.13d; 4.14b; 4.15a; 4.16b; 4.17c; 4.18c; 4.19a; 4.20d; 4.21b; 4.22a; 4.23b; 4.24c; 4.25c; 4.26c; 4.27c; 4.28b; 4.29d; 4.30b; 4.31b; 4.32c; 4.33c; 4.34b; 4.35b; 4.36d; 4.37b; 4.38d.

## Soluciones al test del Capítulo 5:

5.1b; 5.2b; 5.3b; 5.4c; 5.5d; 5.6b; 5.7a; 5.8a; 5.9b; 5.10c; 5.11c; 5.12d; 5.13c; 5.14b; 5.15c; 5.16c; 5.17a; 5.18a; 5.19d; 5.20b; 5.21a; 5.22a; 5.23c; 5.24c; 5.25d; 5.26b; 5.27a; 5.28c; 5.29c; 5.30c; 5.31c; 5.32b; 5.33c.

## Soluciones al test del Capítulo 6:

6.1c; 6.2c; 6.3a; 6.4d; 6.5b; 6.6a; 6.7d;

6.8c; 6.9b; 6.10c; 6.11b; 6.12d; 6.13c; 6.14c; 6.15c; 6.16a; 6.17a; 6.18c; 6.19b; 6.20b; 6.21a; 6.22c; 6.23a; 6.24b; 6.25c.

## Soluciones al test del Capítulo 7:

7.1a; 7.2c; 7.3c; 7.4d; 7.5a; 7.6b; 7.7c; 7.8a; 7.9b; 7.10d; 7.11c; 7.12d; 7.13a; 7.14a; 7.15d; 7.16b; 7.17a; 7.18c; 7.19c; 7.20d.

## Soluciones al test del Capítulo 8:

8.1d; 8.2b; 8.3a; 8.4b; 8.5c; 8.6c; 8.7c; 8.8a; 8.9d; 8.10c; 8.11a; 8.12d; 8.13b; 8.14c; 8.15c; 8.16b; 8.17d; 8.18a; 8.19a; 8.20b; 8.21d; 8.22c; 8.23a; 8.24d; 8.25c; 8.26b; 8.27a; 8.28c; 8.29b; 8.30c; 8.31d; 8.32a; 8.33a; 8.34a; 8.35b; 8.36b; 8.37d; 8.38c; 8.39b; 8.40c; 8.41a; 8.42d; 8.43d; 8.44b; 8.45c; 8.46b; 8.47b; 8.48d; 8.49a; 8.50c; 8.51d.

## Soluciones al test del Capítulo 9:

9.1c; 9.2c; 9.3b; 9.4b; 9.5b; 9.6d; 9.7b; 9.8b; 9.9a; 9.10c; 9.11b; 9.12d; 9.13b; 9.14b; 9.15b; 9.16a; 9.17a; 9.18a; 9.19b; 9.20d; 9.21b; 9.22c; 9.23a; 9.24c; 9.25b; 9.26d; 9.27a; 9.28b; 9.29c; 9.30d; 9.31d; 9.32c; 9.33b; 9.34a; 9.35a; 9.36a; 9.37b; 9.38c; 9.39a; 9.40a; 9.41c; 9.42a; 9.43c; 9.44a; 9.45c; 9.46b; 9.47c; 9.48b; 9.49a; 9.50a.

## Soluciones al test del Capítulo 10:

10.1d; 10.2a; 10.3d; 10.4b; 10.5a; 10.6b; 10.7c; 10.8d; 10.9d; 10.10a; 10.11b; 10.12c; 10.13a; 10.14a; 10.15d; 10.16b; 10.17c; 10.18c; 10.19d; 10.20d; 10.21c; 10.22a; 10.23c; 10.24c; 10.25b; 10.26d; 10.27c; 10.28a; 10.29b; 10.30d; 10.31a; 10.32b; 10.33d; 10.34a; 10.35a; 10.36c; 10.37d; 10.38a; 10.39c; 10.40c.

## Soluciones al test del Capítulo 11:

11.1c; 11.2a; 11.3a; 11.4d; 11.5d; 11.6c; 11.7d; 11.8a; 11.9b; 11.10c; 11.11d; 11.12d; 11.13b; 11.14c; 11.15a.

## Soluciones al test del Capítulo 12:

12.1d; 12.2d; 12.3d; 12.4a; 12.5d; 12.6a; 12.7b; 12.8c; 12.9c; 12.10a; 12.11b; 12.12a; 12.13b; 12.14b; 12.15c; 12.16d; 12.17a; 12.18a; 12.19b; 12.20b; 12.21a; 12.22b; 12.23d; 12.24a; 12.25d; 12.26c; 12.27a; 12.28d; 12.29b; 12.30d; 12.31a; 12.32d; 12.33a; 12.34a; 12.35b; 12.36a; 12.37c; 12.38a; 12.39a; 12.40d.

## Soluciones al test del Capítulo 13:

13.1d; 13.2a; 13.3d; 13.4c; 13.5a; 13.6b; 13.7c; 13.8d; 13.9a; 13.10b; 13.11a; 13.12b; 13.13c; 13.14a; 13.15c; 13.16b; 13.17d; 13.18b; 13.19c; 13.20a; 13.21d; 13.22a; 13.23a; 13.24b; 13.25a; 13.26c; 13.27b; 13.28c; 13.29d; 13.30a; 13.31b; 13.32b; 13.33a.

## Soluciones al test del Capítulo 14:

14.1d; 14.2b; 14.3a; 14.4a; 14.5b; 14.6c; 14.7d; 14.8a; 14.9c; 14.10b; 14.11d; 14.12d; 14.13a; 14.14d; 14.15b; 14.16a; 14.17a; 14.18d; 14.19b; 14.20d; 14.21c; 14.22d; 14.23a; 14.24b; 14.25a; 14.26c; 14.27d; 14.28b; 14.29c; 14.30c.

## Soluciones al test del Capítulo 15:

15.1c; 15.2d; 15.3b; 15.4a; 15.5b; 15.6c; 15.7d; 15.8b; 15.9d; 15.10a; 15.11b; 15.12b; 15.13d; 15.14a; 15.15a; 15.16c; 15.17d; 15.18a; 15.19a; 15.20c; 15.21d; 15.22a; 15.23b; 15.24c; 15.25c; 15.26a; 15.27b; 15.28c; 15.29d; 15.30b; 15.31c; 15.32c; 15.33b; 15.34a; 15.35a; 15.36b; 15.37b; 15.38c; 15.39d; 15.40b; 15.41d; 15.42c; 15.43b; 15.44b.



## BIBLIOGRAFÍA

- [1] J. G. Brookshear, *Computer Science*, 7.<sup>a</sup> ed., Pearson, 2003.
- [2] C. J. Date, *An Introduction to Database Systems*, 7.<sup>a</sup> ed., Addison-Wesley, 2000.
- [3] T. L. Floyd, *Fundamentos de sistemas digitales*, 7.<sup>a</sup> ed., Prentice-Hall, 2003.
- [4] B. A. Forouzan, *Foundations of Computer Science*, Thomson, 2003.
- [5] D. Gajski, *Principios de Diseño Digital*, Prentice-Hall, 1997.
- [6] H. García-Molina, J. D. Ullman y Jennifer Widom, *Database Systems: The Complete Book*, Prentice-Hall, 2002.
- [7] C. Hamacher, Z. Vranesic y S. Zaky, *Organización de computadores*, 5.<sup>a</sup> ed., McGraw-Hill, 2003.
- [8] L. Joyanes, *Fundamentos de Programación*, McGraw-Hill, 2003.
- [9] B. A. Kernighan y R. Pike, *La práctica de la programación*, Prentice-Hall, 2000.
- [10] D. E. Knuth, *The Art of Computer Programming*, Addison-Wesley, 2003.
- [11] K. C. Loudon, *Programming Languages*, 2.<sup>a</sup> ed., Thomson, 2003.
- [12] A. Lloris, A. Prieto y L. Parrilla, *Sistemas Digitales*, McGraw-Hill, 2003.
- [13] A. B. Marcovitz, *Introduction to Logic Design*, McGraw-Hill, 2002.
- [14] P. Norton, *Introduction to Computers*, 5.<sup>a</sup> ed., McGraw-Hill, 2003.
- [15] J. Ortega, M. Anguita y A. Prieto, *Arquitectura de Computadores*, Thomson, 2005.
- [16] R. S. Presman, *Ingeniería del software: un enfoque práctico*, 5.<sup>a</sup> ed., McGraw-Hill, 2003.
- [17] A. Prieto, A. Lloris y J. C. Torres, *Introducción a la Informática*, 3.<sup>a</sup> ed., McGraw-Hill, 2002.
- [18] A. Silberschatz, H. F. Korth y S. Sudarshan, *Fundamentos de bases de datos*, 4.<sup>a</sup> ed., McGraw-Hill, 2002.
- [19] A. Silberschatz, P. B. Galvin y G. Gagne, *Operating System Concepts*, 6.<sup>a</sup> ed., John Wiley and Sons, 2002.
- [20] I. Sommerville, *Ingeniería de software*, Pearson Education, 2002.
- [21] W. Stallings, *Computer organization and architecture: designing for performance*, 6.<sup>a</sup> ed., Pearson Education, 2003.
- [22] W. Stallings, *Comunicaciones y redes de computadores*, 7.<sup>a</sup> ed., Pearson Education, 2004.
- [23] W. Stallings, *Sistemas operativos: principios de diseño e interioridades*, 4.<sup>a</sup> ed., Prentice-Hall, 2001.
- [24] A. S. Tanenbaum, *Modern Operating Systems*, 2.<sup>a</sup> ed., Prentice-Hall, 2001.
- [25] A. S. Tanenbaum, *Computer Networks*, 4.<sup>a</sup> ed., Prentice-Hall, 2003.
- [26] *TechWebEncyclopedia*, <http://www.techweb.com/encyclopedia/>.
- [27] A. Tucker y R. Noonan, *Lenguajes de programación*, McGraw-Hill, 1998.
- [28] J. F. Wakerly, *Diseño digital: principios y prácticas*, Prentice-Hall, 2001.
- [29] *Webopedia*, <http://www.webopedia.com/>.



# ÍNDICE

- Abrir el archivo, 441
- Acceso
  - calculado, 438
  - directo, 184, 436
  - múltiple, 460
  - por bloques, 178
  - por bytes, 178
  - por palabras, 178
  - secuencial, 184
- Acelerador de gráficos, 227, 254
- Acoplamiento de, 494
  - control, 494
  - datos, 494
- Actividad, 440
- Actuador, 231
- Acumulador, 105
- Adaptador de red, 265
- ADDS (suma), 119
- ADSL, 272, 273
- Agotamiento, 337
- Agrupación redundante de discos independientes, 189
- Algoritmo, 373
  - de reemplazo, 182
  - FCFS, 311
  - SSF, 311
  - SCAN, 311
- Algoritmos de planificación, 298
- Altavoces, 230
  - externos, 231
- Análisis, 490
  - del problema, 382
- Analizador
  - gramatical, 407
  - lexicográfico, 407
- Ancho de
  - banda, 6, 178
  - bus, 5
- Ángulo de visión, 222
- Anillo, 267
  - con paso de testigo, 271
- Aplicaciones
  - básicas de Internet, 277
  - científico-técnicas, 415
  - de gestión de información, 415
  - de inteligencia artificial, 415
  - de la informática, 13
  - de la web, 281
  - de programación de sistemas, 416
  - horizontales, 15
  - multimedia, 15, 28, 230
  - para web, 416
  - verticales, 15
- Árbol, 267
- Árboles, 350
- Archivos, 312, 431
  - constantes, 432
  - de índices, 442
  - de maniobras, 432
  - de resultados, 432
  - históricos, 432
  - indexados, 436
  - inmediatos, 443
  - intermedios, 432
  - maestros, 431
  - permanentes, 431
  - temporales, 432
- Arquitectura del computador, 13
- Arquitecturas superescalares, 257
- Array, 340
- Asignaciones de, 376
  - memoria, 129
  - registros, 129
- ATA, 250
- ATA-2, 250
- ATA-IDE, 254
- ATM, 273
- Atributos, 460
- B-* (salto), 122
- Bancos de registros, 87, 105
- Barrido de rastreo, 225-226
- Básculas, 84
- Base, 36
- Bases de datos, 459
  - distribuidas, 473
  - distribuidas con duplicación, 473
  - distribuidas con fragmentación, 473
  - orientadas a objetos, 473
- BCD
  - desempaquetada, 36
  - empaquetada, 36
- Bibliotecas de funciones, 406
- Biestables, 83, 84
  - cerrojo, 84
  - RS, 83
  - sincronizados, 84
- Bit de
  - modificación, 181
  - paridad, 43
- Bloques, 184, 195, 440
  - combinacionales básicos, 72
  - de datos, 180
  - físicos, 191
  - secuenciales básicos, 85
- BMP, 31
- Bobinas deflectoras, 224
- Buffer, 214, 440
  - tri-estado, 77
- Burbuja térmica, 229
- Bus, 5, 248, 266
  - AGP, 250
  - de datos, 176
  - de direcciones, 176
  - del sistema, 245
  - PC/XT, 249
- Búsqueda binaria, 436
- Cabecera, 346, 347, 398
  - del procedimiento, 404
- Cabeza, 183
- Cabezas magnetorresistivas, 184
- Cable módem, 273
- CAD, 14
- Cadenas de caracteres, 343
- CADMAT, 14
- CAI, 14
- CAL, 14
- Calidad fotográfica, 45
- CALL- (llamada a subrutina), 112, 123
- CAM, 14
- Cámaras digitales de vídeo, 231
- Campo de dirección, 8



- Campos, 431, 461
- Canal de entrada/salida, 247
- Capa
  - de aplicación, 269
  - de enlace de datos, 270
  - de presentación, 269
  - de procesos y aplicaciones, 276
  - de red, 269, 277
  - de sesión, 269
  - de transporte, 269, 277
  - física, 270
  - internet, 277
- Capacidad, 440
  - de almacenamiento, 6
- Cápsula de lectura/escritura, 183
- Caracteres, 338
  - alfabéticos, 25
  - ASCII, 519
  - de control, 26
  - de control ASCII, 519
  - especiales, 26
  - geométricos, 26
  - gráficos, 26
  - numéricos, 25
- Cargador, 410
- Carpeta, 312
- Cartuchos, 192
  - compactos, 192
- Case, estructura, 401
- Cauce, 256
- CD-ROM, 194
- Celdas, 183, 221
- Cerrar el archivo, 441
- Cerrojos, 84
- Chat*, 278
- Chipset, 252
- Ciclos, 376
  - condicionales, 401
  - de la vida de software, 487
  - repetitivos, 401
- Cilindro, 187
- Cintas magnéticas, 190
  - clásicas, 192
  - DAT, 193
  - de tipo carrete, 190, 192
  - DLT/SDLT, 193
  - EXABYTE, 193
  - LTO, 193
  - QUIC, 192
  - SAIT, 193
- CITT, 266
- Clases, 353
- Clasificación de
  - computadores según generalidad de uso, 9
  - los lenguajes de programación, 410
- Clave, 433
- Cliente, 276
- Cluster de computadores, 259
- CODEC AC97, 254
- Codificación
  - con diccionario adaptativo, 44
  - de la información en CD-ROM, 195
  - dependiente de la frecuencia, 44
  - Lempel-Ziv LZ77, 45
  - por longitud de secuencias, 44
- Codificador
  - binario, 75
  - de prioridad, 75
- Código
  - ASCII, 26
  - de barras, 219
  - de canal, 194
  - de entrada/salida, 26, 519
  - de operación, 8, 396
  - EBCDIC, 26
  - ISO 8859-n, 27
  - Unicode, 27
- Codop, 8
- Coefficiente de respuesta, 297
- Cohesión, 494
- Cola, 347
  - lineal, 348
- Comentarios, 400
- Compactación, 303
- Comparar, 343
  - dos números, 125
- Compiladores, 398, 406, 410
- Complementación, 64
- Compresión de datos, 43, 44
- Compresión
  - en GIF, 45
  - en JPEG, 45
  - MPEG, 46
  - MP3, 46
- Computador, 1
  - de uso específico, 9
  - de uso general, 9, 10
  - huésped, 266
  - móvil, 11
  - personal, 11
  - von Neumann, 3
- Concatenar, 343
- Conectores
  - de panel frontal, 254
  - PCI, 254
- Confidencialidad y seguridad, 460
- Conmutador, 273
- Constantes, 336
- Consulta directa, 460
- Contadores, 87
  - ascendentes y descendentes, 126
  - de programa, 107
- Controlador
  - de entradas/salidas, 246
  - de teclado, 215
  - de vídeo, 223, 225
- gráfico, 223
- DMA, 247
- Conversor
  - analógico/digital (A/D), 28, 214, 232
  - digital/analógico (D/A), 214, 233
- Copiar el contenido de un registro en otro, 124
- Copias de seguridad, 190
- Correo electrónico, 278
- CPLD, 81
- CPM, 259
- CPU, 5
- Crecimiento, 440
- Criterio del
  - extremo mayor, 32, 177
  - extremo menor, 32, 177
- Crominancias, 45
- CRT, 223
- CYMK, 228
- Datos, 412
  - de tipo entero, 33
  - de tipo real, 36
- DBMS, 463
- Decisiones, 376
  - dobles, 401
  - múltiples, 401
  - sencillas, 401
- Declaración, 335
- Decodificador binario, 75
- Delete*, 466
- Demultiplexor, 76
- Dependencias de
  - datos, 256
  - instrucciones, 256
- Desbordamiento, 336
- Descompresión de datos, 44
- Descripción
  - algebraica, 65
  - en lenguaje natural, 65
- Descriptor de archivos, 441
- Detectar si un número es cero o negativo, 124
- Detectar si un número es par, 126
- Detector, 231
  - de datos magnetizados, 221
  - óptico, 218
- Diagramas
  - de estado, 490
  - de flujos de datos, 490
  - sintácticos, 407
- Diference*, 472
- Dígitos decimales codificados en binario, 33
- DIMM, 252
- Dirección
  - base, 300
  - de la web, 279
  - en Internet, 275
  - IP, 275

- Direcciones
  - alineadas, 177
  - lógicas, 300
  - no alineadas, 177
  - virtuales, 300
- Directorio, 312
- Discos, 353
  - cartucho, 187
  - compacto, 193
  - compactos grabables, 196
  - compactos regrabables, 196
  - de cabezas fijas, 187
  - digital versátil (DVD), 193
  - duros, 187
  - magnéticos, 184
  - magnéticos, principios de funcionamiento, 185
  - ópticos, 193
  - Winchester, 187
- Diseño, 492
  - del algoritmo, 382
- Dispositivo
  - de carga acoplada, 220
  - de encaminamiento, 273
  - de interconexión en redes, 273
  - de red, 266, 273
  - lógicos programables, 78
- Disquetera, 254
- Disquetes, 187
- Distribuidor, 298
- DMAC, 247
- DNS, 275, 277
- Documentación
  - para el usuario, 489
  - técnica, 489
- Documentos
  - activos, 281
  - dinámicos, 281
  - estáticos, 281
- DRAM, 89
- DSL, 272
- Editor de textos, 409
- Efecto, 231
- Eficiencia de un código, 42
- EFM, 195
- EIDE, 250
- EISA, 250
- Ejecución de un programa, 409
- Elemento
  - de imágenes, 30
  - de memoria, 83
  - internos de un procesador, 105
- Enlazador, 410
- Ensamblador, 397
- Enteros, 336
  - en complemento a dos, 34
  - en complemento a uno, 34
  - en signo y magnitud, 33
  - sin signo, 33
- Entidades, 460
- Entrada
  - de reloj, 84
  - manuales directas, 215
- Entradas/Salidas, 376
  - de señales analógicas, 231
- Entrelazado de memoria, 179
- Enumerados, 339
- EOF, 433
- Escáneres, 218
- Escritura
  - de memoria, 176
  - inmediata, 182
- Esquema, 461
  - de funcionamiento de un procesador, 107
- Estaciones de trabajo, 11
- Estados, 82
  - activo, 295
  - bloqueado, 295
  - de espera, 123
  - inicial, 83
  - preparado, 295
- Estilos de programación, 412
- Estrella, 266
- Estructura
  - básicas de interconexión, 245
  - de computadores, 245
  - de control, 376
  - de datos, 335, 340
  - definidas por el usuario, 353
  - funcional de los computadores, 3
  - unibus, 245
- Ethernet, 270
- Eventos asíncronos, 414
- Evolución de sistemas operativos, 292
- Exabyte, 3
- Exponente, 36
- Expresión
  - algebraica, 400
  - de caracteres, 400
  - lógica, 400
- Extraer, 343
- Factor
  - de bloqueo, 440
  - de velocidad, 197
- Fase de
  - captación de instrucción, 108
  - diseño, 492
  - ejecución, 108
- Fast Ethernet, 270
- FAT16, 442
- FAT32, 442
- FCFS, 299
- FIFO, 309
- FireWire, 251
- Flexibilidad, 460, 497
- Flip-flops, 83
- Forma
  - canónica, 69
  - normalizada, 67
- Formato de datos, 116
- FPGA, 81
- Frecuencia de
  - consulta, 440
  - refresco, 226
  - reloj, 5
  - renovación, 440
- FTP, 277, 278
- Funciones, 404, 405
  - AND, 65
  - de conmutación, 64
  - exclusive-EXNOR complementada, 65
  - exclusive-OR complementada, 65
  - exclusive-OR, 65
  - exclusive-XOR, 65
  - NAND, 65
  - NO, 64
  - NOR, 65
  - NOT, 64
  - OR, 65
- Futurebus+, 250
- Gama de colores, 223
- Generación de códigos, 409
- Generador de pulsos, 5
- Gestión de archivos, 312
  - de entradas/salidas, 309
  - de la memoria, 300
  - de proyectos, 486
  - del procesador, 294
- Gestor del teclado, 215
- Gigabit Ethernet, 270
- Gigabyte, 3
- Grabación
  - helicoidal, 192
  - lineal paralelo, 191
  - lineal serpentina, 192
- Grado, 350
- HALT* (parada), 123
- Hardware, 12
- HDSL, 272
- Hebras, 257, 415
- Herramientas de software, 486
- Heterogéneas, 340
- Hijos, 350
- Hojas, 350
- Homogéneas, 340
- Hoyos, 194
- HRRN, 299
- HTTP, 277
- IDE, 250
- Identificadores, 399
- Identificar subcadena, 343
- IDSL, 272
- If-then*, 401

- If-then-else*, 401
- ILP, 255
- Imagen del proceso, 306
- Implementación, 495
  - de dos niveles, 71
  - de funciones. Puertas lógicas, 69
  - del algoritmo, 382
- Impresoras, 227
  - de cera, 228
  - de inyección de tinta, 229
  - de papel térmico, 228
  - de transferencia térmica con sublimación, 229
  - láser, 229
  - matriciales o de agujas, 228
  - térmicas, 228
- IN (entrada), 119
- Independencia
  - física, 460
  - lógica, 460
- Índice, 436
- Información magnética, 183
- Informática, 1
- Ingeniería del software, 485
- I-nodos, 442
- Insert*, 466
- Instrucciones, 395
  - de salto condicional, 112
  - máquina, 109
- Interbloque, 191
- Interfaz de alto nivel, 460
- Internet, 265, 273
- Intérpretes, 398, 406
- Intersection, 471
- Inversión, 64
- IRC, 278
- IRG, 191
- Irregular, 267
- ISA Bus, 249
- ISO, 266
  - 8859-1, 27
  - Latín-1, 27
- ISO/IEC 10646, 27
- JavaScript, 281
- Jerarquía de memoria, 181
- Join*, 469
- JPEG, 46
- Kilobyte, 3
- LAN, 268, 270
- Lápiz
  - de presión, 216
  - electrostático, 216
  - óptico, 216
- Latencia, 178
  - rotacional, 186
- LD (cargar un registro con un dato de la memoria), 118
- Lectores ópticos de marcas, 219
- Lectura anticipada de
  - datos, 189
  - memoria, 176
- Lenguaje
  - de alto nivel, 397
  - de programación, 8, 395
  - ensamblador, 129, 397
  - imperativo, 398, 412
  - macroensamblador, 397
  - máquina, 8, 129, 395
  - simbólico, 396
- LHI (carga inmediata alta), 119
- LIFO, 346, 347
- Línea telefónica
  - convencional, 272
  - de interconexión, 266
  - digitales, 272
- Lineal, 266
- Listas, 344
  - contigua, 344
  - de enlaces, 441
  - encadenada, 441
  - enlazadas, 344
- Llave, 433
- LLI (carga inmediata baja), 118
- Localidad
  - de las referencias, 180
  - espacial, 180
  - temporal, 180
- Lógica de control, 107
- Lógicos, 337
- Longitud de la palabra, 6
- LRU, 308
- Luminancia, 45
- Luminóforos, 221, 224
- Macro, 397
- Macroinstrucciones, 397
- Magnitudes, 336
- Malla, 267
- MAN, 268
- Mantisa, 36
- Mapa de
  - bits, 30
  - colores, 226
  - vectores, 31
- Máquina de fotos digital, 231
- Marcos de página, 303
- Matrices de visualización
  - activas, 225
  - pasivas, 225
- MCA, 250
- Medida de prestaciones de memoria, 178
- Megabyte, 3
- Memoria, 175
  - auxiliar, 4, 213
  - caché, 180
  - de vídeo, 226
  - direccionables por bytes, 177
  - direccionables por palabras, 176
  - externa, 4, 181, 183, 213
  - intermedia, 245
  - interna, 175
  - magnéticas, 183
  - masiva, 4
  - ópticas, 183
  - principal, 4, 175
  - RAM, 4, 88
  - ROM, 4
  - SDRAM, 254
  - secundaria, 4
  - temporal, 214
  - virtual, 305
- Mensaje, 268
- MFLOPS, 7
- Microcomputador, 11
- Microcontrolador, 130
- Microfonos, 230
- Microprocesador, 5, 129
- MIDI, 230
- MIMD, 258
- Minimización algebraica de funciones, 65
- MIPS, 7
- Modelo
  - arquitectónico, 494
  - de abajo a arriba, 493
  - de arriba a abajo, 493
  - de bases de datos, 463
  - de comportamiento, 494
  - de diseño, 493
  - de marcos, 493
  - de referencia OSI, 268
  - en cascada, 488
  - en red, 464
  - envase-contenido, 493
  - estructural, 494
  - incremental, 488
  - jerárquico, 463
  - relacional, 464
  - suscriptor-editor, 493
- Modos de asignación del procesador a los procesos, 297
- Modularidad, 494
- Módulos de memoria, 252
- Monitores de visualización, 221
- MPP, 259
- Muestreo, 28
- Multicomputadores, 258
- Multiplexor, 76
  - analógico, 233
- Múltiplos de frecuencia, 521
- Multiprocesadores, 258
- Multiprogramación, 295, 299
  - apropiativa, 298
  - no apropiativa, 298
- Multipunto, 266
- NAND (operación lógica NAND), 120
- Navegadores web, 280

- Negación, 64
- NIC, 265
- Nivel
  - de lenguaje máquina, 13
  - de máquina simbólica, 13
  - de temporización, 249
  - de transferencias, 249
  - eléctrico, 249
  - electrónico, 13
  - lógico, 249
  - mecánico, 249
- Niveles conceptuales de descripción de un computador, 12
- NNTP, 278
- Nodos, 344
  - terminales, 350
- No hacer nada (instrucción no operativa), 125
- Notación
  - BNF, 407
  - científica, 36
  - en coma flotante, 36
  - exponencial, 36
- NRU, 309
- Número de líneas de código, 486
  
- Objeto, 412
- Obtener la longitud, 343
- Operaciones lógicas, 337
- Operadores, 399
  - aritméticos, 399
  - de asignación, 399
  - de relación, 399
  - lógicos, 399
  - otros, 399
- Optimización, 382, 409
- Ordenador, 1
- Organigramas, 375
- Organización secuencial, 433
- OUT (salida), 119
  
- P2P, 278
- Paginación, 303
- Palabra, 6
  - de memoria, 6
  - reservada, 398
- Paleta de color, 45
- Paneles sensibles a la presión, 215
- Pantalla
  - de cristal líquido, 224
  - de efecto de campo (FED), 225
  - de plasma, 225
  - de tubo de rayos catódicos, 223
  - de vídeo, 223
  - electroluminiscentes, 225
  - plana, 224
  - sensible a la presión, 215
- Paquetes de discos, 187
- Paralelismo en computadores, 254
  
- Parámetros
  - para caracterización de prestaciones, 6
  - reales, 404
- Paridad
  - impar, 43
  - par, 43
- Particiones, 191
  - dinámicas, 302
  - estáticas, 301
- Pasarela, 273
- Paso de
  - mensajes, 412
  - punto, 223
- PC, 11
  - compatible, 252
- PCB, 248
- PCI, 250
- PDU, 268
- Película delgada, 184
- Periféricos, 5
  - de E/S, 213
  - ejemplos de, 214
  - para aplicaciones multimedia, 230
  - parte electrónica, 214
  - parte mecánica, 214
- Petabyte, 3
- Pilas, 346
- Pistas, 183, 185
- Píxel, 30, 221
- Planificación de prioridad, 299
- Planteamiento del problema, 382
- Polimorfismo, 412
- POP, 278
- Portabilidad, 497
- Posición, 6
- Primero
  - al que reste menor tiempo de procesador, 299
  - el de mayor coeficiente de respuesta, 299
  - el de menor tiempo de procesador, 299
  - en llegar, primero en procesar, 299
- Principios de álgebra y conmutación, 63
- Procedimientos, 404
- Procesadores, 5, 105
  - CISC, 131
  - de lenguaje, 354, 398
  - digitales de señales, 234
  - gráficos, 227
  - matriciales, 258
  - multihebra, 257
  - RISC, 131
  - RISC y CISC, 130
  - vectoriales, 258
  - VLIW, 257
- Procesamiento
  - cola serie, 299
  - de transacciones, 300
  - paralelo, 255
- Proceso de
  - creación de un programa, 382
  - traducción, 406
- Producto lógico, 65
- Programa, 395
  - de suma de dos tablas, 127
  - en código máquina (CODE-2), 114
  - fuelle, 406, 410
  - objeto, 406
- Programación
  - concurrente, 415
  - declarativa, 414
  - en lenguaje máquina, 114
  - guiada por elementos, 414
  - imperativa, 412
  - orientada a objetos, 412
- Programas e instrucciones, 8
- Project, 469
- Protección contra fallos, 460
- Protocolo
  - HTTP, 280
  - TCP/IP, 276
- Prototipado, 488
- Prueba, 382, 495
  - beta, 496
- Pseudocódigo, 374, 495
- Puente, 273
- Puerta lógica, 69
  - tri-estado, 77
- Puesta a cero o a uno de un registro, 124
- Pulverización piezoeléctrica, 229
- Punero, 344, 347
  - de cabecera, 344
  - de instrucciones, 107
  - de pila, 107, 113, 115
- Punto a punto, 266
- Puntos de pantalla (véase píxel), 221, 224
  
- Quick-Ring, 250
  
- RADSL, 272
- RAID, 189
- RAM
  - dinámicas, 89
  - estáticas, 89
- Ramas, 350
- Ratón
  - estacionario, 218
  - mecánico, 217
  - óptico, 217
  - opto-mecánico, 217
- RDSI (Red Digital de Servicios Integrados), 272
- Reales, 336
- Reconocimiento
  - del habla, 231
  - óptico de caracteres, 220
- Recorrido
  - en orden, 352
  - en post-orden, 352

- en pre-orden, 352
  - primero en anchura, 352
  - primero en profundidad, 352
- Recursividad, 381
- Recursos
  - hardware, 486
  - humanos, 486
  - software, 486
- Redes de
  - área amplia (WAN), 268, 272
  - área local (LAN), 268, 270
  - computadores, 265
  - comunicaciones, 266
- Redondeos, 39
- Redundancia controlada, 460
- Refresco, 225
- Registros, 5, 85, 343, 461
  - de desplazamiento, 86
  - de desplazamiento con carga en paralelo, 86
  - de dirección, 115
  - de instrucción, 107
  - de uso general, 105
  - físicos, 184, 440
- Relación
  - de aspecto, 222
  - panorámica, 222
- Relativa o incremental, 44
- Reloj, 5, 107, 309
  - de tiempo real, 252
- Rendimiento, 7, 297
- Repertorio de instrucciones máquina, 116
- Repetidores, 273
- Representación
  - binaria, 33
  - con exceso, 34
  - de algoritmo, 374
  - de complementos, 512
  - de datos numéricos, 32
  - de imágenes, 29
  - de sonidos, 28
  - de textos, 25
  - sesgada, 34
- Reproducción de sonido, 230
- Requisitos del software, 490
- Resolución, 30
  - gráfica, 222
  - óptica, 224
- RET, 112, 123
- RIMM, 252
- RTP, 277
- Ruta, 312
- Rutina, 112, 403
- SCSI, 250
- SDSL, 272
- Sectores, 185
- Secuencias, 376
- Segmentación, 303
  - de cauce, 256
- Segmento, 300
- Select, 468
- Sensor, 231
- Sentencias
  - de asignación, 400
  - declarativas, 398
  - de control, 401
  - de entrada/salida, 401
  - imperativas, 398
- Señal
  - analógica, 28, 231
  - de control, 5
  - de estado, 5
- Serial ATA (SATA), 254
- Servicios de red, 11
- Servidor, 276
- SHL (desplazamiento a izquierda), 120
- SHR (desplazamiento a derecha), 121
- SHRA (desplazamiento aritmético a derecha), 122
- Signos de puntuación, 400
- Símbolo, 194
- SIMD, 258
- SIMM, 252
- Simple presión, 37
- Síncronos, 82
- Sinónimos, 439
- Síntesis
  - de sonido, 230
  - del habla, 231
- SISD, 257
- Sistemas
  - combinacionales, 61
  - de adquisición de datos analógicos, 223
  - de audio, 230
  - de gestión de bases de datos, 462
  - de multiprocesamiento, 300
  - de multiusuario, 299
  - de numeración, 509
  - de postescritura, 181
  - de vídeo, 231
  - digital binario, 61
- Sistema operativo
  - de multiprocesamiento, 293
  - de multiprogramación, 295
  - distribuido, 294
  - operativo en red, 293
  - secuenciales, 82
  - ZCAV, 188
- SMP, 258
- SMTP, 276, 278
- Software, 12
  - abierto, 493
  - de aplicación, 292
  - de calidad, 496
  - de control, 292
  - de diagnóstico, 292
  - de sistema, 291
  - de un computador, 291
  - específico, 231
- Soporte
  - físico, 12
  - lógico, 12, 291
- SP, 113
- SPLD, 78
- SPN, 299
- SQL, 466
- SRAM, 89
- SRT, 299
- ST (almacenar un contenido de un registro en la memoria), 118
- Subalgoritmo, 380
- Submúltiplos de tiempo, 521
- Subprogramas, 403
- Subpuntos, 221
- Subrango, 340
- Subrutina, 112, 403
- SUBS (resta), 120
- Suma lógica, 65
- Sumador binario, 72
- Supercomputadores, 11
- Switch, 401
- Tabla
  - de decisión, 490
  - de procesos, 302
  - de símbolos, 409
  - de ubicación de archivos, 441
  - verdad, 62
- Tamaño de la pantalla, 222
- Tarjeta
  - circuito impreso, 248
  - de sonido, 231
  - edición de vídeo, 231
  - expansión, 248
  - red, 265
- Tasa
  - consulta, 440
  - de aciertos, 183
  - fallos, 183
  - renovación, 440
- TCP, 277
- Teclados, 215
  - de membrana, 215
  - mecánico, 215
- Tecnología
  - de computadores, 13
  - SAIT-1, 193
- TELNET, 277, 278
- Teorema de Shannon, 67
- Terabyte, 3
- Término mínimo, 67
- Test de caja de cristal, 496
- Testeabilidad, 497

- Testigo, 271
- Tiempo
  - compartido, 299
  - de acceso, 6, 184, 186
  - de acceso a memoria, 178
  - de búsqueda, 186
  - de ciclo, 5
  - de ciclo de memoria, 178
  - de ejecución de un proceso, 297
  - de escritura, 186
  - de espera, 186
  - de lectura, 186
  - de respuesta, 297
  - de retardo, 74
  - de retorno, 297
  - en máquina, 297
  - real, 299
- TIFF, 31
- Tipos
  - bases de datos, 473
  - computadores, 9
  - de datos, 32, 336
- TLD, 275
- Tóner, 229
- Topología de redes, 266
- Traductor, 406
  - de ensamblador, 397
- Transformaciones
  - con la base decimal, 513
  - con la base hexadecimal, 517
  - con la base octal, 515
  - entre bases distintas, 513
- Trucos de programación, 124
- Tubo de rayos catódicos, 223
- Tuplas, 464
- Turno rotatorio, 298
- UDP, 277
- Ultra DMA 336/66/100 Bus IDE, 254
- UMA, 258
- Unidades
  - aritmético-lógicas, 5, 82
  - de control, 5
  - de entrada, 3, 213
  - de gestión de memoria, 301
  - de léxico, 407
  - de salida, 4, 213
  - de ubicación, 312, 441
  - funcionales, 3
  - RAID, 189
- Union*, 470
- Unzip, 45
- Update, 467
- URL, 279
- USB, 251
- USB 2.0, 254
- Utilidades, 292
- Utilización múltiple, 460
- Valles, 194
- Variables, 336
  - de conmutación, 64
  - locales, 404
- VDSL, 272
- Velocidad
  - angular constante, 188
  - lineal constante, 188
- Versión beta, 496
- VLB, 250
- Volatilidad, 440
- Volumen, 440
- WAN, 272
- White, 401
- World Wide Web, 279
- WWW, 279
- XHTML, 281
- Zip, 45









# ***¡Estudia a tu propio ritmo y aprueba tu examen con Schaum!***

Los Schaum son la herramienta esencial para la preparación de tus exámenes.  
Cada Schaum incluye:

- Teoría de la asignatura con definiciones, principios y teoremas claves.
- Problemas resueltos y totalmente explicados, en grado creciente de dificultad.
- Problemas propuestos con sus respuestas.

***Hay un mundo de Schaum a tu alcance...¡BUSCA TU COLOR!***

