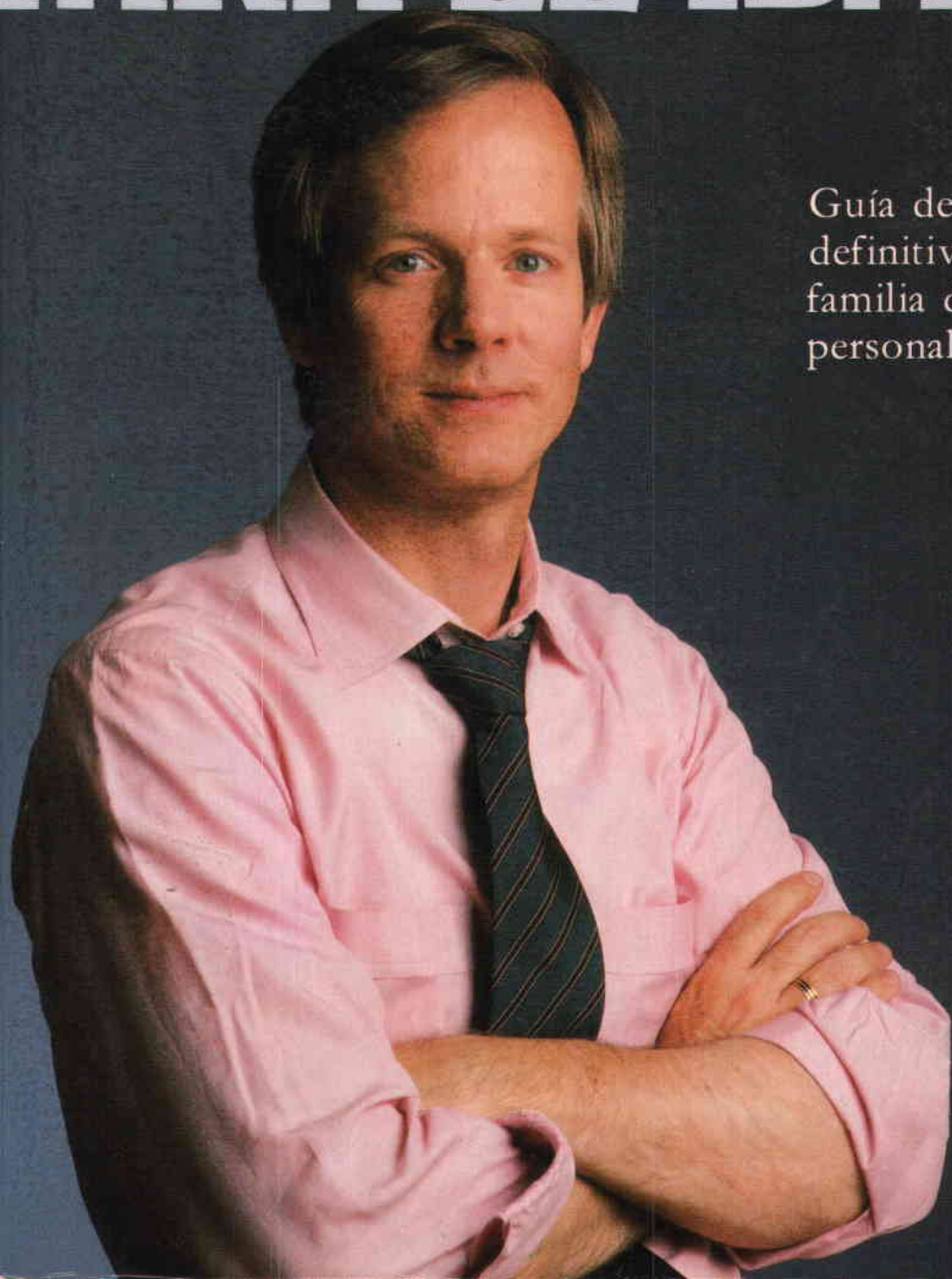
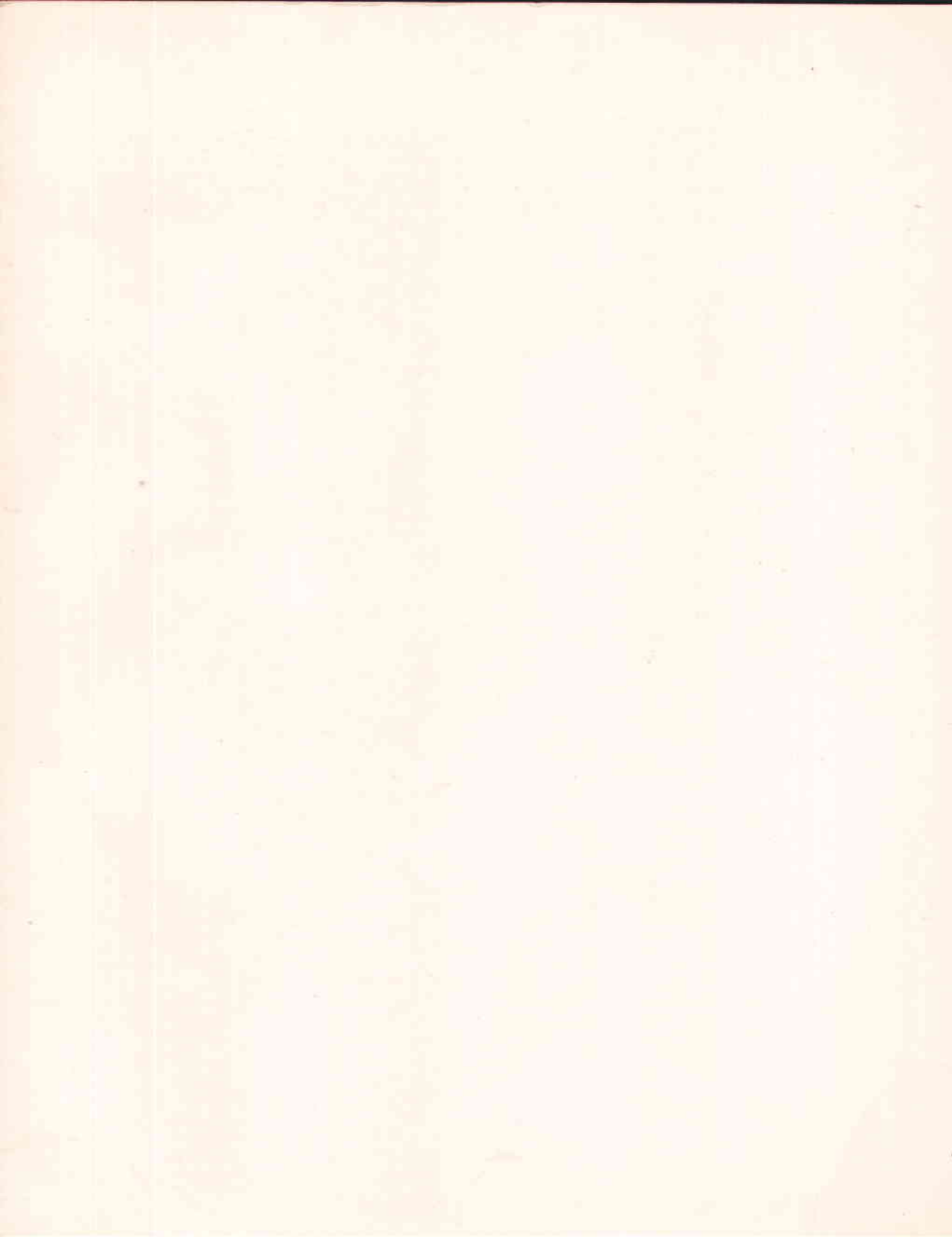


D E T E R M I N O R T O N GUÍA DEL PROGRAMADOR PARA EL IBM PC

Guía de referencia
definitiva para *toda* la
familia de ordenadores
personales IBM





P E T E R N O R T O N
GUIA DEL PROGRAMADOR
PARA EL IBM PC

P E T E R N O R T O N
GUIA DEL PROGRAMADOR
PARA EL IBM PC

La guía definitiva de
referencia de la familia
de ordenadores personales de IBM.

MICROSOFT-ANAYA MULTIMEDIA

Título de la obra original:

The Peter Norton Programmer's Guide to the IBM PC

Traducción: José M. Gutiérrez, Pedro Muro y Francisco J. Serón

Revisión técnica y adaptación: Javier David García

Reservados todos los derechos. Ni la totalidad ni parte de este libro puede reproducirse o transmitirse por ningún procedimiento electrónico o mecánico, incluyendo fotocopia, grabación magnética o cualquier almacenamiento de información y sistema de recuperación, sin permiso escrito de Ediciones Anaya Multimedia, S. A.

Edición original en lengua inglesa

Copyright © 1985 by PETER NORTON

Edición publicada por acuerdo con el editor original,
Microsoft Press, Bellevue, Washington

© EDICIONES ANAYA MULTIMEDIA, S. A., 1987
Villafranca, 22. 28028 Madrid
Depósito legal: M. 13.529-1987
ISBN: 84-7614-130-0
Printed in Spain
Imprime: ORYMU. Pinto (Madrid)

CONTENIDOS

Agradecimientos vii

Introducción ix

- 1 Anatomía del PC 1
- 2 Las entradas y las salidas 21
- 3 El *software* de la ROM 47
- 4 Video, conceptos básicos 77
- 5 Los discos, conceptos básicos 111
- 6 El teclado, conceptos básicos 143
- 7 La generación de sonido 163
- 8 La ROM-BIOS, conceptos básicos 179
- 9 Servicios de video de la ROM-BIOS 191
- 10 Servicios para disco de la ROM-BIOS 209
- 11 Servicios para el teclado de la ROM-BIOS 227
- 12 Miscelánea de servicios 233
- 13 Resumen de servicios de la ROM-BIOS 251
- 14 El DOS, conceptos básicos 269
- 15 Las interrupciones del DOS 277
- 16 Las funciones universales del DOS 301
- 17 Las funciones nuevas del DOS 333
- 18 Resumen de servicios del DOS 367
- 19 La construcción de un programa 379
- 20 Los lenguajes de programación 391

Apéndice A: Controladores de los dispositivos periféricos 429

Apéndice B: Aritmética hexadecimal 437

Apéndice C: Sobre los caracteres 447

Índice alfabético 459

RECONOCIMIENTOS

Tanta gente ha contribuido en la realización de este libro que sería imposible nombrarlos a todos. Sin embargo, hay una persona que se merece una mención y un agradecimiento especial por sus esfuerzos y dedicación en este proyecto: Suzanne Ropiequet.

INTRODUCCION

Mi meta al escribir este libro es simple pero ambiciosa: ayudarle a comprender los principios de la programación de la familia de ordenadores personales de IBM. Desde el momento en que el primer Ordenador Personal de IBM (conocido por nosotros como «el PC») fue presentado en otoño de 1981, quedó claro que iba a ser un ordenador muy importante. Más tarde, cuando las ventas del PC se dispararon, sobrepasando lo que todo el mundo esperaba, incluyendo la IBM, y cuando al modelo original se le añadieron un hermano o dos, el PC empezó a ser reconocido como el modelo estándar de los ordenadores personales serios. Desde el PC original ha evolucionado una familia completa de ordenadores, una familia con muchas ramificaciones. Y al mismo tiempo, también ha crecido la importancia de la familia PC. Sin embargo, el hecho de que cada miembro de la familia difiera de sus parientes en sus detalles y características ha provocado que el desarrollo de programas para la familia sea muy complejo.

Este libro trata sobre los conocimientos, las técnicas y los conceptos que son necesarios para crear programas para la familia PC, no sólo para un miembro de la familia, aunque podríamos quizá atender a las peculiaridades de un miembro, sino para toda la familia como un todo, de una forma que sea tan universal que nuestros programas podrían trabajar no sólo con todos los miembros actuales de la familia, sino también con futuros miembros.

He escrito este libro para cualquiera que esté relacionado con el desarrollo de programas para la familia PC. Es para programadores, pero no sólo para programadores. Es para cualquiera que esté relacionado o necesite conocer los detalles técnicos y las ideas de trabajo que son las bases del desarrollo de programas para el PC, incluyendo a cualquiera que dirija a programadores, cualquiera que

planee o diseñe programas del PC y cualquiera que use programas del PC y quiera conocer detalles sobre ellos.

ALGUNOS COMENTARIOS SOBRE LA FILOSOFIA

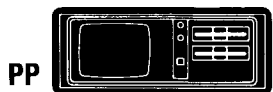
Uno de los elementos más importantes de este libro es el que trata sobre la *filosofía* de la programación. Encontrará a lo largo de este libro explicaciones de las *ideas* fundamentales del diseño de la familia PC de IBM, y los principios de una programación fiable de los PC, vistos desde mi propia experiencia.

Si este libro fuera a proporcionarle únicamente hechos —información técnica en forma de tablas— no le sería útil. Esto es debido a que he entretelado con el planteamiento técnico una explicación de lo que pasa con la familia PC, de los principios que unen a los diferentes miembros de la familia y de las técnicas y métodos que nos ayudan a producir programas que pueden perdurar y prosperar junto a la familia PC.

COMO USAR ESTE LIBRO

Este libro es al mismo tiempo un libro de lectura y un libro de consulta, y hay al menos dos formas en las que puede ser utilizado. Usted puede desear leerlo como cualquier otro libro, desde el principio hasta el fin, profundizando en donde el planteamiento le sea útil y de una rápida lectura pasar el material que todavía no necesita. Esta forma proporciona un gran conocimiento general de los funcionamientos y de las ideas sobre la explotación de programas del PC. Usted puede también usar este libro como simple referencia, buscando información específica en capítulos específicos. Hemos proporcionado tablas de contenidos detalladas al principio de cada capítulo y un amplio índice para ayudarle a encontrar lo que necesite.

Cuando use este libro como guía de acceso aleatorio a los detalles de la programación de los PC, se dará cuenta que la mayor parte del material está íntimamente interrelacionado. Para ayudarle a hacer frente a estas interrelaciones, verá que he repetido algunos detalles cada vez que viene a cuento duplicar la información, y he usado un símbolo  para remitirle a otras secciones cuando no era práctico duplicar la información. También he usado los siguientes símbolos autoexplicativos para ayudarle a encontrar el material que es específico de una máquina en particular:

y **PC** el PC originaly **XT** el PC XTy **JR** el PCjry **PP** el PC portátily **AT** el PC AT

Los símbolos de máquina se sitúan sobre los párrafos y secciones completas que se refieren a una máquina específica, mientras que las iniciales se usan para llamar la atención sobre comentarios referentes a una máquina específica dentro de un tema que se refiere a la familia al completo. Espero que este sistema le permitirá acotar más fácilmente la información que necesite para sus programas.

OTRAS FUENTES

Un libro, por supuesto, no puede proporcionarle todos los conocimientos que posiblemente necesita. He hecho este libro tan rico y completo como he podido, pero siempre habrá necesidad de otro tipo de información. Aquí tiene algunos de los sitios donde puede buscar material para complementar lo que encuentre en este libro.

Para información técnica detallada sobre la familia PC, la fuente fundamental es la serie de manuales de Referencia Técnica de IBM. Hay manuales de Referencia Técnica específicos del PC original, del XT, del PCjr, del AT y de otros modelos específicos. La mayoría de la información relacionada con la programación en estos manuales está esencialmente repetida, y cualquier manual podría servir como referencia para la familia entera. Usted debe conocer ciertas cosas sobre estos manuales que son específicas de un modelo: Primero, la información que es particular de un modelo no está diferenciada de la familia completa. Para asegurarse de las diferencias se debe de usar el sentido común, comparar los diferentes manuales y consultar

este libro. En segundo lugar debe recordar que cada nuevo modelo de PC añade nuevas características. Si acude al manual para un modelo reciente de PC, encontrará información sobre una amplia variedad de características; si acude al manual para uno de los primeros modelos, evitará la distracción de las características que no se aplican a todos los modelos de la familia.

Hay también un manual de Referencia Técnica de Opciones y Adaptadores para las diversas opciones y adaptadores, tales como diferentes unidades de disco o monitores, usados por la familia PC. La información técnica sobre este tipo de equipo está agrupada en ese libro, que es puesto al día periódicamente (las puestas al día se pueden obtener por suscripción). Mucha de la información en este manual de Referencia Técnica no es para el uso de programadores, pero encontrará algunas partes que sí lo pueden ser.

IBM también publica manuales de Referencia Técnica para ampliaciones especiales del PC, como el PC Network.

Quizá el más importante de los manuales de Referencia Técnica es la serie del DOS. Estos manuales contienen una información técnica abundante y detallada que yo he resumido en este libro. Si usted se encuentra con que necesita más detalles específicos sobre la operación del DOS, debe acudir a este manual.

Además de estos manuales de IBM, hay otros sitios a donde acudir para obtener más información. Para una perspectiva un poco más amplia sobre el Ordenador Personal de IBM, no está enfocado a la programación, lea *El IBM PC a fondo*, publicado también por Anaya Multimedia. Para una perspectiva similarmente amplia sobre el DOS, lea el libro de Van Wolvertón's *El libro del MS-DOS*, de esta misma colección. Para más detalles sobre las peculiaridades y los pros y los contras del PCjr vea mi *Exploring the PCjr*, publicado por Microsoft Press.

Debido a que este libro cubre el tema de la programación de PC de un modo amplio, no puede proporcionarle más que unos pocos detalles clave sobre lenguajes de programación individuales. Para cualquier lenguaje de programación en particular, y para los muchos compiladores específicos de estos lenguajes, necesitará más libros de los que podría nombrar o recomendar.

Habiendo completado estas observaciones introductorias, ¡es hora de zambullirse en nuestra tarea de dominar los principios de la programación de la familia PC!

Anatomía del PC

El microprocesador 8088 2

El microprocesador 80286 6

El coprocesador matemático 8087 8

Los *chips* de soporte 9

El controlador de interrupciones 8259 10

El controlador de DMA 8237A 11

El generador de reloj 8284A 11

El interfaz de periféricos programable 8255 12

El reloj programable 8253 12

El controlador de CRT 6845 12

El controlador de disco PD765 12

La conexión entre componentes: el *bus* 13

El *bus* de direcciones 13

El *bus* de datos 13

Los *chips* de memoria 15

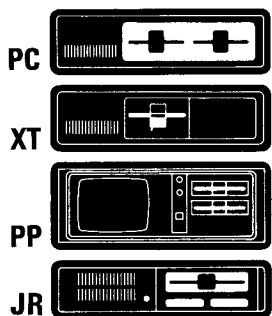
Filosofía de diseño 18

Desde el punto de vista del programador, todos los miembros de la familia PC contienen un procesador, *chips* de memoria, y varios circuitos integrados inteligentes o programables. Todos los componentes del circuito principal, que necesita el ordenador para trabajar, están localizados en la placa del sistema; otra parte muy importante se encuentra en las tarjetas de expansión, que pueden conectarse en la placa del sistema.

La placa del sistema contiene el microprocesador (bien el 8088 o el 80286) junto con al menos 64 Kbytes de memoria, algunos programas internos en ROM, tales como el BASIC y la ROM-BIOS, y varios *chips* de soporte muy importantes. Estos *chips* controlan dispositivos externos, como la pantalla o el controlador del disco, y realizan algunas otras ayudas al microprocesador durante la realización de sus tareas.

En esta sección vamos a describir los *chips* principales y algunas de sus características más importantes. Los símbolos que irán apareciendo en el margen le indican qué miembros utilizan cada *chip* dentro de la familia PC. De esta forma se puede dar cuenta, al repasar la literatura técnica, que estas pastillas tienen frecuentemente más de un nombre asociado. Estos sinónimos los mencionaremos a lo largo de nuestra discusión. Por ejemplo, algunos de los periféricos del PC, tal como el teclado, están bajo la supervisión de una pastilla (*chip*) conocido como el 8255. La forma completa de su nombre es 8255-A, pero también recibe el nombre de su función, interfaz programable de periféricos o PPI. Ambos nombres se refieren a la misma parte del circuito del PC.

EL MICROPROCESADOR 8088



El 8088 es el microprocesador de 16 bits que controla a los ordenadores personales estándar IBM, incluyendo el original PC, el XT, el PC portátil y el PCjr. Es la unidad central del proceso (CPU) del ordenador —el cerebro que hay detrás de la máquina—. Cada bit de datos que entra o sale del ordenador pasa a través de la CPU para ser procesado, o redirigido.

El 8088 controla las operaciones básicas del ordenador, envía y recibe señales de control, direcciones de memoria y datos, de una parte del ordenador a otra, a través de la red electrónica de interconexiones llamada *bus*. A lo largo del *bus* se encuentran los puertos de entrada y salida (E/S), las diversas memorias y los *chips* de soporte. Los datos pasan a través de estos puertos de E/S y viajan a la CPU a las otras partes del ordenador, o viceversa.

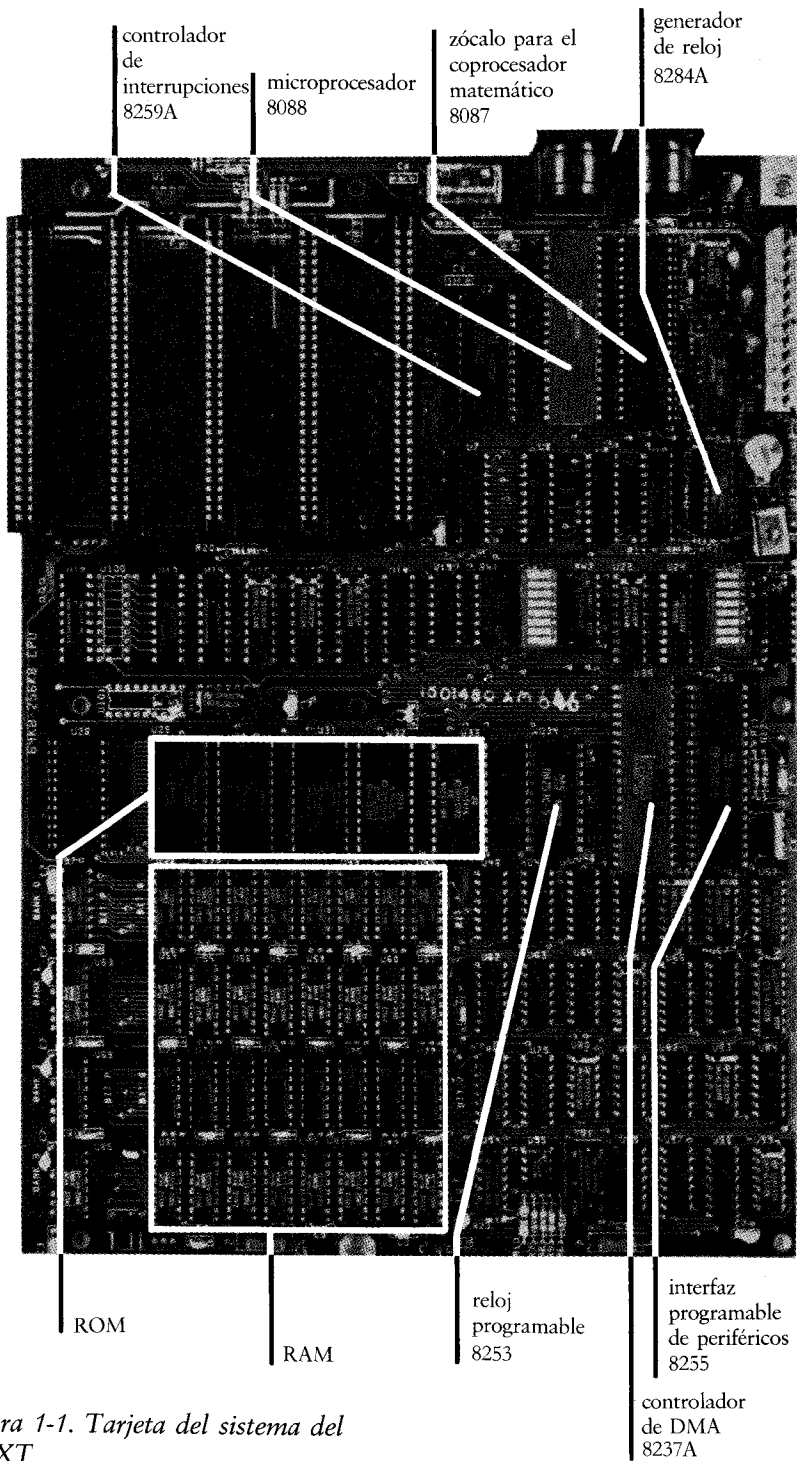


Figura 1-1. Tarjeta del sistema del PC/XT

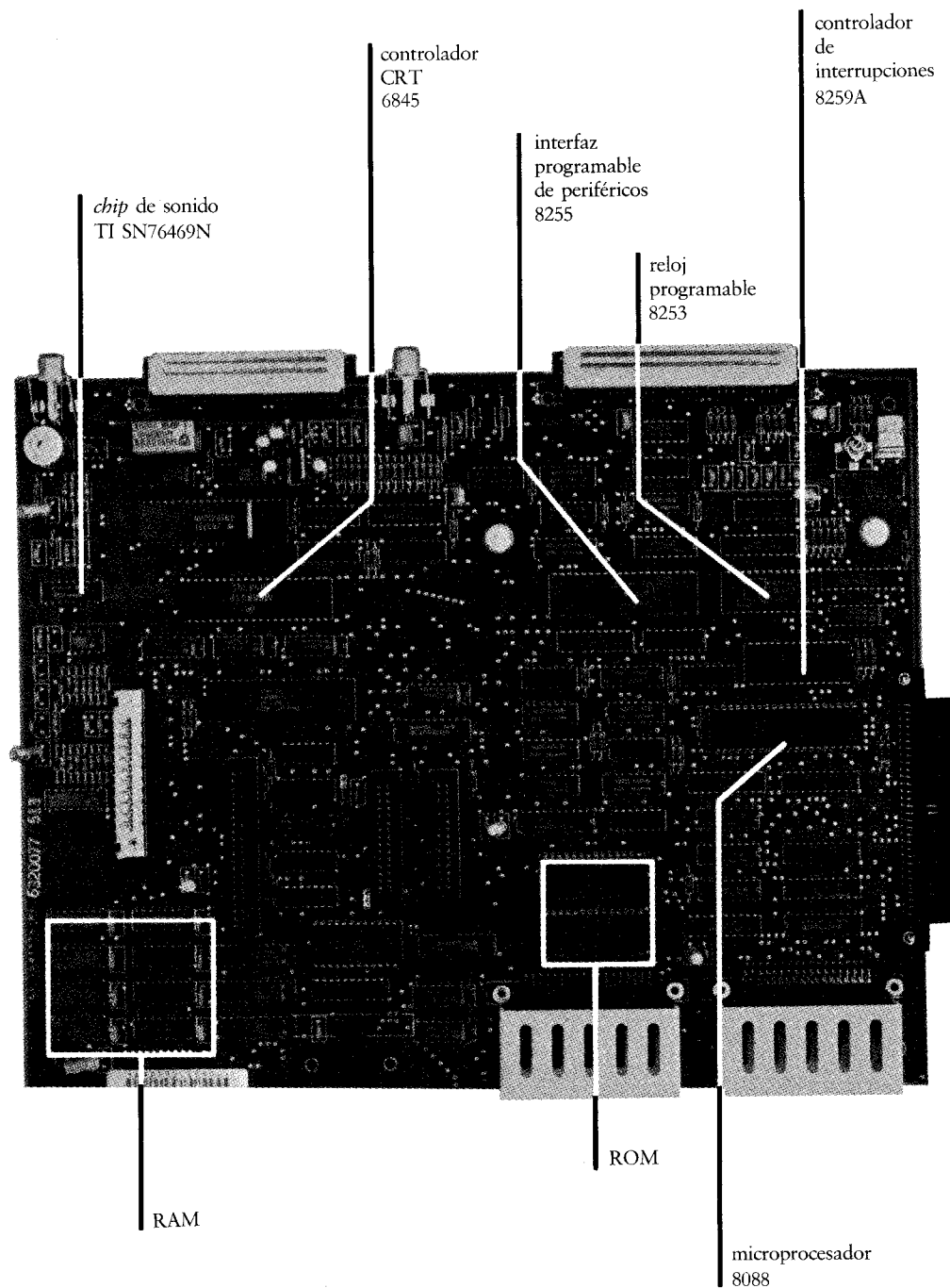


Figura 1-2. Tarjeta del sistema del PCjr

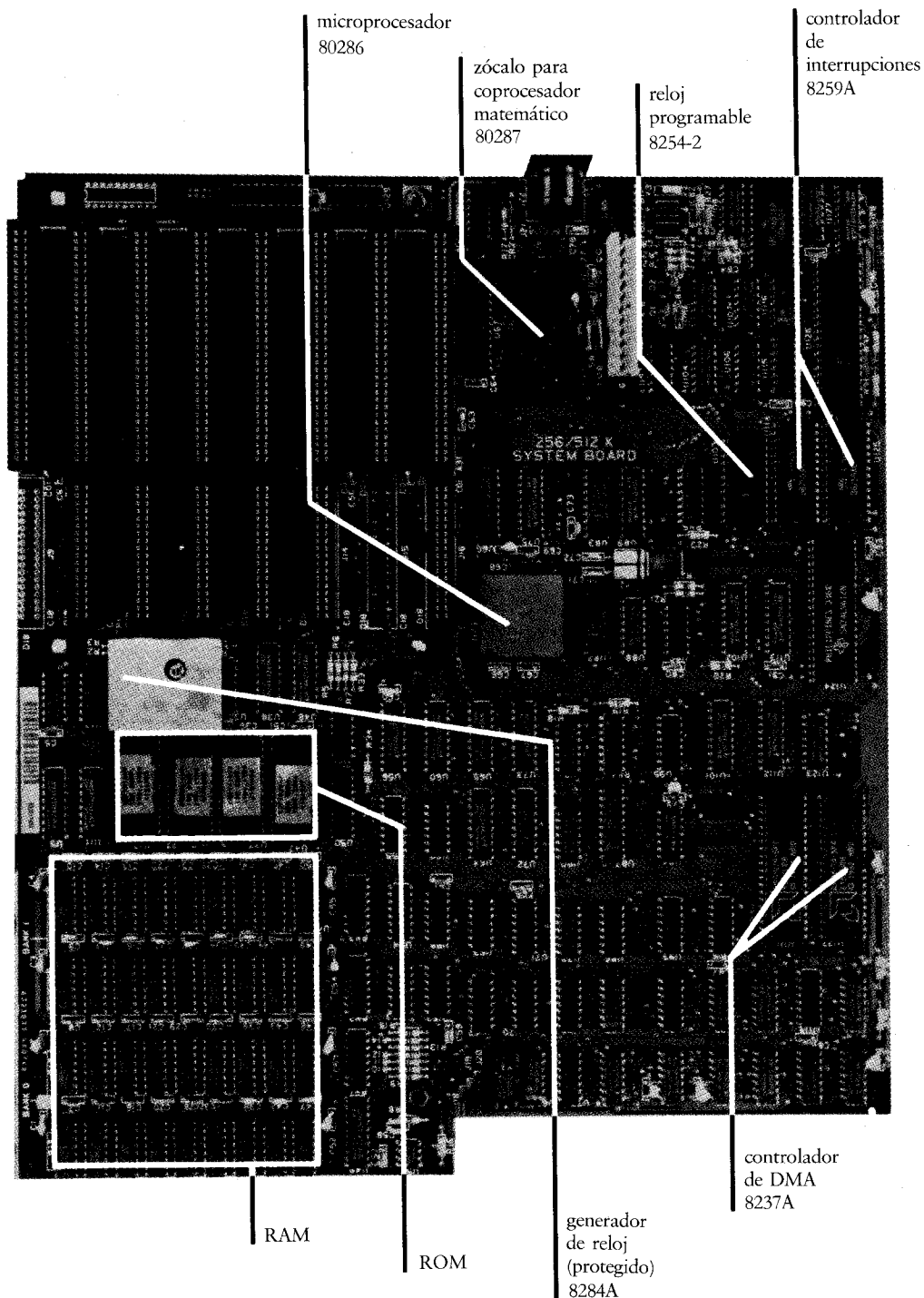


Figura 1-3. Tarjeta del sistema del AT

Dentro del 8088 existen 14 registros, que le proporcionan una área de trabajo adecuada para poder realizar la transferencia de los datos y su procesamiento. Estos registros internos constituyen una memoria interna de 28 bytes de tamaño, y pueden almacenar datos, direcciones de memoria, punteros de instrucción e indicadores de estado y control. A través de estos registros, el 8088 puede acceder a más de un millón de bytes de memoria y 64K puertos de E/S. Véase el capítulo 2 para encontrar un resumen de las características de funcionamiento del 8088.

Arbol de la familia 8088. El 8088 es un miembro de una familia estrechamente relacionada de microprocesadores de 16 bits, diseñados por Intel Corporation. El miembro fundador de esta familia de microprocesadores es el 8086. El 8088 difiere del 8086 sólo en un pequeño pero importante aspecto: aunque el 8088 es un microprocesador de 16 bits, utiliza un *bus* de datos de 8 bits en lugar del *bus* de 16 bits que utiliza el 8086. (La diferencia entre los *buses* de 8 y 16 bits se discute en la página 14.) Virtualmente, cualquier cosa que pueda leer sobre el 8086 también es aplicable al 8088. El 8088 ha sido durante mucho tiempo el principal cerebro para la familia PC, aunque no es el único disponible. Otros microprocesadores de Intel están siendo utilizados en algunos de los primos distantes de la familia PC. Por ejemplo, el 8086 es el cerebro del Compaq Deskpro, un ordenador compatible PC portátil bien conocido. Un par de microprocesadores conocidos, como el 80188 y 80186 (que se denominan usualmente 188 y 186), son las versiones más avanzadas de los microprocesadores originales 8088 y 8086, y se han utilizado con gran variedad de ordenadores relacionados todos con la familia IBM PC, como, por ejemplo, el ordenador Tandy 2000. Estos dos microprocesadores tienen más potencia de cálculo que sus precedentes, pero su recurso principal es que combinan, en un *chip*, el microprocesador y algunas de las operaciones de soporte más necesarias e importantes, que normalmente son manejadas externamente. Pero, a pesar de sus progresos, el 186 y el 188 no son todavía lo último en cuanto concierne a la familia 8086.

El microprocesador 80286

AT



El microprocesador del Intel más avanzado hasta el momento y más utilizado en los ordenadores personales de IBM es el 80286 (o 286). Este es el *chip* que controla la operación del AT. El 80286 es un verdadero microprocesador de 16 bits, que utiliza un *bus* completo de datos de 16 bits y añade características de programación extra al diseño del 8086. Quizá las mejoras más importantes del 286 son su

posibilidad de permitir realizar trabajo multitarea y almacenamiento en memoria virtual —dos conceptos familiares a alguien que esté familiarizado en el trabajo en grandes ordenadores.

Multitarea es la posibilidad de que una CPU pueda realizar varias tareas a la vez —tales como imprimir un documento o hacer cálculos con hoja electrónica— cambiando rápidamente su atención entre los programas que controla. Un PC normal, que utiliza el microprocesador 8088, puede hacer multitarea de una forma limitada con la ayuda de un *software* muy sofisticado, tal como los programas Topview o Microsoft's Windows, de IBM. Pero un verdadero procesador multitarea, como el 286, realiza tareas conmutando internamente, con alguna ayuda del sistema operativo. Puesto que las capacidades multitarea en el 286 son, en gran manera, una parte del diseño del *hardware*, son mucho más rápidas y más fiables que la multitarea conducida por *software*.

La **memoria virtual** permite que un ordenador actúe como si dispusiera de mucha más memoria que la que realmente tiene. A través de un diseño de *software* y *hardware* extremadamente sofisticado, un programa puede llegar a creer que tiene hasta un gigabyte (un millón de Kbytes) de memoria a su disposición, incluso aunque los *chips* de memoria del *hardware* representen sólo una fracción de aquel tamaño. Este engaño se consigue a través de un elaborado esquema de direccionamiento de memoria, que implica el almacenamiento de algunas partes del programa en el disco y, parte, en la memoria principal. Cuando se necesitan instrucciones particulares, o datos de programa que no están en la memoria física, se buscan y se traen desde el disco. El 286 y el sistema operativo tienen la pesada tarea de resolver dónde está la información y dónde debe de ir, de forma que el programa se ejecute fácil y eficazmente, incluso en el caso de que esté disperso a través del sistema.

La memoria virtual se ha utilizado en ordenadores, minis y grandes, durante mucho tiempo, pero sólo recientemente se ha introducido en el mundo de los microordenadores. Su introducción, a través del 286 en el AT, producirá un profundo efecto en la programación de aplicaciones, puesto que permite escribir programas, cuyos tamaños estén, para todos los propósitos prácticos, limitados sólo por la capacidad física de los discos.

El AT se ve a menudo tan sólo como el miembro más rápido y potente de la familia 8086 —con la posibilidad de correr casi todos los programas populares del PC, incluyendo el sistema operativo del DOS y casi todos los programas DOS—. Sin embargo, el trabajo multitarea y el almacenamiento en memoria virtual cambian las características operativas del 286. Cuando se utilizan estas caracterís-

ticas, el AT se convierte en un ordenador diferente, requiriendo programas diferentes y un sistema operativo diferente. Esto hace del AT el primero de una generación enteramente nueva de ordenadores personales, una generación diferente de la familia PC original.

Con esto en mente, lo mejor es dejar la discusión de las avanzadas capacidades del AT para otro libro y enfocar en éste las capacidades del PC estándar. Encontrará que la mayoría de las técnicas de programación discutidas en este libro se enfoca al 8088, con anotaciones sobre el 80286, donde sea necesario.

Todos los miembros de la familia 8086 están diseñados para trabajar con procesadores adicionales. También trabajan con dos coprocesadores especiales: el matemático 8087 y el E/S 8089. Estos *chips* opcionales pueden conectarse juntos para ayudar a reducir la carga de trabajo de la CPU principal. IBM sólo proporciona la circuitería para soportar el coprocesador matemático 8087. Empleemos unos momentos en discutir este *chip* con más detalle.

El coprocesador matemático 8087

El 8088 sólo puede trabajar con números enteros. Los números «reales», o de punto flotante, se manipulan de forma especial, normalmente mediante subrutinas, que llevan a cabo las operaciones con efectividad suficiente, pero con un gran coste de eficacia y velocidad.

El coprocesador matemático 8087 realiza los cálculos en punto flotante entre 10 y 50 veces más rápidamente que el 8088, alcanzando en las operaciones aritméticas un grado de precisión más elevada que la conseguida normalmente por el 8088 e incluso si se compara con grandes ordenadores de costo mucho mayor. El 8087, además de realizar las operaciones aritméticas suma/resta/multiplicación/división, es capaz de realizar cálculos trigonométricos (seno, coseno, tangente, etc.), lo que permite simplificar parte de la compleja programación que suele ser necesaria. A esto se le añade el hecho de que puede trabajar con números que vengan en diferentes formatos, incluyendo enteros, coma flotante y formatos decimales, ejecutando su labor, mientras que el 8088 realiza otro trabajo.

En todos los modelos basados en el 8088, excepto el PCjr, puede acomodarse al 8087, aunque requiere un *software* especial. (AT El AT utiliza el coprocesador matemático 80287, que es una variante del 8087 que está adaptada para trabajar con el microprocesador 80286.) Pero aunque el 8087 aumenta en gran manera la capacidad de realizar operaciones aritméticas, en los ordenadores personales IBM existe relativamente poco *software* que obtenga ventajas de él. Esta

situación desafortunada se debe a un hecho histórico: aunque el uso del 8087 estaba previsto en el diseño del modelo original del PC (y en los otros modelos), IBM no lo introdujo hasta que el *hardware* y el *software* para el PC quedaron bien establecidos, lo que hizo que un gran porcentaje del *hardware* y *software* original que se diseñó para la familia PC no incorporase el uso del 8087, privándose de su extraordinaria potencia de cálculo.

Tipo de dato	Rango aproximado		Bits	Dígitos significativos (decimal)
	(desde)	(hasta)		
Entero palabra	-32 768	+32 767	16	4
Entero corto	$-2 \times 10E9$	$+2 \times 10E9$	32	9
Entero largo	$-9 \times 10E18$	$+9 \times 10E18$	64	18
Decimal condensado	-99...99	+99...99	80	18
Real corto	$8.43 \times 10E-37$	$3.37 \times 10E38$	32	6-7
Real largo	$4.19 \times 10E-307$	$1.67 \times 10E308$	64	15-16
Real provisional	$3.4 \times 10E-4932$	$1.2 \times 10E4932$	80	19

Figura 1-4. Rango de los diferentes tipos de datos numéricos que se pueden operar en los ocho registros de 80-bits del coprocesador matemático 8087 ó 80287

Aunque los *chips* y el *software* para el 8087 se han vendido hasta ahora en cantidades limitadas, estamos empezando a ver más programas, tales como el Framework de Ashton-Tate, que no sólo aprovechan las ventajas del 8087, sino que también detectan su presencia, y automáticamente lo utilizan o no dependiendo de los requerimientos del programa. Desafortunadamente, sólo hay unos pocos programas de este tipo disponibles todavía.

Como la utilización del coprocesador matemático en la familia PC es raro, no trataremos en este libro los problemas especiales que aparecen a la hora de programarlo.

LOS CHIPS DE SOPORTE

El microprocesador no puede controlar el ordenador entero sin recibir ayuda. Para ello delega algunas de las funciones de control a otros *chips*, y de esta forma la CPU se encuentra libre para atender su propio trabajo. Estos *chips* de soporte se responsabilizan de procesos,

como, por ejemplo, controlar el flujo de información dentro del ordenador, como lo hacen los controladores de interrupciones y de DMA, o el flujo de información con un dispositivo externo que está unido al ordenador, como una pantalla de video, o una unidad de disco. Estos dispositivos, que también reciben el nombre de *controladores de periféricos*, están a menudo en tarjetas separadas que se insertan en los conectores de expansión de los PC.

Algunos de los *chips* de soporte del IBM PC son programables, lo que significa que se pueden manipular para que realicen tareas especializadas. En la mayor parte de las ocasiones no es una buena idea realizar la programación directa de estos dispositivos, pero, en la discusión que sigue de cada *chip*, señalaremos cuáles están libres de riesgos al programar y cuáles no lo están. Como este libro no cubre el objetivo de realizar el control directo del *hardware*, deberá mirar en el manual IBM Technical Reference para encontrar los detalles relacionados con la programación de cada uno de los *chips*.

El controlador de interrupciones 8259

El 8259 supervisa la realización de las interrupciones. Las interrupciones son señales enviadas a la CPU por el *hardware* para requerir su atención, o responder a alguna acción. Este *chip* intercepta las señales, determina su nivel de importancia en relación a otras señales que se estén recibiendo y activa una señal de interrupción de la CPU en función de su decisión. Cuando la CPU recibe la señal de la interrupción, llama a un programa específico asociado con el dispositivo periférico que está involucrado en el proceso. Es este programa el que llevará a cabo la acción requerida. ■ Discutiremos las interrupciones más adelante en los capítulos 2 y 3.

El 8259 puede manejar ocho solicitudes de interrupción a la vez y puede unirse a otros 8259 para aumentar la capacidad. En el **AT** IBM se ha hecho uso de esta capacidad de expansión, conectando dos de ellos juntos, de tal forma que puedan manejar quince interrupciones a la vez.

Generalmente no se programa el 8259; cuando alguna operación lo hace, es probable que interfiera en las operaciones básicas del ordenador. Sin embargo es posible reconfigurar los niveles de prioridad de las interrupciones en algún momento durante la ejecución del programa principal. Esto significa que el programa puede cambiar el orden en que son procesadas las solicitudes de interrupción por el 8259 para atender a sus propias necesidades.

Otros nombres para el 8259: el 8259A, el INTR y el PIC (que proviene de controlador de interrupciones programable).

El controlador de DMA 8237A

Para evitar saturar al microprocesador, algunas partes del ordenador pueden transferir datos a la memoria, o viceversa, del ordenador sin pasar a través de la CPU. Esta operación se llama acceso directo a memoria, y se controla mediante un *chip* conocido como el 8237A, o controlador de DMA (Direct memory access). El propósito principal del DMA es permitir al controlador del disco que lea, o escriba, datos sin involucrar al microprocesador. Como las operaciones de E/S desde el disco son relativamente lentas, el DMA puede aumentar un poco las prestaciones del ordenador.

Todos los miembros de la familia PC, con la importante excepción del PCjr, utilizan el 8237A, o su equivalente, para el acceso directo a memoria. **JR** La razón de la carencia del DMA en el PCjr es porque es más pequeño que sus primos. Sin un controlador DMA para ayudar, el 8088 del PCjr tiene que realizar las operaciones del disco siempre que sea necesario, que es, indirectamente, razón por la cual no se puede escribir en el teclado del PCjr mientras se está usando el disco.

El controlador DMA contiene cuatro canales separados para llevar datos y traerlos a la memoria, y 344 bits de memoria interna para almacenar los datos que están en tránsito. Teóricamente es posible conectar varios controladores de DMA unos con otros y, en efecto, **AT** el AT utiliza dos controladores DMA en su circuitería.

El generador de reloj 8284A

El generador de reloj proporciona las señales de temporización multifase que son necesarias para el funcionamiento del microprocesador y de los periféricos. Su frecuencia base es de 14.3128 megahercios (MHz, o millones de ciclos por segundo). Los otros *chips* generalmente dividen la frecuencia base por una constante, con objeto de obtener la frecuencia que necesitan para efectuar sus tareas. La familia PC estándar del 8088 funciona con 4.77 MHz, que es un tercio de la frecuencia base. Los nuevos elementos de la familia 8086 pueden funcionar de forma más rápida. Por ejemplo, el 8088-2, que se utiliza en algunas versiones de PC, puede funcionar a una velocidad de reloj de 8 MHz, incrementando en cerca de dos veces la potencia de cálculo bruta del 8088, y el 80286 actúa a 6 MHz, incrementando aproximadamente una vez y media la potencia de cálculo del 8088. El *bus* interno y el reloj programable 8253 (● que se discutirán brevemente) utilizan una frecuencia de 1.193 MHz, funcionando a un cuarto de la velocidad del 8088, y a un doceavo de la velocidad del reloj base.

El interfaz de periféricos programable 8255

El 8255 se utiliza para conectar los dispositivos periféricos del ordenador al *bus*. La información que se envía a dispositivos tales como el altavoz y el cassette, o viceversa, viaja a través de los puertos E/S vía este *chip*. El 8255 también se denomina 8255A-5 y PPI (por interfaz de periféricos programable). Normalmente se programa mediante *software*, y aunque es programable, no es necesario programarlo personalmente.

El reloj programable 8253

El 8253 (**AT** el 8254A en el AT) es un reloj de propósito múltiple y un controlador que puede generar tres tiempos de retardo exactos, bajo control de programa. Obtiene su señal del generador de reloj 8284A, y está diseñado para oscilar con una frecuencia de 1.190 MHz.

El 8253 se utiliza principalmente para generar sonidos en el altavoz interno del PC, pero también se utiliza para otras funciones que dependen de la frecuencia, tales como la E/S de datos al cassette y cronómetro. ● Véase la discusión sobre el sonido del capítulo 7 para obtener más información sobre este *chip*.

Otros nombres del 8253 son 8253-5, contador de tiempo, temporizador (*timer*), y a veces reloj. Hay que tener en cuenta que la palabra «reloj» también se refiere al *chip* 8284, que genera el verdadero latido del ordenador de 14.3 MHz.

El controlador de CRT 6845

El 6845 también se denomina CRT Motorola y se localiza generalmente en una placa de expansión conocida como adaptador de video. Tiene 19 registros, que se utilizan para definir y controlar el barrido de un CRT. Aunque es programable, resulta conveniente dejarlo bajo control del BIOS del PC. ● Véase el capítulo 4 para obtener más información sobre monitores y adaptadores de video.

El controlador de disco PD765

El PD765 supervisa y controla la operación del encargado de gestionar el disco. Se denomina comúnmente FDC (controlador de disco flexible) o controlador NEC (*Nippon Electric Company*). Como en el caso del controlador de CRT 6845, deberíamos dejar este *chip* bajo control del BIOS y no tocarlo nunca.

LA CONEXION ENTRE LOS COMPONENTES: EL BUS

Como ya hemos mencionado, la familia de ordenadores PC interconexiona toda la circuitería de control interna mediante un diseño de circuito, conocido con el nombre de *bus*. Un *bus* es simplemente un conjunto compartido de pistas trazadas en la placa de circuito principal, al que se conectan todas las partes que controlan y forman el ordenador. Cuando un dato pasa de un componente a otro, viaja a lo largo de este camino común para alcanzar su destino.

Cada *chip* de control y cada byte de memoria del PC están conectados directa o indirectamente al *bus*. Cuando un nuevo componente se inserta en uno de los conectores de expansión, queda unido directamente al *bus*, convirtiéndose en un objeto más de la unidad completa.

Cualquier información que entra o sale de un sistema ordenador se almacena temporalmente en al menos una de las distintas localizaciones que existen a lo largo del *bus*. La mayor parte de las veces el dato se sitúa en la memoria principal, que en la familia PC está formada por miles de posiciones de memoria de 8 bits. Pero algún dato puede acabar en un puerto, o registro, durante unos instantes, mientras espera que la CPU lo envíe a una posición adecuada. Generalmente, los puertos y registros almacenan sólo uno o dos bytes de información a la vez, y se utiliza normalmente como lugares de parada intermedia para los datos, que se están enviando de un lugar a otro. (● Los puertos y registros se discuten con detalle en el capítulo 2.)

Siempre que se utiliza una posición de memoria, o un puerto, como lugar de almacenamiento, su localización está marcada por una dirección que la identifica individualmente. Cuando el dato está listo para ser transferido, se transmite primero su dirección de destino por el *bus* de direcciones; el dato sigue a la zaga por el *bus* de datos. Por tanto, el *bus* transporta algo más que datos. Lleva información de control, tales como las señales de temporización (del sistema reloj), las señales de interrupción, así como las direcciones de las miles de posiciones que forman tanto la memoria como los dispositivos que están conectados al *bus*. Para diferenciar estas cuatro funciones diferentes, el *bus* está dividido en cuatro partes: líneas de potencia, *bus* de control, *bus* de direcciones y *bus* de datos. Vamos a ahondar más profundamente en los *buses* de datos y de direcciones, porque conducen la información de una forma que ayuda a explicar algunas de las propiedades particulares de la familia PC.

El bus de direcciones

El *bus* de direcciones de la familia estándar PC utiliza 20 líneas de señales para transmitir las direcciones de las posiciones de memoria y de los dispositivos que están conectados al *bus*. (El direccionamiento de la memoria se discute de forma más detallada en el capítulo 3.) Como hay dos valores posibles que pueden viajar a lo largo de cada una de las 20 líneas de direcciones, los ordenadores estándar PC pueden representar 2^{20} direcciones. Esta cantidad cubre aproximadamente un millón de direcciones posibles. **AT** El AT utiliza 24 líneas de direcciones, permitiendo especificar 2^{24} , o la que representa aproximadamente 16 millones de direcciones distintas.

El bus de datos

El *bus* de datos trabaja junto con el *bus* de direcciones para transportar los datos a través del ordenador. El sistema basado en el 8088 del PC utiliza un *bus* de datos que tiene 8 líneas, cada una de las cuales lleva un único dígito binario (bit). Esto significa que el dato transmite a través del *bus* de 8 líneas en unidades de 8 bits (byte). **AT** El microprocesador 80286 del AT utiliza un *bus* de datos de 16 bits, y, por consiguiente, pasa datos en unidades de 16 bits (una palabra).

El 8088, al ser microprocesador de 16 bits, puede trabajar con 16 bits de datos a la vez, justo como su análogo el 80286. Aunque el 8088 puede trabajar internamente con números de 16 bits, cuando trabaja con la circuitería que le rodea, sólo pasa datos de 8 bits cada vez, debido al tamaño de su *bus* de datos; por ello, algunas personas han llegado a comentar que el 8088 no es de verdad un microprocesador de 16 bits. Queden seguros de que lo es, aunque es menos potente que el 80286. El *bus* de datos de 16 bits del 80286 ayuda a que se muevan los datos de manera más efectiva que con el 8088, pero la diferencia real en velocidad entre el 8088 y el AT viene de la mayor rapidez del reloj del AT y su potente organización interna.

Hay una importante razón práctica por la que algunos ordenadores, incluyendo los miembros más antiguos de la familia PC, utilizan el 8088 con su *bus* de datos de 8 bits en vez del 8086 con su *bus* de 16 bits. La causa es simplemente económica. Existe una multitud de elementos de circuitería de 8 bits que se encuentran disponibles en grandes cantidades a bajos precios. Cuando se estaba diseñando el PC, la circuitería de 16 bits era más cara y se encontraba menos disponible. La utilización del 8088, en vez del 8086, fue importante no sólo para bajar el coste del PC, sino también para evitar la escasez de las piezas. El precio de los elementos de circuitería de 16 bits ha

decrecido significativamente desde entonces; por tanto, ya es factible económicamente utilizar el 80286, que es más eficaz al contar con un *bus* de 16 bits. Además, el 286 puede utilizar una mezcla de componentes de 8 y 16 bits, manteniendo de este modo la compatibilidad con la familia PC.

LOS CHIPS DE MEMORIA

Hasta ahora hemos discutido sobre la CPU, los *chips* de soporte y el *bus*, pero no se ha tocado para nada a la memoria. Hemos dejado la discusión sobre ella para el final de este capítulo, porque los *chips* de memoria, a diferencia de los otros *chips*, no controlan o dirigen el flujo de la información a través de un sistema computador, sino que simplemente almacenan la información hasta que se necesita.

El número de *chips* de memoria que existe físicamente dentro del ordenador determina la cantidad de memoria que pueden ocupar los programas y los datos. Aunque ésta puede variar de un ordenador a otro, un PC estándar viene aproximadamente con 40K de memoria del tipo sólo lectura (ROM) — con espacio para más — y de 128K a 256K de memoria de tipo acceso aleatorio (RAM). Como sólo caben 256K de RAM en la tarjeta del sistema, es posible añadir tarjetas de memoria para variar la capacidad de almacenamiento, vía los conectores de expansión del PC. Para el usuario, éste es el aspecto físico de la memoria del PC. Para el ordenador, los *chips* de memoria no son más que unas pocas miles de posiciones de almacenamiento (1 byte), cada una de las cuales tiene su propia dirección asociada.

Los programadores deben de imaginarse la estructura de la memoria de esta forma, no en términos de cuánta memoria física hay, sino en términos de cuánta memoria *direccionable* hay. El 8088 puede direccionar hasta 1.024K, o exactamente 1.048.576 bytes de memoria. En otras palabras, éste es el máximo número de direcciones y, por tanto, el máximo número de bytes individuales de información a los que puede referirse. El direccionamiento de la memoria se discute con más detalle en la página 26.

Cada byte está referenciado mediante una dirección numérica de 20 bits. En el esquema de memoria del 8088, las direcciones son de 20 bits de «anchura», porque tienen que viajar a través de un *bus* de direcciones de 20 bits. Se tiende a utilizar la notación hexadecimal más que la binaria cuando se quiere representar alguna de las posiciones de memoria; de esta forma se traducen los 20 bits que forman la dirección a 5 dígitos hexadecimales, que son completamente equivalentes. Por tanto, los valores de dirección permitidos están en un rango que va desde la 00000 hex hasta la FFFFF hex (es decir,

de 0 a 1.048.576 en notación decimal). Si tiene problemas para comprender la notación hexadecimal, puede echar un vistazo rápido al apéndice B.

Cuando se trata el espacio de memoria direccionable del PC de 1024 bytes, se divide en 16 bloques de 64K bytes cada uno, identificando cada bloque de 64K por el primer dígito hexadecimal, o la parte de mayor orden, de todas las direcciones de memoria que componen el bloque. Por ejemplo, el primer bloque de 64K de memoria es el 0, con los bytes cuyas direcciones van de la 00000 a la 0FFFF; el último bloque de memoria es el bloque F, cuyas direcciones van de la F0000 a la FFFFF.

Desde un punto de vista real no hay fronteras funcionales entre los diferentes bloques. Se hace esta división de la memoria en parte por conveniencia y en parte porque el esquema global formado por la memoria direccionable en todos los ordenadores personales IBM asigna usos específicos a cada bloque.

F 0 0 0 0	Area de ROM fija: ROM-BIOS, BASIC, diagnósticos
E 0 0 0 0	Area para cartuchos ROM
D 0 0 0 0	Area para cartuchos ROM
C 0 0 0 0	Extensiones al BIOS (Discos en el XT)
B 0 0 0 0	Memoria de pantalla convencional
A 0 0 0 0	Extensión de la memoria de pantalla
9 0 0 0 0	RAM de trabajo, hasta 640K
8 0 0 0 0	RAM de trabajo, hasta 576K
7 0 0 0 0	RAM de trabajo, hasta 512K
6 0 0 0 0	RAM de trabajo, hasta 448K
5 0 0 0 0	RAM de trabajo, hasta 384K
4 0 0 0 0	RAM de trabajo, hasta 320K
3 0 0 0 0	RAM de trabajo, hasta 256K
2 0 0 0 0	RAM de trabajo, hasta 192K
1 0 0 0 0	RAM de trabajo, hasta 128K; memoria máxima del PCjr
0 0 0 0 0	RAM de trabajo, hasta 64K; utilizados por el sistema

Figura 1-5. Esquema de los bloques de memoria para la familia PC

En teoría, cualquier área puede contener ROM permanentemente, o RAM cambiante. Por convenio, los primeros diez bloques (del 0 al 9), que totalizan 640K, se reservan para la RAM en los PC de IBM y se utilizan como memoria de trabajo ordinaria. La memoria que se va instalando en estos ordenadores se sitúa aquí, empezando por el

primer bloque. Como la RAM siempre ocupa un espacio contiguo, no se salta ningún bloque. Cualquier dirección de esta área de 640K, cuyo valor sea mayor que la memoria instalada, no se utiliza normalmente. Si un programa trata de utilizar una dirección donde no hay memoria, el resultado puede variar. Usualmente no detectará ningún error, y el programa continuará ejecutándose.

Todos los ordenadores personales IBM tienen por lo menos memoria instalada en el primer bloque de 64K, el bloque 0. Por supuesto, 64K es una cantidad mínima de memoria y es raro para un PC tener sólo esta cantidad. Las direcciones más pequeñas del bloque 0 se reservan tradicionalmente para que las utilice el *software* del sistema. Estas direcciones almacenan cosas tales como información de estado, tablas de direcciones, tablas de caracteres y rutinas del sistema operativo. El capítulo 3 explora estas localizaciones bajas de la memoria con mayor detalle.

El bloque A de memoria se reserva para ampliar la memoria del video y se utiliza por la EGA (adaptador gráfico mejorado) de IBM y por el Adaptador Gráfico Profesional de IBM. El uso de este bloque A es tan especial y tan caprichoso, que hay poca información provechosa y fiable que podamos contarle sobre él. La mejor forma de ver el bloque A es como un área de trabajo provisional que se utiliza durante breves momentos en algunos usos de video avanzados.

El bloque B se utiliza para la memoria de video ordinaria en todos los modelos, excepto en el PCjr. Está dividida en dos mitades, de 32K, cuyas direcciones empiezan en B0000 y B8000; por conveniencia, estas áreas se denominan simplemente como B0 y B8. El Adaptador Monocromo de IBM, tarjeta que controla el monitor monocromo, utiliza 4K de memoria, que se encuentran al principio del área B0 (los 28K que quedan del área no se utilizan). El Adaptador Color/Gráficos de IBM, tarjeta que gestiona al resto de los monitores, utiliza 16K de memoria, que se encuentran al principio del área (los restantes 16K no se utilizan).

Aunque los otros modelos del IBM PC tienen instalados uno o ambos de estos adaptadores de visualización, el PCjr tiene el equivalente funcional del adaptador color/gráficos implementado interiormente y no puede acomodar un adaptador monocromo. Además, el PCjr simula el uso del bloque B8, pero, en realidad, utiliza la parte alta del final de la RAM (el bloque 0 para el modelo inicial, y el bloque 1 para el modelo mejorado de 128K) para almacenar los datos del video. A pesar del circuito especial, denominado VGA (matriz de puertas para video), el Junior simula exactamente las funciones del video del PC, lo que permite que los programas piensen que están utilizando el bloque B, que es la localización típica para la



memoria de video del PC. El resultado es que un PCjr actúa como un PC que tenga instalado un Adaptador Color/Gráficos. Afortunadamente, el PCjr va lejos a la hora de disfrazar sus diferencias con el resto de los miembros de la familia PC, lo que desde un punto de vista de la programación significa que se pueden ignorar las peculiaridades del Junior y tratarlo como un PC estándar.

Es importante tener en cuenta que el uso del bloque B, la mitad B0 para monocromo y la mitad B8 para color/gráficos, es un estándar universal en la familia PC. Todos los modelos del PC, incluyendo PCjr, y todos los adaptadores de visualización, incluyendo el adaptador gráfico profesional y la EGA, utilizan, o al menos lo aparentan, las posiciones de memoria del bloque B de forma estándar.

El bloque C se dedica a recibir las ampliaciones que son necesarias de programas en ROM. IBM utilizó inicialmente esta área para almacenar las rutinas de la ROM-BIOS del disco fijo, que viene con el modelo XT (y que puede añadirse al modelo PC). Estas rutinas no se sitúan al principio, o al final, del bloque C, como cabría esperarse, sino que están situadas en el medio, empezando en C8. Se puede suponer que probablemente la mayoría de las ampliaciones del BIOS se colocará también en esta área general, particularmente aquella que soporta las nuevas ampliaciones del *hardware*.

Los bloques D y E se dedican a recibir las ampliaciones de memoria ROM en cartuchos con *software*. Estos cartuchos se pueden añadir a cualquier modelo PC, pero se usan raramente, excepto con el Junior. Estas ampliaciones de memoria se conectan al principio, o en la mitad de los bloques; en D0, D8, E0 o E8. Sólo en el PCjr los cartuchos se pueden conectar también en el siguiente bloque, en F0 o F8.

Normalmente, el bloque F se usa para los programas permanentes en ROM. Entre estos programas son la ROM del BASIC, la ROM-BIOS y las rutinas de comprobación y de diagnóstico. ➤ Véase el capítulo 3 para encontrar más detalles. **JR** El PCjr utiliza el bloque F para un propósito especial: conectar cartuchos en el bloque F para sustituir los programas de la ROM-BIOS convencionales que se encuentran situados allí. ➤ Puede encontrar más información sobre la utilización de los cartuchos del Junior en el capítulo 3.



FILOSOFIA DE DISEÑO

Antes que pasemos a los siguientes capítulos, debemos discutir la filosofía de control que hay detrás de la familia PC. Esto nos ayudará a comprender qué es (y qué no es) importante, o útil. Parte de la

filosofía de diseño de la familia de ordenadores personales de IBM se centra alrededor de las rutinas de servicio del BIOS (■ capítulos del 8 al 13) que proporcionan esencialmente todas las funciones y operaciones de control que IBM considera necesarias. La filosofía básica de la familia PC es: deje que el BIOS lo haga, no se enrede con el control directo. Desde mi punto de vista, ésta es una buena idea que tiene varios resultados beneficiosos. Cuando se utilizan las rutinas del BIOS, se fomentan las prácticas características de una buena programación y se evitan algunos de los trucos y chapucillas que han sido tan habituales en otros ordenadores. También incrementa la posibilidad de que los programas trabajen en todos los miembros de la familia PC. En resumen, da a IBM más flexibilidad a la hora de hacer mejoras y ampliaciones en la línea de ordenadores PC. Sin embargo, sería ingenuo por mi parte decirle simplemente: «No enrede con el control directo del *hardware*.» Por buenas o malas razones, en un momento determinado puede desear, o necesitar, que sus programas trabajen tan directamente con el *hardware* del ordenador como sea posible, teniendo que realizar lo que se llama pintorescamente *programación a nivel de placa*.

Cuando considere controlar directamente el *hardware* con sus propios programas, comprenderá que el mecanismo básico para hacerlo descansa en la utilización de puertos (■ que se discute en el capítulo 2). Con la única excepción de enviar información directamente a la pantalla (que se hace a través del uso de la memoria), todo el control directo del *hardware* del PC se hace enviando datos a través de los puertos del *hardware*. Con sólo unas pocas excepciones, el uso directo de los puertos para controlar el PC va en contra de la filosofía de diseño de IBM, y otra vez debemos insistir en que evite hacerlo. Las excepciones a esta regla incluyen aquellas prestaciones que el BIOS no controla, como es el caso específico de la generación de sonido (■ véase capítulo 7).

Las entradas y las salidas

¿Cómo se comunica el 8088? 24

Los formatos de datos del 8088 25

¿Cómo direcciona la memoria el 8088? 26

Expansión de la memoria mediante direcciones segmentadas 27

Los catorce registros del 8088 28

¿Cómo utiliza los puertos el 8088? 40

Diferencias en el uso de los puertos de los distintos elementos
de la familia PC 41

¿Cómo utiliza las interrupciones el 8088? 42

Hablando en general, cuanto más aprendemos sobre programación, más comprendemos las limitaciones que tienen los lenguajes de programación. Los lenguajes de alto nivel, como son el BASIC o el C, no están diseñados para incluir todas las posibles funciones que se pueden necesitar mientras programamos, a pesar de que algunos lenguajes son mejores que otros. En algún momento será necesario profundizar en nuestro sistema y utilizar algunas de las rutinas que los propios lenguajes usan, o incluso si nos adentramos más, llegar a programar a nivel de *hardware*.

Aunque algunos lenguajes proporcionan medios, que siempre son limitados, para hablar directamente con la memoria (como las sentencias PEEK y POKE del BASIC) o también con algunos de los *chips* (como las sentencias INP y OUT del BASIC), la mayoría de los programadores recurre eventualmente al lenguaje ensamblador, que es el lenguaje básico con el que se han construido el resto de los lenguajes y sistemas operativos. El lenguaje ensamblador del 8088, como el resto de los otros lenguajes, está formado por un conjunto de códigos de instrucciones simbólicas, que en el caso del 8088 se muestran en la figura 2-1. En el interior del 8088 estos códigos y los datos que están asociados con ellos están representados en forma binaria, con objeto de que puedan residir en la memoria y se muevan a través de la circuitería electrónica para realizar ciertas tareas específicas. A esta representación se le denomina lenguaje máquina.

Las operaciones que ejecutan las instrucciones del 8088 se pueden clasificar en unas cuantas categorías: las que hacen operaciones simples, como son las cuatro funciones aritméticas con números enteros de 8 y 16 bits; las que se emplean para mover datos de un lado a otro; las que son capaces de manipular bits individualmente, utilizando toscos métodos de desplazamiento; las que suelen comprobar valores y tomar decisiones lógicas en función de los resultados, y por último, pero no por ello con menor importancia, las que pueden interaccionar con la circuitería de su entorno. El tamaño de cada instrucción varía de uno a seis bytes. Por diseño, las instrucciones básicas, que son las que se utilizan más a menudo, tienen una longitud menor.

La programación en lenguaje ensamblador se puede llevar a cabo de dos formas diferentes: creando *rutinas de interfaz*, que unen a los programas de alto nivel con las rutinas de bajo nivel del DOS y de la ROM-BIOS, o creando programas completos en lenguaje ensamblador, que realicen tareas a nivel del *hardware* que no estén pensadas, o realizadas, por ninguna parte del sistema. De cualquier forma, para

comprender cómo se puede usar el lenguaje ensamblador, es necesario comprender cómo procesa la información el 8088 y cómo trabaja con el resto del ordenador. El objetivo de discusión del resto de este capítulo es estudiar cómo se comunica el 8088 con las otras partes del ordenador.

Mnemónico	Nombre completo	Mnemónico	Nombre completo
AAA	Ajuste ASCII para la suma	JE	Salta si igual
AAD	Ajuste ASCII para la división	JG	Salta si mayor
AAM	Ajuste ASCII para multiplicación	JGE	Salta si mayor o igual
AAS	Ajuste ASCII para la resta	JL	Salta si menor que
ADC	Suma con arrastre	JLE	Salta si menor que o igual
ADD	Suma	JMP	Salta
AND	Y lógico	JNA	Salta si no superior
CALL	Llamada	JNAE	Salta si no superior o igual
CBW	Convierte byte a palabra	JNB	Salta si no inferior
CLC	Borra el indicador de arrastre	JNBE	Salta si no inferior o igual
CLD	Borra el indicador de dirección	JNC	Salta si no arrastre
CLI	Borra el indicador de interrupción	JNE	Salta si no igual
CMC	Complementa el indicador de arrastre	JNG	Salta si no mayor
CMP	Compara	JNGE	Salta si no mayor o igual
CMPS	Compara byte o palabra (de cadena)	JNL	Salta si no menor
CMPSB	Compara cadena de bytes	JNLE	Salta si no menor o igual
CMPSW	Compara cadena de palabras	JNO	Salta si no desbordamiento
CWD	Convierte palabra a doble palabra	JNP	Salta si no paridad
DAA	Ajuste decimal para la suma	JNS	Salta si no signo
DAS	Ajuste decimal para la resta	JNZ	Salta si no cero
DEC	Decrementa	JO	Salta si desbordamiento
DIV	Divide	JP	Salta si paridad
ESC	Escape	JPE	Salta si paridad par
HLT	Alto	JPO	Salta si paridad impar
IDIV	Divide enteros	JS	Salta si signo
IMUL	Multiplica enteros	JZ	Salta si cero
IN	Entra byte o palabra	LAHF	Carga AH con indicadores
INC	Incrementa	LDS	Carga puntero en DS
INT	Interrumpe	LEA	Carga dirección efectiva
INTO	Interrumpe por desbordamiento	LES	Carga puntero en ES
IRET	Vuelta de la interrupción	LOCK	Bloquea <i>bus</i>
JA	Salta si superior	LODS	Almacena byte o palabra (de cadena)
JAE	Salta si superior o igual	LODSB	Almacena byte (cadena)
JB	Salta si inferior	LODSW	Almacena palabra (cadena)
JBE	Salta si inferior o igual	LOOP	<i>Bucle</i>
JC	Salta si arrastre	LOOPE	<i>Bucle</i> mientras igual
JCX	Salta si CX cero	LOOPNE	<i>Bucle</i> mientras no igual

Figura 2-1. Conjunto de instrucciones del 8088

Mnemónico	Nombre completo	Mnemónico	Nombre completo
LOOPNZ	Bucle mientras no cero	ROR	Rotación derecha
LOOPZ	Bucle mientras cero	SAHF	Almacena AH en los indicadores
MOV	Mueve	SAL	Desplazamiento aritmético a la izquierda
MOVS	Mueve Byte o palabra (de cadena)	SAR	Desplazamiento aritmético a la derecha
MOVSB	Mueve byte (cadena)	SBB	Resta con acarreo
MOYSW	Mueve palabra (cadena)	SCAS	Busca byte o palabra (de cadena)
MUL	Multiplicar	SCASB	Busca byte (cadena)
NEG	Niega	SCASW	Busca palabra (cadena)
NOP	No operación	WAIT	Espera
NOT	Negación lógica	SLAT	Traslada
OR	O lógico	SHL	Desplazamiento a la izquierda
OUT	Salida de byte o palabra	SHR	Desplazamiento a la derecha
POP	Desapila	STC	Pon indicador de arrastre
POPF	Desapila indicadores	STD	Pon indicador de dirección
PUSH	Apila	STI	Pon indicador de interrupción
PUSHF	Apila indicadores	STOS	Almacena byte o palabra (de cadena)
RCL	Rotación a la izquierda contando con el arrastre	STOSB	Almacena byte (cadena)
RCR	Rotación a la derecha contando con el arrastre	STOSW	Almacena palabra (cadena)
REP	Repite	SUB	Resta
RET	Retorno	TEST	Test
RDL	Rotación izquierda	XCHG	Intercambio
		XOR	O exclusivo lógico

Figura 2-1. Conjunto de instrucciones del 8088

¿COMO SE COMUNICA EL 8088?

El 8088 interacciona con la circuitería que le rodea de tres formas: accede a la memoria de forma directa o indirecta, a través de los puertos y mediante señales denominadas interrupciones.

La **memoria** se utiliza leyendo, o escribiendo, valores que están almacenados en localizaciones de memoria que se identifican mediante direcciones numéricas. A estas localizaciones de memoria se puede acceder de dos formas: directamente a través del *chip* 8237A, comúnmente conocido como controlador de acceso directo a memoria (DMA), o indirectamente a través de los registros internos del 8088. El controlador del disco y las puertas de comunicaciones en serie pueden acceder directamente a la memoria a través del controlador DMA. El resto de los dispositivos transfiere los datos (a y de) la memoria a través de los registros del 8088. Para encontrar más información sobre el controlador DMA, véase la página 28. Sobre los registros hablaremos en este mismo capítulo más detenidamente en la página 25.

Los **puertos** son el modo general de comunicación que emplea el 8088 con el resto de los componentes de la circuitería distintos de la memoria. Como en el caso de las localizaciones de memoria, los puertos se identifican mediante un número, y los datos se pueden leer, o escribir, en ellos. La asignación de un puerto es único y característico del diseño de un modelo de ordenador particular. Generalmente, todos los miembros de la familia IBM PC utilizan las mismas especificaciones de puerto, con algunas variaciones entre los diferentes modelos (☛ véase la página 43).

Las **interrupciones** son la forma empleada por la circuitería exterior al 8088 para informar que algo (como, por ejemplo, la pulsación de una tecla) ha ocurrido y que, por tanto, requiere que se realice alguna acción. Aunque las interrupciones son esenciales para la interacción del 8088 con su entorno, el concepto de interrupción se utiliza también con otros propósitos. Por ejemplo, el sistema BIOS, o el sistema operativo, pueden producir interrupciones mediante *software* para requerir y ejecutar programas especiales de servicio. Las interrupciones son un tema bastante importante cuando se desea programar a la familia PC; por ello, les dedicamos una sección especial al final de este capítulo.

Los formatos de datos del 8088

Datos numéricos. El 8088 puede trabajar con sólo cuatro formatos de datos numéricos, todos ellos de tipo entero. Los formatos se han implementado utilizando dos bloques diferentes de construcción: el byte de 8 bits y la palabra de 16-bits (2-bytes). Ambas unidades básicas se derivan de la capacidad de procesamiento de 16-bits del 8088 y de su *bus* de datos de 8-bits. El byte es la unidad fundamental, y cuando el 8088 direcciona la memoria, los bytes son la unidad básica de direccionamiento. En un único byte, el 8088 puede representar los números positivos sin signo que están contenidos en el rango de valores del 0 al 255 (esto es, 2^8 posibilidades). Si el número es un valor con signo, un byte puede representar los valores que están contenidos en el rango de -128 a $+127$ (☛ véase la figura 2-2).

Cuando se necesitan valores enteros mayores que un byte, el 8088 utiliza dos bytes adyacentes y los trata como una unidad. La palabra de 2 bytes es el formato más común. En una palabra de 2 bytes se pueden representar los números enteros sin signo que están contenidos en un rango del 0 al 65535. Cuando el número tiene signo, el valor puede estar comprendido entre el -32768 y el $+32767$.

Datos tipo carácter. El dato tipo carácter se almacena en formato ASCII estándar, con cada carácter ocupando un byte. El 8088 no sabe nada sobre caracteres ASCII y los trata como bytes arbitrarios, con una excepción parcial: hay un conjunto de instrucciones del 8088 que permite realizar la suma y la resta decimal con caracteres numéricos ASCII. La aritmética se hace en binario, pero la combinación del indicador AF (véase página 37) y unas pocas instrucciones especiales le permiten, en la práctica, trabajar con caracteres decimales y obtener resultados decimales.

Tamaño	Signo	Rango	
		Dec	Hex
8	No	0 a 255	00 a FF
8	Sí	-128 a 0 a +127	80 a 00 7F
16	No	0 a 65 535	0000 a FFFF
16	Sí	-32 768 a 0 a +32 767	8000 a 0000 a 7FFF

Figura 2-2. Los cuatro formatos de datos utilizados por el 8088

➤ Véase el apéndice C para encontrar más información sobre el código ASCII y el conjunto extendido de caracteres ASCII que emplea la familia PC.

¿COMO DIRECCIONA LA MEMORIA EL 8088?

El 8088 es un microprocesador de 16 bits, y por consiguiente no puede trabajar directamente con valores mayores de 16 bits; por tanto, el valor decimal más grande que puede representar es el 65 535, o 64K. Teóricamente, esto significa que el 8088 sólo podría acceder a 64K direcciones de memoria. Pero, como ya dijimos en el capítulo previo, puede acceder a muchas más direcciones que ésas, para ser exactos a 1.024K. Esto es posible porque el esquema de direccionamiento de 20 bits utilizado por el 8088 amplía el rango de posiciones de memoria con el que el 8088 puede trabajar, pasando de 2^{16} (65 535) a 2^{20} (1 048 576). Pero como el 8088 está limitado por su capacidad de procesamiento de 16 bits, para acceder a direcciones de 20 bits necesita utilizar un procedimiento de direccionamiento que se adapte al formato propio de 16 bits.

Expansión de la memoria mediante direcciones segmentadas

El 8088 divide el espacio de memoria direccionable en un número arbitrario de *segmentos*, cada uno de los cuales contiene como máximo 64Kbytes. Cada segmento empieza en una localización cuya dirección es divisible por 16 bytes, y que se conoce como *dirección del segmento* o *párrafo del segmento*. Para acceder a los bytes individuales, o a las palabras, se emplea una dirección adicional llamada *desplazamiento* u *offset*, que apunta a la posición de un byte exacto dentro del segmento de 64K designado por la dirección del segmento. Las direcciones internas siempre son relativas al comienzo del segmento, por lo que también se denominan *direcciones relativas*.

Las direcciones se crean y se manipulan combinando la dirección del segmento de 16 bits y una dirección relativa de 16 bits. La dirección de segmento se trata como si estuviera desplazada a la izquierda cuatro bits. Cuando se añade la dirección relativa, se obtiene una dirección completa de 20 bits, tal y como aparece en la figura 2-3. A las dos palabras de 16 bits se les denomina usualmente *dirección segmentada*, o también *vector*, cuando se hace referencia a interrupciones (☛ véase página 44 para más información).

La dirección de segmento se escribe mediante 5 dígitos en hexadecimal y siempre tienen un cero situado en el último lugar, tal como FFE40 o B8120. El cero proviene de multiplicar los 16 bits originales, número de cuatro dígitos hexadecimales, por 16. (Se obtiene el mismo desplazamiento efectivo cuando se multiplica un valor decimal por su número de base 10, como, por ejemplo, en $23 \times 10 = 230$.) El hecho de que la parte de segmento de una

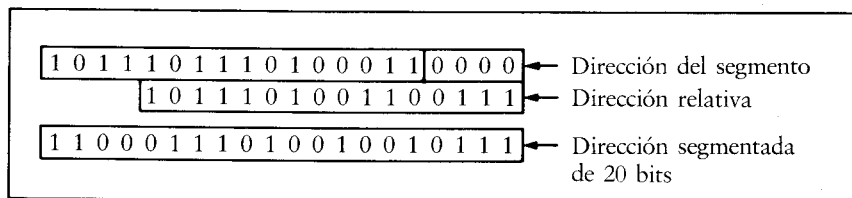


Figura 2-3. La dirección del segmento apunta al principio de un segmento de memoria de 64K, y la dirección relativa, que apunta a un byte específico dentro del segmento, se combina en la CPU para formar una dirección física de 20 bits

dirección segmentada se desplace a la izquierda cuatro bits (que es lo mismo que si estuviera multiplicada por 16), es la razón por la que la parte segmento sólo puede apuntar a las direcciones de memoria reales que son un múltiplo de 16, y por ello es necesaria la dirección relativa para definir la posición precisa dentro del segmento. Las direcciones relativas se escriben con valores hexadecimales de 4 dígitos. Cuando se suman estos dos números, forman otro número hexadecimal de 5 dígitos, que es una dirección de 20 bits. Por ejemplo, si cogemos una dirección de segmento hexadecimal tal como 12340 y lo multiplicamos por 16, obtenemos 12340. Entonces si le añadimos la dirección relativa del byte que estamos buscando, tal como 4321, obtenemos un resultado hexadecimal de 5 dígitos, tal y como se puede comprobar con el siguiente ejemplo:

12340	La dirección de segmento en notación hexadecimal, desplazada 4 bits
+ 4321	La dirección relativa en notación hexadecimal
16661	La dirección segmentada de 20 bits en notación hexadecimal

Cuando se escribe una dirección de 20 bits partida en segmento y dirección relativa, se utiliza la notación 0000:0000, con el segmento en el lado izquierdo, los dos puntos y la dirección relativa a la derecha. Por ejemplo, una dirección de 20 bits escrita como FFE6E podría aparecer escrita como FFE4:002E en notación segmentada. Tal y como aparece en la figura 2-4, se puede expresar una dirección simple de 20 bits mediante una gran variedad de formas, utilizando notación segmentada, en función de cual es el segmento que se elija.

Los catorce registros del 8088

El 8088 fue diseñado para ejecutar instrucciones y llevar a cabo operaciones aritméticas y lógicas, así como recibir instrucciones y pasar datos a y desde la memoria. Para hacer esto, utiliza registros de 16 bits.

Hay catorce registros en total, cada uno de ellos con un objetivo especial. Cuatro *registros de trabajo* que se utilizan temporalmente por los programas para almacenar resultados intermedios y los operandos que necesitan las operaciones aritméticas y lógicas. Cuatro *registros de segmento* que almacenan las direcciones de comienzo de ciertos segmentos en la memoria. Cinco *registros relativos* que almacenan las direcciones relativas que se emplean con las direcciones de segmento para indicar dónde están situados los datos en la

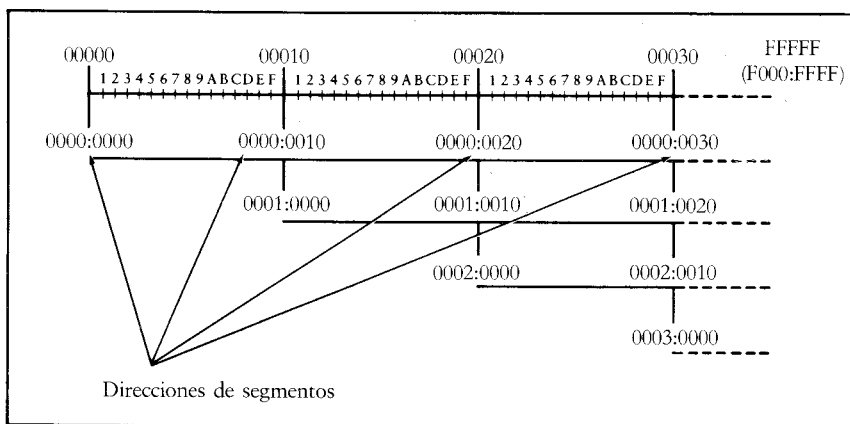


Figura 2-4. La dirección interna se determina siempre de forma relativa a la dirección del parágrafo de segmento. Por esta razón puede haber varias direcciones segmentadas que indiquen la misma localización de memoria

memoria. Finalmente, hay un *registro de estado*, formado por nueve bits que se emplean para registrar la información del estado del 8088 y de las operaciones de control del 8088 (■ Véase figura 2-5.)

Los registros de trabajo

Cuando un ordenador está procesando datos, gran parte del tiempo el microprocesador lo gasta en traer y llevar datos a la memoria. Este tiempo de acceso se puede reducir sustancialmente en muchas ocasiones, guardando en el interior del 8088 los operandos más usados y los resultados obtenidos. Hay cuatro registros de 16 bits, normalmente llamados registros de trabajo, o de datos, que emplean con este propósito.

Los registros de trabajo se conocen como AX, BX, CX y DX. Cada uno de ellos se puede a su vez subdividir y direccionarse de forma separada, como si fueran dos medio-registros de 8 bits. Los medio-registros de mayor orden se conocen como AH, BH, CH y DH, y los medio-registros de orden bajo se conocen como AL, BL, CL y DL. La utilización de los registros completos, o medios, se puede entremezclar libremente, a gusto del programador en función de las necesidades.

Los registros de trabajo se emplean la mayor parte de las veces como áreas temporales de trabajo, en particular para realizar

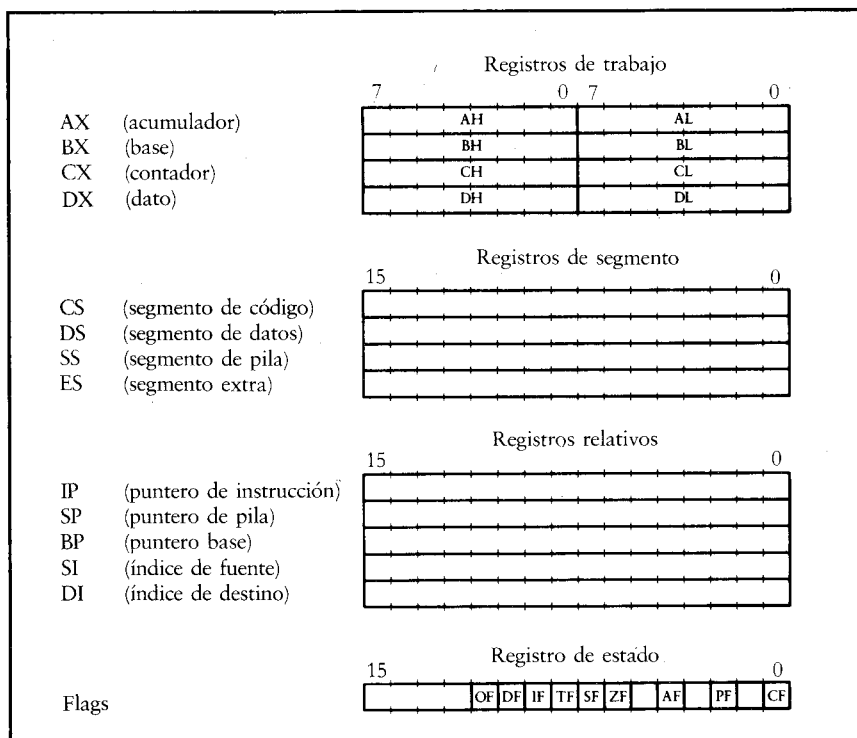


Figura 2-5. Registros e indicadores del 8088

operaciones aritméticas. La suma y la resta se pueden hacer en memoria sin utilizar los registros, pero éstos emplean menos tiempo y, por tanto, las operaciones se realizan de forma más rápida.

Aunque estos registros se encuentran disponibles para realizar cualquier tipo de trabajo, cada uno tiene asignado algunos usos especiales. Por ejemplo:

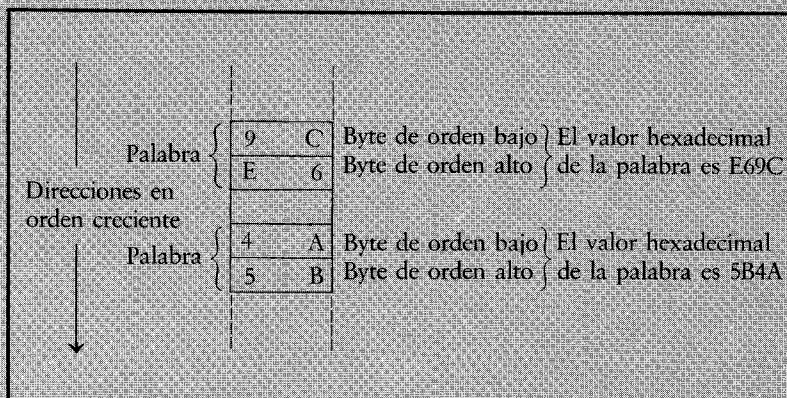
- El registro AX es un acumulador, y es el registro principal para realizar las operaciones aritméticas.
- El registro BX (base) se utiliza a menudo para apuntar el comienzo de una tabla en memoria. Se puede utilizar también para almacenar la parte relativa de una dirección segmentada.
- El registro CX (contador) se utiliza como un contador de repetición para control de *bucles*. Por ejemplo, la instrucción LOOP utiliza CX para almacenar el contador que indica el número de iteraciones del *bucle*. Ninguno de los otros registros puede realizar esta función.

- El registro DX se utiliza para almacenar datos de 16 bits con propósitos generales.

Aunque los registros de trabajo se utilicen para el almacenamiento temporal de datos y operandos, o para las tareas específicas ya mencionadas, la naturaleza de «zona de trabajo» les posibilita también para otros usos. Por ejemplo, los cuatro se usan a menudo para albergar las direcciones relativas de datos que se pasan como parámetros de un programa a otro.

Almacenamiento invertido o *back-words*

Mientras la memoria del PC se direcciona haciendo referencia a bytes individuales de 8 bits, muchas operaciones involucran palabras de 16 bits. En la memoria, una palabra de 16 bits se almacena en dos bytes adyacentes de 8 bits. El byte menos significativo de la palabra se almacena en la localización de memoria más baja, y el byte más significativo se almacena en la localización de memoria más alta. Desde este punto de vista, el almacenamiento de una palabra de esta forma es opuesto a lo que cabría esperar. Debido al aparente cambio que implica este esquema de almacenamiento, a veces se denomina caprichosamente almacenamiento invertido o *back-words*.



Si se trabaja con bytes y palabras en memoria, se debe tener cuidado en no confundirse con el almacenamiento tipo *back-words*. El origen de la confusión se debe principalmente a como se escriben los datos. Si escribimos el valor de una palabra en hexadecimal, por ejemplo ABCD, el orden de significación de los caracteres es análogo al que utilizamos con los números decimales. El dígito más significativo se escribe primero. Pero cuando escribimos una palabra de la forma en que se debe de almacenar en la memoria, se escribe primero la localización de la dirección más baja. Por tanto, en la memoria tendríamos que escribir el número ABCD como CDAB; es decir, con los bytes intercambiados.

Los registros de segmento

Como acabamos de ver hace poco, la dirección completa de una posición de memoria consiste en la dirección de un segmento de 64K y la dirección interna en el segmento. Los cuatro registros, llamados CS, DS, SS y ES, se utilizan para identificar cuatro segmentos de 64K específicos de memoria. Otros cinco registros relativos o índice, que estudiaremos brevemente, se utilizan para almacenar la dirección interna del dato en el segmento de 64K que se determine.

De los cuatro registros de segmento, los tres siguientes se dedican a propósitos especiales:

- El registro CS localiza el segmento del código; es decir, la dirección a partir de la cual empieza el programa que está siendo ejecutado.
- El registro DS localiza el segmento de datos, que es la dirección donde empieza el área de memoria que almacena los datos que se están utilizando.
- El registro SS localiza el segmento de pila, que es la dirección donde empieza un lugar de trabajo temporal que almacena parámetros y direcciones que utiliza el programa que está activo (● véase página 35 para encontrar más información sobre la pila).

El cuarto registro de segmento, denominado registro ES, apunta hacia un segmento extra, que se utiliza normalmente como suplemento del segmento de datos, con objeto de que se pueda utilizar más de 64K de memoria para almacenamiento. También se utiliza para transferir datos entre segmentos.

Es común que los cuatro segmentos se solapen o sean idénticos; también es común que sólo se utilice una parte del segmento de 64K; por ejemplo, un programa puede necesitar tan sólo 16K de un segmento de 64K. La figura 2-6 ilustra cómo se puede distribuir la zona de la memoria.

Todas las instrucciones del 8088 que utilizan la memoria también emplean implícitamente el registro de segmento apropiado para que la operación se realice. Por ejemplo, la instrucción MOV, cuando actúa con datos, utiliza el registro DS. La instrucción JMP, que afecta al flujo de un programa, utiliza automáticamente el registro CS. En algunos casos, si fuera necesario, podemos sustituir el registro de segmento implícito por otro. En lenguaje ensamblador, éste se puede hacer con prefijos de cambio de segmento.

El comprender cómo se usan los registros de segmento le dará

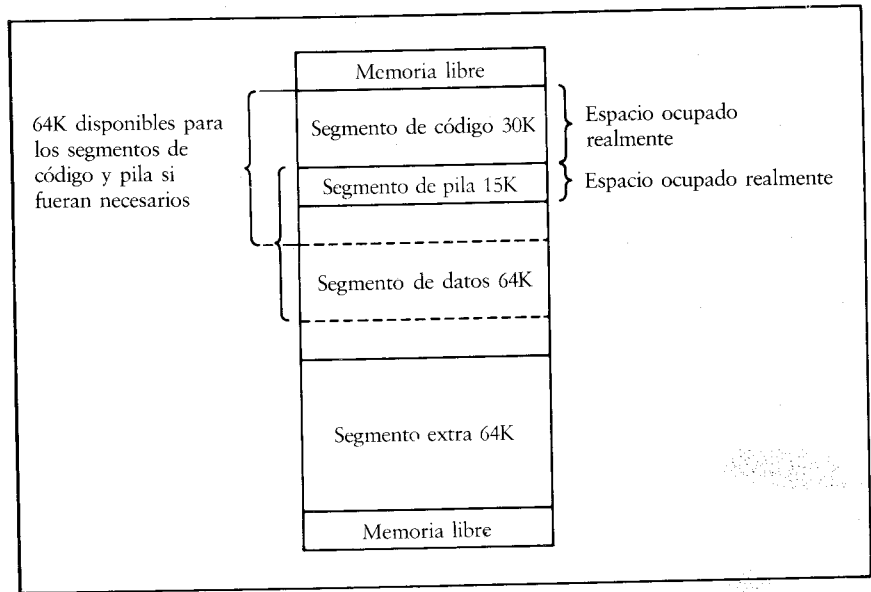


Figura 2-6. Los segmentos pueden abarcar áreas de 64K separadas, o pueden solaparse. Se indica el lugar donde están los segmentos de la pila, de datos y de un segmento extra, almacenando en el registro de segmento apropiado (SS, DS o ES) la dirección de comienzo del respectivo segmento

capacidad de discernimiento sobre los límites prácticos de la utilización de la memoria cuando se emplea un sistema de 16 bits. Si sus programas, o datos, requieren bloques de memoria mayores de 64K, necesitará aplicar este conocimiento para manipular los registros de segmentos. A continuación encontrará algunas indicaciones.

Si utiliza tan sólo el registro CS, el máximo tamaño de los programas es el límite de 64K que tienen las direcciones internas. El 8088 se diseñó para tener el control de un programa, así que no es necesario manipular directamente el registro CS y cambiar la dirección del segmento de código cuando se necesita más memoria. Sin embargo, utilizando ciertas instrucciones para el control del programa que tiene el 8088, tales como llamadas y saltos, es posible actualizar indirectamente el registro CS. Así es como algunos lenguajes de programación permiten cualquier tamaño a los programas. (El intérprete BASIC y la versión 1 de Microsoft-C no; Pascal y la Versión 2 de C sí que la permiten.)

Por otra parte es relativamente fácil manipular el registro DS, o usar el registro ES, para permitir utilizar más de 64K de datos. Pero, aunque teóricamente esta capacidad podría permitir emplear una cantidad de datos ilimitada, en la práctica la mayoría de los lenguajes de programación sólo puede trabajar con 64K de datos en memoria, debido a la forma en que han sido diseñados. En la mayor parte de las ocasiones esto no representa ningún problema, ya que los programas pueden llevar a cabo su trabajo de manera muy confortablemente con el límite de 64K. Aunque programas más sofisticados podrían hacer un buen uso de grandes cantidades de memoria, pocos programas necesitan acercarse a los 64K.

❑ *NOTA: Para el intérprete BASIC hay algunas pocas cosas más relacionadas con los registros de segmento. El registro CS apunta al intérprete BASIC. Tanto un programa BASIC como sus datos son considerados datos desde el punto de vista del 8088 y, por ello, ambos utilizan el registro DS. Por esta razón hay un límite máximo de 64K bytes para el tamaño combinado de los programas BASIC y sus datos.*

La orden DEF SEG del BASIC hace la operación equivalente a seleccionar el registro DS para ciertas operaciones de BASIC, tales como PEEK y POKE, aunque siempre mantiene la dirección de segmento DS original del BASIC. (PEEK y POKE se utilizan para especificar la dirección relativa en el segmento de datos.) ■ Véase página 65, cómo acceder al valor verdadero del registro DS del BASIC.

Los registros relativos

Se utilizan los cinco registros relativos para localizar un byte en concreto, dentro de un segmento específico de 64K. El registro, llamado puntero de instrucción (IP), indica cuál es la dirección donde se encuentra situada dentro del segmento de código instrucción que se va a ejecutar; dos más, que se denominan registros de pila, están íntimamente ligados a la pila, que no es más que una zona de la memoria donde el 8088 guarda información de direcciones y de datos, que necesita recordar para su utilización posterior (■ encontrará más información sobre pilas en la página 35), y los dos registros que quedan se denominan registros de índice y se utilizan para señalar la situación de los operandos que se van a utilizar dentro del segmento de datos.

El **puntero de instrucción** (IP), también llamado contador del programa (PC), proporciona la dirección relativa en el segmento de

código donde se encuentra localizada la instrucción del programa que se va a ejecutar. Se utiliza junto con el registro CS.

Los programas no tienen acceso directo al registro IP, pero hay un conjunto de instrucciones, tales como JMP y CALL, que cambian el contenido a o desde la pila.

Los registros de pila, denominados puntero de pila (SP) y puntero de base (BP), proporcionan direcciones relativas dentro del segmento de pila. SP proporciona la situación de la cabecera de la pila y es análogo al registro IP. El registro BP se utiliza para indicar, partiendo de la posición inicial de la pila, dónde se encuentra determinada información. Este registro es particularmente importante para crear rutinas de interfaz en lenguaje ensamblador. Por ello, aparecerá muy a menudo en los ejemplos construidos en lenguaje ensamblador que aparecen a lo largo de los capítulos comprendidos entre el 8 y el 20.

Los registros de índice, denominados el índice de fuente (SI) y el índice de destino (DI), se utilizan normalmente con otro registro (AX, BX, CX o DX) o con ciertas instrucciones, que proporcionan la dirección relativa del lugar inicial de un campo de datos que esté comprendido en el interior de un segmento de datos. Se utilizan muy a menudo cuando se transfieren grandes listas de datos. Las instrucciones de cadena que utilizan SI y DI transfieren los bytes o las palabras de uno en uno. Entonces los registros SI y DI incrementan sus valores automáticamente, de tal forma que cuando acaba cada transferencia no se tiene que añadir 1 cada vez para mover el siguiente byte de información.

El registro de estado

El decimocuarto y último registro del 8088, llamado registro de estado, es realmente una colección de bits individuales de control, denominados indicadores o *flags*, que están organizados en forma de registro, de manera que o bien pueden ser guardados y recuperados como si constituyesen un dato, o bien los indicadores se modifican y se inspeccionan como elementos independientes; esta segunda técnica suele ser la más normal.

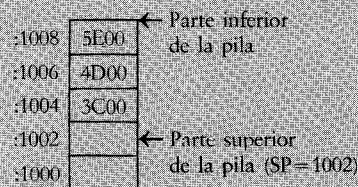
La pila

La pila es una característica del 8088 que proporciona a los programas un lugar para almacenar información que permita seguir la pista del trabajo que se está realizando. La utilización más importante de

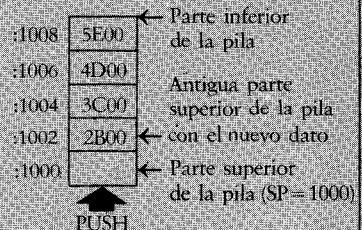
la pila es mantener la información de la localización de memoria, desde donde fueron llamadas las subrutinas, y de algunos otros parámetros que le fueron pasados. La pila también se puede utilizar temporalmente como zona de trabajo, aunque esto es menos importante y menos común.

El nombre de pila proviene de una analogía entre su funcionamiento y el de una pila de platos de una cafetería. Cuando el dato nuevo es apilado en la parte superior, el dato viejo desciende en el orden de colocación. Una pila siempre se opera del siguiente modo: el último en entrar es el primero en salir (LIFO). Esto quiere decir que cuando la pila se utiliza para seguir la pista del lugar al que tiene que retornar un programa, se vuelve primero al programa que realizó la llamada más reciente. De esta forma, una pila mantiene el orden de funcionamiento de los programas, subrutinas y manejo de interrupciones, sin tener en cuenta la complejidad del proceso.

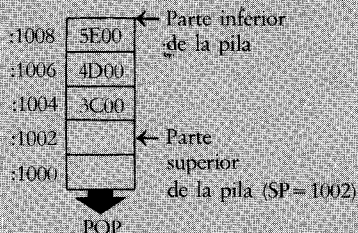
Una pila se utiliza de la parte inferior (dirección más alta) a la superior (dirección más baja), de tal forma que, cuando se apila un dato, se almacena en las direcciones de memoria que están justo debajo del puntero de la pila. La pila crece de manera descendente, de tal forma que, cuando se anade un dato, la localización de la parte superior de la pila se mueve para descender a direcciones más bajas, decreciendo cada vez el valor del SP. Esto tiene la ventaja de hacer avances positivos cuando se recuperan datos de la pila. Es necesario tener esto en cuenta cuando se accede a la pila, ya que es una operación que realizan las rutinas en lenguaje ensamblador.



a. Pila antes de un PUSH



b. Pila después de un PUSH



c. Pila después de un POP

Cualquier programa en cualquier momento puede crear un espacio de pila nuevo, pero esto no es una operación que se haga usualmente. Lo normal, cuando se está ejecutando algún programa, es que se cree sólo una pila, que se utiliza conjuntamente por el programa, por sus subrutinas y por algún programa de servicio del DOS y del BIOS que se invoque durante la ejecución del programa. Cuando no se está ejecutando ningún programa, el DOS utiliza su propia pila.

No hay ninguna forma sencilla para estimar el tamaño de la pila que puede necesitar un programa, y el diseño interno del 8088 no proporciona ninguna forma automática para detectar cuándo el espacio ocupado por la pila es pequeño o exagerado. Esto puede llegar a poner nervioso a más de un programador cuando piense acerca del espacio de memoria que sería necesario reservar para la pila. Es común en los lenguajes de programación, incluyendo el intérprete BASIC, utilizar automáticamente un tamaño de pila de 512 bytes, a menos que se especifique otra cantidad. Puede suponer con bastante seguridad que este tamaño es suficiente, a menos que tenga razones especiales para esperar otra cosa.

Hay nueve indicadores de 1 bit en el registro de estado de 16 bits y los otros siete bits se dejan sin utilizar. Estos indicadores se pueden dividir de forma lógica en dos grupos: seis indicadores de estado, que se utilizan para registrar la información del estado del procesador (que señalan normalmente lo que ocurre con una comparación y operación aritmética), y tres indicadores de control, que dirigen a algunas de las instrucciones del 8088. Prepárese para ver una gran variedad de notaciones para estos índices, incluyendo nombres distintos para indicar cuándo están con un uno (1) o con un cero (0). Los términos que aparecen en las figuras 2-7 y 2-8 son los más comunes.

Código	Nombre	Uso
CF	Indicador de arrastre	Indica un bit de arrastre aritmético
OF	Indicador de desbordamiento	Indica desbordamiento aritmético
ZF	Indicador de cero	Indica resultado cero, o comparación igual
SF	Indicador de signo	Indica resultado/comparación negativo
PF	Indicador de paridad	Indica un número par de bits a 1
AF	Indicador de arrastre auxiliar	Indica ajuste necesario en operaciones aritméticas decimales codificadas en binario (BCD)

Figura 2-7. Los indicadores (flags) de estado del registro de estado del 8088

Código	Nombre	Uso
DF	Indicador de dirección	Controla la dirección izquierda/derecha en operaciones repetidas (p. ej., usando SI y DI)
IF	Indicador de interrupción	Controla si están habilitadas las interrupciones
TF	Indicador de detención del <i>software</i>	Controla operaciones de paso a paso (usada por DEBUG) para generar una detención del <i>software</i> al final de cada instrucción.

Figura 2-8. Los tres indicadores (flags) de control del registro de estado del 8088

Direccionamiento de la memoria a través de los registros

Ya hemos visto que la memoria se direcciona siempre mediante la combinación de un valor de la dirección del segmento y de un valor relativo, y que la parte del segmento de una dirección siempre viene de uno de los cuatro registros de segmento. La parte relativa puede venir de una combinación de uno, dos o tres de las siguientes fuentes:

- Un valor relativo que aparece en la propia instrucción.
- Un registro, usualmente AX, BX, CX o DX.
- Un registro de índice, SI o DI.

Debe darse cuenta de este rango de posibilidades cuando escriba o lea códigos en lenguaje ensamblador, ya que no todas las instrucciones permiten utilizar todas las formas posibles indicadas a la hora de componer una dirección. Una forma para comprobar si está permitida una combinación es tratar de ver si es aceptada por el ensamblador o por el comando A del DEBUG.

También debe tener en cuenta las convenciones de notación utilizadas para las direcciones de memoria y para los registros. Los corchetes [] se utilizan para indicar que lo que encierran se utiliza como una dirección relativa. Este es un elemento clave en el direccionamiento de la memoria: sin corchetes, el valor que se almacene en el registro se utilizará en cualquier operación que se especifique. Veamos los siguientes ejemplos:

ADD AX,BX	Suma el contenido de BX al de AX; no hay ningún direccionamiento de memoria
ADD AX,[BX]	Direccionamiento indirecto: añade a AX un valor que está en la memoria; BX da la dirección relativa del valor
ADD [BX],AX	Añade el valor contenido en AX al contenido en una localización de memoria; BX da la dirección relativa del valor al que se sumará el valor contenido en AX
ADD AX,123	Direccionamiento inmediato: suma 123 al valor contenido en AX
ADD AX,[123]	Suma el valor localizado en la dirección relativa 123 al valor contenido en AX
ADD AX,[BX + SI + 123]	Direccionamiento indirecto indexado: suma el valor localizado en la dirección relativa generada sumando los dos registros y un número al valor contenido en AX

Reglas para el uso de los registros

Es importante conocer que se pueden aplicar un conjunto de reglas cuando se utilizan los registros, y es esencial tenerlas en cuenta cuando se escriban rutinas en lenguaje ensamblador. Las reglas y convenciones de uso varían con las circunstancias y con el lenguaje de programación; desafortunadamente no hay reglas de oro que se apliquen siempre, pero aquí daremos algunas, que se pueden aplicar en la mayoría de las ocasiones. (● Encontrarás más orientaciones, y formas de trabajo para imitar, en los ejemplos de los capítulos comprendidos entre el 8 y el 20.) Tenga en cuenta, por tanto, que las reglas siguientes sólo son generales y no absolutas.

En una rutina de interfaz construida en lenguaje ensamblador, hay tres formas generales de utilizar los registros: algunos registros pueden alterarse libremente; otros pueden cambiarse, pero deben volver a su situación inicial al final de la rutina, y el resto de los registros no puede cambiarse nunca.

Generalmente, los registros de trabajo (AX a DX) pueden variar libremente, sin causar ningún daño al programa de llamada. Téngase en cuenta que el registro AX se utiliza comúnmente para devolver resultados, y que, bajo algunas circunstancias, los parámetros se pasan vía estos registros y son descartados después de su uso.

Algunas reglas particulares se deben aplicar a los cuatro registros de segmento (CS, DS, SS y ES). El registro CS no debe ser cambiado nunca directamente, aunque se puede cambiar indirectamente a través de llamadas a subrutinas. El registro DS puede variar, pero normalmente es necesario restablecer el valor inicial después. El valor

original del registro SS es necesario preservarlo siempre que se hagan cambios en el registro. Normalmente, las subrutinas continúan usando la pila a la que apuntaba SS. Observe que el cambio del valor de SS puede interferir con el uso del puntero base (BP) para los parámetros de acceso. El registro ES se puede variar normalmente a voluntad.

El puntero de instrucciones (IP/PC) no debe de variar directamente, como en el caso del registro CS, ya que los cambios se producen de forma automática. Y correcta.

El puntero de pila (SP) puede cambiar, pero, normalmente, todos los cambios de este registro se producen como resultado indirecto del uso de la pila. Restaurar la pila (lo que implica dar una valor adecuado al SP) es uno de los convenios importantes que existen a la hora de utilizar las subrutinas; las reglas para esto suelen variar (véanse los ejemplos que aparecen en los capítulos comprendidos entre el 8 y el 20).

El puntero base (BP) se cambia normalmente para conseguir el acceso directo a algunos parámetros, y, a menudo, al finalizar se restablece su contenido.

Los registros de índice (SI y DI) pueden ser alterados libremente cuando sea necesario.

En el registro de estado, los indicadores de estado pueden variar rutinariamente. Recuerde que algunos de los indicadores de estado se utilizan ocasionalmente para señalar que se ha producido algún resultado; por tanto, su inicialización puede ser importante. Los indicadores CF y ZF se utilizan más a menudo con este propósito. Como en el caso de los indicadores de control, el indicador de interrupción (IF) debe dejarse a uno (interrupciones invalidadas), y probablemente también es prudente dejar el indicador de dirección (DF) a uno; poner a uno el indicador de detención mediante *software* (TF) es suicida.

¿COMO UTILIZA LOS PUERTOS EL 8088?

El 8088 se comunica y controla algunas partes del ordenador a través de los puertos de entrada y salida (E/S). Los puertos de (E/S) son zonas físicas a través de las cuales pasa información a o desde un dispositivo de E/S, como, por ejemplo, un teclado o una impresora. La mayoría de los *chips* de soporte que describimos en el capítulo 1 utiliza los puertos de E/S; en efecto, cada *chip* puede usar varias direcciones de puertos con diferentes propósitos.

Cada puerto se identifica mediante un número de 16 bits, que puede estar comprendido entre 0 y 65535. La CPU envía datos, o

información de control a una puerta determinada, especificando su número, y la puerta entonces responde pasando datos, o información de su estado, a la CPU.

De la misma forma que cuando accede a la memoria, la CPU utiliza los *buses* de datos y direcciones como los conductos de comunicación con los puertos. Para acceder a un puerto, la CPU envía una señal por el *bus* de control que notifica a todos los dispositivos de E/S que la dirección colocada en el *bus* de direcciones es de un puerto; posteriormente, el dispositivo correspondiente al puerto activado responde.

El número de puerto direcciona una posición de memoria, que es parte de los dispositivos de E/S, pero no de la memoria principal. Se usan instrucciones especiales de E/S para señalar el acceso a un puerto y enviar información en ambos sentidos a los dispositivos de E/S. Algunos dispositivos de E/S, tales como los controladores de video, también usan las direcciones de la memoria principal, además de sus puertos de E/S, y hace que la CPU las tome como una parte de la memoria principal. Esto se conoce como E/S mapeada en memoria. Generalmente, los dispositivos mapeados en memoria son más fáciles de programar, porque nos permiten utilizar las instrucciones más flexibles de memoria, en vez de las instrucciones inflexibles y limitadas de E/S del conjunto que ofrece el 8088.

□ NOTA: El conjunto de instrucciones del 8088 incluye las instrucciones IN y OUT para leer y escribir datos en un puerto. El BASIC incluye las instrucciones INP y OUT para leer y escribir datos en puertos de la misma forma, permitiéndonos experimentar con los diferentes puertos, utilizando simples rutinas de BASIC e incorporándolas entonces en nuestros programas sin tener que recurrir a la programación en lenguaje ensamblador.

Diferencias en el uso de los puertos de los distintos elementos de la familia PC

El uso de los puertos específicos está determinado por los diseñadores del *hardware*. Los programadores que hacen uso de estos puertos necesitan conocer los números de puerto, así como su uso y significado. Como las asignaciones de los puertos difieren ligeramente entre los miembros de la familia PC, hemos incluido una lista de puertos estándar, sus números y sus usos en la figura 2-9.

En algunos casos, el uso es común para la familia completa. Sin embargo, notará que el PCjr y el AT han introducido un pequeño número de cambios a las asignaciones de los números de puerto.

✎ Antes de usar estas direcciones de puerto, lea las descripciones de los *chips* que aparecen en el capítulo 1. El capítulo 7, que cubre el uso de los puertos para la generación de sonido, muestra cómo se pueden utilizar en algunos casos de programación directa del *hardware* para controlar la salida de sonido.

Al escribir en ciertos puertos, se puede interferir en el funcionamiento normal del ordenador, pero también se pueden producir efectos adversos leyendo. No suponga que la simple lectura de un puerto no va a interferir con el funcionamiento normal del ordenador, ni tampoco que lo que es seguro en un modelo PC también lo va a ser para la familia completa. Por ejemplo, el siguiente programa trabaja perfectamente bien en la mayoría de los modelos del PC, pero bloquea un PCjr.

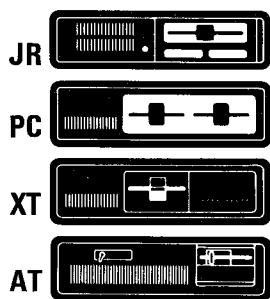
```
10 FOR I = 50 TO 75
20   IF I = 64 THEN PRINT "¿Qué ocurre ahora?"
30   PRINT I, INP (I)
40 NEXT I
```

Este programa utiliza simplemente el comando INP del BASIC para leer datos de un puerto (INP representa entrada a través de un puerto). El programa trata de leer los datos de los puertos 50 a 75; aparentemente, es una acción inocua. Sin embargo, este pequeño programa bloquea el PCjr cuando alcanza el puerto 64, aunque trabaja sin problemas con los otros PC. Las razones están encerradas en los detalles de diseño del *hardware*, y para ser honestos, no los conocemos. Sin embargo, es interesante conocer este tipo de curiosidades.

¿COMO UTILIZA LAS INTERRUPCIONES EL 8088?

Siempre que un dispositivo de *hardware*, o un programa, necesita la atención de la CPU, envía una señal, o instrucción, llamada interrupción, al microprocesador, identificando el tipo de tarea particular que quiere realizar. Cuando el microordenador recibe la señal de interrupción, generalmente para todas las otras actividades, activa una rutina almacenada en memoria, llamada *controlador* o *gestor de interrupción*, que está asociada al número particular de la interrupción. Después de que la rutina de tratamiento de la interrupción ha realizado su tarea, la actividad del ordenador continúa donde estaba cuando ocurrió la interrupción.

Hay tres categorías principales de interrupciones. En primer lugar, hay interrupciones generales por la circuitería del ordenador en



	Descripción	Rango		
		PCjr	PC/XT	AT
	Controlador DMA (8237A-5)	n/a	000-00F	000-01F
	Controlador de interrupciones (8259A)	020-027	020-021	020-03F
	Timer (8253-5; 8254.2 en AT)	040-047	040-043	040-05F
	PPI (8255A-5; 8255-5 en PCjr)	060-067	060-063	n/a
	Teclado (8042)	n/a	n/a	060-06F
	Registro de página de DMA (74LS612)	n/a	080-083	080-09F
	Registro de máscara de NMI (interrupción no enmascarable)	0A0-0A7	0A	070-07F
	Controlador de interrupción 2 (8259A)	n/a	n/a	0A0-0BF
	Generador de sonido (SN76496N)	0C0-0C7	n/a	n/a
	Controlador de DMA 2 (8237A-5)	n/a	n/a	0C0-0DF
	Borrado/reset del coprocesador matemático	n/a	n/a	0F0-0F1
	Coprocesador matemático	n/a	n/a	0F8-0FF
	Joystick (controlador de juegos)	200-207	200-20F	200-207
	Unidad de expansión	n/a	210-217	n/a
	Impresora paralelo (secundaria)	n/a	n/a	278-27F
	Puerto serie (primario)	2F8-2FF	3F8-3FF	3F8-3FF
	Puerto serie (secundario)	n/a	2F8-2FF	2F8-2FF
	Tarjeta de prototipos	n/a	300-31F	300-31F
	Disco duro	n/a	320-32F	1F0-1F8
	Impresora paralelo (primaria)	n/a	378-37F	378-37F
	SDLC (comunicación bisíncrona secundaria sólo en AT)	n/a	380-38F	380-38F
	Comunicación bisíncrona (primaria)	n/a	n/a	3A0-3AF
	Adaptador monocromo/impresora	n/a	3B0-3BF	3B0-3BF
	Adaptador color/gráficos	n/a	3D0-3DF	3D0-3DF
	Controlador de disco	0F0-0FF	3F0-3F7	3F0-3F7

Figura 2-9. Puertas y direcciones de las puertas usadas en los ordenadores PCjr, PC/XT y AT

respuesta a algún acontecimiento, tal como la presión de una tecla del teclado. Estas interrupciones están manejadas por el *chip* controlador de interrupciones (8259), el cual las prioriza por orden de importancia antes de enviarlas a la CPU para que actúe. En segundo lugar, hay interrupciones que son generadas por la CPU como resultado de alguna actuación inusual producida por el programa, como, por ejemplo, una división por cero. Y en tercer lugar, hay interrupciones generadas deliberadamente por los programas como

una forma de invocar subrutinas almacenadas en RAM o ROM. Estas interrupciones, a menudo llamadas interrupciones por *software*, son comunes en las rutinas de servicio de la ROM-BIOS y del DOS. (● Estas rutinas están descritas a fondo en los capítulos comprendidos entre el 8 y el 18.) Siempre es posible cambiar el *software* de las rutinas de tratamiento de las interrupciones, o escribir otras nuevas si la aplicación lo requiere.

Además de estas interrupciones, hay también un tipo especial de interrupción, llamada interrupción no enmascarable (NMI), que se utiliza para solicitar la atención inmediata de la CPU. A menudo indica que se ha producido una emergencia, como, por ejemplo, una caída de voltaje, o un error de memoria. Cuando se envía una NMI, como tiene la prioridad más alta, la CPU la atiende antes que al resto de las interrupciones.

Siempre que se genera una interrupción, el causante de la interrupción no necesita conocer la dirección de memoria de la rutina encargada de su tratamiento, sólo necesita conocer el número asociado a la interrupción. Este número apunta a una tabla almacenada en las posiciones de memoria más bajas, que contiene la dirección segmentada de la subrutina encargada del tratamiento de la interrupción. A la dirección inicial de la rutina encargada del tratamiento de la interrupción se llama *vector de la interrupción*, y la tabla se llama *tabla de vectores de interrupción*. Esta tabla es supervisada normalmente por el BIOS y el DOS. (● Discutiremos todo esto más en el capítulo 3.) Cuando se crean nuevas subrutinas de tratamiento de interrupciones, tenemos que usar números de interrupciones y vectores que ya existen, o bien es necesario asignar otros nuevos.

Las interrupciones guardan automáticamente los valores contenidos en los registros (CS) e (IP). De esta forma, el ordenador puede volver al sitio donde estaba trabajando cuando se produjo la interrupción. Además, el proceso de la interrupción guarda el registro de estado en la pila y borra el indicador de interrupción (IF), previniendo temporalmente la actuación de otras interrupciones. Normalmente, una subrutina de tratamiento de la interrupción termina tan pronto como es posible, normalmente con la ejecución de unas pocas instrucciones. Existe una instrucción especial de retorno de la interrupción, IRET, que realiza esta función; es análoga a la instrucción RET usada con llamadas, o subrutinas. IRET restablece también los indicadores y los registros CS e IP.

Es bastante común enlazar subrutinas hechas en ensamblador a programas, o a lenguajes de programación, de tal forma que se pueda realizar el acceso a las rutinas de servicio del DOS y del BIOS y

reforzar la eficacia de un programa. Para realizar este tipo de rutinas de interfaz, y en especial las que llaman a rutinas de servicio del DOS o del BIOS, es necesario programar en lenguaje ensamblador. Aunque para la mayor parte de las necesidades, estos interfaces consisten en llamadas a subrutinas sencillas, o en llamadas de interrupción mediante la instrucción INT. Sólo los lenguajes de programación más avanzados incluyen la posibilidad de creación de rutinas que permitan el tratamiento de interrupciones y el uso de la instrucción IRET.

El *software* en ROM



La ROM de arranque 48

La ROM-BIOS 50

Vectores de interrupción 51

Referencia de las direcciones bajas de memoria 58

Los identificadores de la versión de la ROM y de la máquina 67

La ROM-BASIC 70

Las extensiones de la ROM 71

Los cartuchos de *software* 72

La cabecera del cartucho 73

Comentarios 75

Se utiliza el *software* para hacer funcionar un ordenador. Conseguir que funcione y se mantenga funcionando es mucho más fácil si parte del *software* está implementado permanentemente en el interior del ordenador. Esta es la razón de la existencia de los programas en la ROM. (ROM son las iniciales de *Read Only Memory*, que se traduce por memoria de sólo lectura.) Esta memoria está registrada permanentemente en la circuitería de los *chips* del PC y no puede ser alterada, borrada o perdida.

Los PC vienen con una cantidad sustancial de ROM, que contiene los programas y los datos que son necesarios para activar y hacer funcionar al ordenador y sus dispositivos periféricos. La ventaja de tener los programas fundamentales del ordenador almacenados en la ROM es que están allí —implementados en el interior del ordenador— y no hay necesidad de cargarlos en la memoria desde el disco, de la misma forma en que se carga el DOS. Debido a que están siempre residentes, los programas en ROM son muy a menudo los cimientos sobre los que se construye el resto de los programas (incluyendo el DOS).

Hay cuatro elementos diferentes en las ROM de la familia PC de IBM: los programas de arranque, que hacen el trabajo de poner en marcha el ordenador; la ROM-BIOS, que es un acrónimo para indicar el sistema de entradas/salidas básico y que está formada por una colección de rutinas en lenguaje máquina, que proporciona los servicios de soporte para las operaciones del ordenador; la ROM-BASIC, que proporciona el núcleo del lenguaje de programación BASIC, y las extensiones ROM, que son programas que se añaden a la ROM principal cuando se conectan al ordenador ciertos equipos especiales. Vamos a examinar cada uno de estos cuatro elementos principales en lo que falta de este capítulo.

El bloque situado en la zona más alta de la memoria se reserva para almacenar los programas en ROM, comenzando en el segmento F000 hexadecimal. Los diferentes modelos de la familia PC usan diferentes cantidades de este espacio de 64K, dependiendo de la complejidad de sus necesidades. Por ejemplo, el modelo original del PC, con su *hardware* relativamente simple, utiliza sólo 40K de los 64K del bloque F para los programas en ROM, mientras que el PCjr y el AT, con su *hardware* mucho más complejo, usan completamente el espacio de 64K.

LA ROM DE ARRANQUE

El primer trabajo de los programas en ROM tiene que ser supervisar la puesta en marcha del ordenador. A diferencia de otros

aspectos de la ROM, las rutinas de arranque están poco relacionadas con la programación de la familia PC, pero vale la pena comprender lo que hacen.

Son varias las tareas realizadas por las rutinas de puesta en marcha. Por ejemplo, ejecutan una comprobación rápida de la fiabilidad del ordenador (y de los programas de la ROM) para estar seguros de que todo está trabajando en orden; inician los *chips* y el equipo estándar conectado al ordenador; ponen la tabla de los vectores de interrupción, comprueban qué equipo opcional está conectado y, si está conectado un controlador de disco, terminan cargando el sistema operativo situado en ese disco.

El test de la fiabilidad, parte de un proceso conocido como POST (*Power On Self Test*), es un primer paso importante que permite estar seguro de que el ordenador se encuentra listo. Todas las rutinas del POST son bastante breves, en tiempo, excepto las que se dedican a realizar los test de memoria, que pueden ser fastidiosamente largos, cuando el ordenador contiene una gran cantidad de memoria.

El proceso de inicialización es ligeramente más complejo. Una rutina pone los valores por defecto de los vectores de interrupción. Estos valores por defecto apuntan hacia las rutinas dedicadas al tratamiento de interrupciones estándar, que se encuentran localizadas en el interior de la ROM-BIOS, o apuntan hacia rutinas que no hacen nada, y que los programas suplirán después. Otra rutina de inicialización determina qué equipo está conectado al ordenador, y entonces sitúa un registro de él en localizaciones estándar de la parte baja de la memoria. (Se discutirá esta lista con más detalle en este capítulo.) La forma de adquirir esta información varía de modelo a modelo; por ejemplo, **PC** en el PC se recoge en su mayor parte de las posiciones de dos bancos de microinterruptores que tiene la placa del sistema ordenador; **JR** en el PCjr se determina por inspección lógica y la comprobación (de hecho, el programa de inicialización sondea a todas las posibles opciones y espera respuesta); **AT**, la información se lee de un área de memoria no volátil especial (que puede ser fijada por los programas de diagnóstico).

Con cualquier método que se utilice, la información del estado queda registrada y almacenada de la misma forma en todos los modelos, de manera que los programas pueden monitorizarla. Las rutinas de inicialización comprueban también los nuevos equipos y las extensiones de la ROM. Si encuentran algo, dan momentáneamente el control a las extensiones de la ROM, de forma que puedan inicializarse a sí mismas. Posteriormente, las rutinas de inicialización continúan ejecutando las rutinas de puesta en marcha que quedan (veremos más sobre esto después).

La parte final del procedimiento de puesta en marcha, después de los test del POST, el proceso de inicialización y la incorporación de las extensiones de la ROM, se denomina *cargador boot-strap*, y es una rutina corta que se emplea para cargar un programa almacenado en el disco. En esencia, el cargador *boot-strap* intenta leer un registro, llamado registro de arranque, que está localizado en un disco, y si es satisfactorio el proceso, pasa el control del ordenador al programa que está incluido en aquel registro. Este programa tiene la tarea de cargar el resto del programa del disco. Normalmente, este programa es el sistema operativo de disco DOS, pero podría ser un programa diferente y autocargable, como, por ejemplo, el simulador de vuelo de Microsoft. Si el cargador *boot-strap* no puede leer el registro cargador del disco, activa simplemente el «cassette» BASIC implementado en la ROM. (Para los equipos que no son miembros de la familia ampliada de PC, se visualiza el mensaje de error *non-boot*.) Tan pronto como ocurre uno de estos dos procesos, el procedimiento de puesta en marcha del sistema termina, y los otros programas se encuentran listos para funcionar.

□ **NOTA:** *Las extensiones de la ROM pueden alterar, o evitar, el proceso de arranque.* **JR** *Como veremos al final de este capítulo, este proceso suele ser más normal en alguno de los cartuchos de software del PCjr.*

LA ROM-BIOS

La ROM-BIOS es la parte de la ROM que está en activa durante todo el tiempo que está trabajando el ordenador. El papel de la ROM-BIOS es proporcionar los servicios fundamentales que se necesitan para que se puedan realizar todas las operaciones del ordenador. En la mayor parte de las ocasiones el BIOS controla los dispositivos periféricos del ordenador, tales como la pantalla, el teclado y los controladores de disco. Cuando se utiliza el término BIOS en su sentido más riguroso, se hace referencia a los programas encargados de realizar el control de dispositivos. Estos programas transforman un simple comando, tal como «leer algo de un disco», en todos los pasos necesarios que permitan realizar de forma efectiva dicha acción, incluyendo la detección de errores y su corrección. En el sentido más amplio, BIOS no sólo hace referencia a las rutinas que son necesarias para controlar los dispositivos del PC, sino también hace referencia a las rutinas que contienen información, o realizan tareas, que son fundamentales en otros aspectos del funcionamiento del ordenador, como, por ejemplo, guardar la hora y el día.

Puede imaginarse que los programas del BIOS están entre los programas personales (incluyendo el DOS) y el *hardware*. En efecto, esto quiere decir que, por un lado, recibe las solicitudes efectuadas por los programas para realizar los servicios de entrada/salida. Estos servicios se llaman desde los programas mediante combinación de dos números: uno, el de la interrupción (que indica el sujeto que ha solicitado un servicio, como, por ejemplo, la impresora), y el otro, un número de servicio (que indica el servicio específico que se debe realizar). Por otro lado, el BIOS se comunica con los dispositivos *hardware* del ordenador (pantalla, controladores de disco, etc.), usando los códigos de control específico que requiere cada dispositivo. En esta actividad, el BIOS también maneja algunas interrupciones *hardware* producidas por algún dispositivo para atraer la atención del procesador. Por ejemplo, siempre que se presiona una tecla, el teclado genera una interrupción que reconoce el BIOS.

De todo el *software* de la ROM, los servicios del BIOS son probablemente los más interesantes y útiles para los programadores; de hecho, hemos dedicado cinco capítulos a estas prestaciones del BIOS desde el capítulo 8 al 13. Dado que se tratará a fondo más adelante, saltaremos sobre las discusiones específicas de qué es lo que hacen los servicios del BIOS y focalizaremos la atención sobre cómo trata el BIOS los procesos de entradas y salidas del ordenador.

Vectores de interrupción

La familia IBM PC, como todos los otros ordenadores basados en la familia 8086 de microprocesadores de Intel, se controla a través del uso de interrupciones, que pueden generarse o bien por *hardware* o bien por *software*. Las rutinas de servicio del BIOS no son una excepción; a cada una se le asigna un número de interrupción al que hay que llamar cuando se desea utilizar el servicio.

Cuando ocurre una interrupción, el control del ordenador pasa a la subrutina de tratamiento de la interrupción, que a menudo está almacenada en la ROM del sistema (una rutina de servicio del BIOS no es nada más que un programa de gestión de interrupciones). El programa de gestión de la interrupción se llama cargando su dirección segmentada en el registro que controla el flujo del programa: el registro CS (segmento de código) y el registro IP (puntero de instrucciones), pareja de registros que se conoce como el par CS:IP. Las direcciones segmentadas utilizadas para localizar las rutinas de tratamiento de las interrupciones se llaman *vectores de interrupción*.

Los vectores de interrupción se colocan, previamente, durante el proceso de puesta en marcha del sistema para apuntar hacia las

rutinas dedicadas al tratamiento de las interrupciones en ROM. Se almacenan en una tabla en la memoria RAM como si fueran pares de palabras, con la porción de la dirección relativa, en primer lugar, y la porción del segmento, en segundo lugar (el 8088 las almacena en la memoria en orden inverso; véase la página 31 para encontrar una explicación del formato de almacenamiento invertido de las palabras). Los vectores de interrupción se pueden cambiar para apuntar a una nueva rutina de tratamiento de la interrupción, simplemente localizando el vector y cambiando su valor.

Como regla general, las interrupciones de la familia PC se pueden dividir en siete categorías: microprocesador, *hardware*, *software*, DOS, BASIC, direccionamiento y uso en general.

Las interrupciones de microprocesador, llamadas a menudo interrupciones lógicas, están integradas en el propio microprocesador. Cuatro de ellas (las interrupciones 0, 1, 3 y 4) se generan por el propio microprocesador, y otra (la interrupción 2, que es la interrupción no enmascarable) se activa mediante una señal generada por uno de los dispositivos externos.

Las interrupciones del hardware están implementadas en el *hardware* del PC. Ocho de estas interrupciones están cableadas en el interior del microprocesador, o en la placa principal del sistema, y no pueden ser cambiadas. Todas las interrupciones del *hardware* son supervisadas por el *chip* 8259A PIC. Los códigos reservados son 2, 8, 9 y del 11 al 15.

Las interrupciones del software están incorporadas dentro del diseño del PC y son parte de los programas de la ROM-BIOS. Las rutinas del BIOS activadas mediante estas interrupciones no pueden cambiarse, pero los vectores que apuntan a estas rutinas sí que pueden ser cambiadas para apuntar hacia rutinas diferentes. Los códigos reservados son 5, 16 al 28 y 72.

Las competencias del DOS

Después de que el cargador boot-strap pasa el control al registro de arranque del disco, este chequea para ver si el DOS está almacenado en el disco, buscando dos ficheros de programa ocultos, llamados IBMBIO.COM e IBMDOS.COM. Si los encuentra, los carga en la memoria junto con el intérprete de comandos del DOS, COMMAND.COM. Durante este proceso, las partes opcionales del DOS, tales como los controladores de los dispositivos instalables, también pueden ser cargados.

El fichero IBMBIO.COM contiene ampliaciones a la ROM-BIOS. Estas ampliaciones pueden ser cambios, o adiciones, de las operaciones

básicas de E/S, que incluyen a menudo correcciones de la ROM-BIOS existente, o nuevas rutinas para nuevos equipos, o cambios personalizados de las rutinas estándar ROM-BIOS. Ya que son parte del *software* del disco, las rutinas del IBMBIO.COM proporcionan una forma conveniente para modificar la ROM-BIOS. Todo lo que se necesita, junto con las nuevas rutinas, es que los vectores de interrupción de las rutinas previas se cambien para apuntar hacia las nuevas localizaciones de memoria, donde se encuentran las nuevas rutinas. Siempre que se añada algún dispositivo al ordenador, sus programas de soporte se pueden incluir en el fichero IBMBIO.COM, o también como un controlador del dispositivo instalable, eliminando la necesidad de reemplazar los *chips* de la ROM. (☛ Véase apéndice A para leer más sobre controladores de dispositivos.)

Puede imaginarse que las rutinas de la ROM-BIOS son el *software* del sistema del más bajo nivel disponible, que permiten realizar las operaciones de entrada y salida más fundamentales, y que se pueden todavía considerar primitivas. Las rutinas del IBMBIO.COM, que son extensiones de la ROM-BIOS, están esencialmente al mismo bajo nivel, proporcionando también funciones básicas. Por comparación, las rutinas del IBMDOS.COM son más sofisticadas, y se puede pensar en ellas como las que ocupan el nivel siguiente a los lenguajes de programación, que son de nivel superior.

El fichero IBMDOS.COM contiene las rutinas de servicio del DOS. Los servicios del DOS, como los servicios del BIOS, se pueden llamar desde los programas a través de un conjunto de interrupciones, cuyos vectores están situados en la tabla de vectores de interrupción que se encuentran en las posiciones bajas de la memoria. Una de las interrupciones del DOS, la interrupción 33 (21 hex), es particularmente importante, porque, cuando se llama, da acceso a un gran grupo de rutinas secundarias, llamadas funciones del DOS. Las funciones del DOS proporcionan un control sobre las operaciones con los ficheros del disco. Todos los procesos del disco estándar —formato de *diskettes*, lectura y escritura de datos, abrir, cerrar y borrar ficheros, efectuar búsquedas en el directorio— están incluidos en las funciones del DOS y proporcionan la base a otros programas del DOS de nivel más alto, tales como FORMAT, COPY y DIR. Los programas pueden utilizar los servicios del DOS cuando es necesario un control mayor de las operaciones de E/S que el permitido por los lenguajes de programación normales y cuando se es reacio a profundizar en el nivel del BIOS. Los servicios del DOS son una parte muy importante de este libro, y les hemos dedicado los cinco capítulos comprendidos entre el 14 y el 18, ambos inclusive.

El fichero COMMAND.COM es la tercera de las partes más importantes del DOS, al menos desde un punto de vista utilitario. Este fichero contiene las rutinas que interpretan lo que se escribe a través del teclado, cuando se está en modo comando del DOS. Compara las palabras tecleadas con una tabla de nombre de comandos. COMMAND.COM puede diferenciar entre comandos internos que son parte del fichero COMMAND.COM, tales como RENAME, o ERASE, y comandos externos, tales como los programas de utilidad del DOS (como DEBUG), o alguno de nuestros propios programas. Actúa sobre las

entradas, ejecutando las rutinas requeridas por los comandos internos, o buscando en el disco los programas requeridos y cargándolos en la memoria. La información completa sobre el fichero **COMMAND.COM**, y de cómo trabaja, es interesante, y por su importancia merece ser investigada, de la misma forma que los programas del DOS. Le recomendamos que lea el manual de «Referencia Técnica del DOS» o *El IBM PC a fondo* para encontrar información adicional.

Las interrupciones del DOS se encuentran siempre disponibles cuando el DOS está en uso. Algunos programas y lenguajes de programación utilizan estos servicios proporcionados, a través de las interrupciones del DOS, para manejar sus operaciones básicas, especialmente las E/S del disco. Los códigos reservados van del 32 al 255 (del 32 al 96 son los que se utilizan, los otros no se aconseja tocarlos).

Las interrupciones del BASIC son asignadas por el propio BASIC y están siempre disponibles cuando se utiliza el BASIC. Los códigos reservados van del 128 al 240.

Las interrupciones de direccionamiento son una parte de la tabla de los vectores de interrupción y se utilizan para almacenar direcciones segmentadas. No hay interrupciones, o subrutinas de tratamiento, asociadas con estas interrupciones. En particular, están asociadas con tres tablas muy importantes: la de inicialización del video, la tabla base del disco y la de caracteres gráficos. Estas tablas contienen parámetros que utiliza la ROM-BIOS en procedimientos de puesta en marcha y para la generación de caracteres gráficos. Los códigos reservados van del 29 al 31, el 68 y el 73 (**JR** 68 y 73 se utilizan sólo en el PCjr).

Las interrupciones de uso general están establecidas para los programas para su uso temporal. Los códigos reservados van del 96 al 103.

Los vectores de interrupción están almacenados en las localizaciones más bajas de la memoria; la primera localización de memoria contiene el vector para la interrupción número 0, y así sucesivamente. Como cada vector tiene una longitud de dos palabras, se localiza cualquier interrupción particular en la memoria multiplicando su número de interrupción por cuatro. Por ejemplo, el vector para la interrupción 5, que es la interrupción de servicio de escritura en pantalla, tendría el byte de dirección relativa de 20 ($5 \times 4 = 20$). Se pueden examinar los vectores de interrupción representando este número decimal en notación hexadecimal y utilizando **DEBUG** (que sólo acepta valores hexadecimales). En el caso de la interrupción 5, la

localización 20 se representa mediante el valor hexadecimal 14, y los siguientes comandos:

```
DEBUG  
D 0000:0014 L 4
```

que mostrarán cuatro bytes, en hexadecimal, como éstos:

```
54 FF 00 F0
```

Convirtiéndolos en una dirección segmentada y teniendo en cuenta el almacenamiento invertido, se puede ver que el vector de interrupción para el punto de entrada en la ROM de la rutina de servicio de escritura en pantalla (interrupción 5) es F000:FF54. La misma instrucción DEBUG se puede utilizar para encontrar otros vectores de interrupción de la misma manera.

La figura 3-1 es un listado de las interrupciones principales y de las localizaciones de sus vectores. Estas son las interrupciones que los programadores suelen encontrar más útiles. De los capítulos 8 al 18 se encuentran más detalles relacionados con la mayoría de estas interrupciones. Las interrupciones que no están mencionadas en esta lista están, en su mayor parte, reservadas para los desarrollos futuros de IBM.

Cambio de los vectores de interrupción

El interés principal de la programación con vectores de interrupción no es leerlos, sino cambiarlos, de forma que apunten a una nueva rutina de tratamiento de la interrupción. Para hacer esto, se debe de escribir una rutina que realice una función diferente a la que realizan las interrupciones estándar de la ROM-BIOS, o del DOS, almacenar la rutina en RAM, y entonces asignar una nueva dirección al vector de la interrupción ya existente en la tabla.

Un vector puede ser alterado byte por byte utilizando lenguaje ensamblador, o utilizando una instrucción del lenguaje de programación como el POKE del BASIC. En algunos casos puede haber peligro de que ocurra alguna instrucción en la mitad de la realización del cambio del vector. Si no está interesado en esta casuística, puede utilizar el POKE. Si está interesado en evitarla, hay dos formas distintas para cambiar un vector, mientras se toman precauciones contra su utilización cuando se está en la mitad del proceso del cambio.

En el primer método es cambiar el vector suspendiendo las interrupciones mientras se realiza el cambio; para ello se utiliza una instrucción de inhibición de interrupciones (CLI). CLI suspende todas las interrupciones, excepto la interrupción no enmascarable

Interrup.				Interrup.			
Dec	Hex	Dirección	Uso	Dec	Hex	Dirección	Uso
0	0	0000	Generada por la CPU cuando se realiza una división por cero	25	19	0064	Invoca a la rutina de puesta en marcha del BIOS
1	1	0004	Usada para ejecutar programas paso a paso (como con DEBUG)	26	1A	0068	Invoca al servicio de hora y fecha del BIOS
2	2	0008	Interrupción no enmascarable; en el PCjr NMI tiene algún uso especial	27	1B	006C	Interrupción generada en el teclado, actúa bajo el BIOS
3	3	000C	Usada para poner puntos de ruptura en los programas (como con DEBUG)	28	1C	0070	Interrupción generada en cada tic de reloj, se activa una rutina si está creada
4	4	0010	Usada cuando un resultado aritmético se desborda	29	1D	0074	Apunta a la tabla de parámetros de control del video
5	5	0014	Invoca a la rutina de servicio de escritura en pantalla del BIOS	30	1E	0078	Apunta a la tabla base del disco
8	8	0020	Tic de reloj generado por <i>hardware</i>	31	1F	007C	Apunta a la parte alta de los caracteres gráficos del video
9	9	0024	En la mayoría de los modelos, generada por la acción del teclado; simulada en el PCjr para compatibilizar el modelo	32	20	0080	Invoca al servicio de «programa terminado» del DOS
13	D	0034	Generada durante el retorno vertical del haz en el CRT, para el control del video	33	21	0084	Invoca a todos los servicios de tipo función del DOS
14	E	0038	Señales de atención del <i>diskette</i> (p. ej., señalar finalización)	34	22	0088	Si está creada, se invoca una rutina de interrupción al final de un programa bajo el DOS
15	F	003C	Usada para el control de la impresora	35	23	008C	Si está creada, se invoca una rutina de interrupción producida por teclado bajo el DOS
16	10	0040	Invoca al servicio de visualización del video del BIOS	36	24	0090	Si está creada, se invoca una rutina de interrupción cuando se produce un error crítico bajo el DOS
17	11	0044	Invoca al servicio de listado de equipo del BIOS	37	25	0094	Invoca el servicio de lectura de <i>diskette</i> del DOS
18	12	0048	Invoca al servicio de tamaño de memoria del BIOS	38	26	0098	Invoca al servicio de escritura del <i>diskette</i> del DOS
19	13	004C	Invoca a los servicios de <i>diskette</i> del BIOS	39	27	009C	Termina el programa, pero lo mantiene en memoria bajo el DOS
20	14	0050	Invoca al servicio de comunicaciones del BIOS	68	44	0110	Apunta a la parte baja de los caracteres gráficos del video; sólo en el PCjr
21	15	0054	Invoca a los servicios de cassette de cinta magnética del BIOS	72	48	0120	Invoca el programa para transformar el teclado del PCjr en el teclado del PC
22	16	0058	Invoca al servicio de teclado estándar del BIOS	73	49	0124	Apunta a la tabla de transformación para los dispositivos suplementarios del teclado
23	17	005C	Invoca al servicio de impresora del BIOS				
24	18	0060	Activa o desactiva el lenguaje ROM-BASIC				

Figura 3-1. Interrupciones principales utilizadas en la familia de los ordenadores personales de IBM

(NMI). Se supone que la NMI sólo será usada para señalar una situación verdaderamente urgente, como puede ser una situación de tipo “incendio-en-el-ordenador”, pero desafortunadamente se ha venido utilizando también para situaciones muy ordinarias, tales como la señalización de una acción en el teclado del PCjr. Como consecuencia, aun cuando el enmascaramiento de las interrupciones con CLI ofrece una seguridad razonable contra la irrupción en la mitad del cambio de un vector de interrupción, no es perfecta.

Le mostraremos dos ejemplos de este primer método —que permiten poner un vector de interrupción con las interrupciones suspendidas—. El primer ejemplo pone el vector con dos instrucciones MOV, que mueven las dos palabras del vector a su lugar:

```
XOR    AX,AX                ; acumulador a cero
MOV     ES,AX               ; registro segmento a cero
CLI                      ; suspende las interrupciones
MOV     WORD PTR ES:36,XX   ; mueve la parte de dirección relativa al vector
MOV     WORD PTR ES:38,YY   ; mueve la parte de segmento del vector
STI                      ; activa las interrupciones
```

Aunque esta técnica es correcta, se corre el pequeño riesgo de que venga una NMI entre las dos instrucciones MOV (aunque es un riesgo muy pequeño). El riesgo se puede reducir combinando los dos MOV en una simple instrucción de movimiento (MOVS). El uso de la instrucción de movimiento de cadena es mucho más pesado, porque requiere inicializar varios registros. Lo usaré para dar un ejemplo de forma alternativa de producir el mismo resultado que en el primer ejemplo.

; primero se ajustan los registros para el movimiento repetido

```
XOR     DI,DI                ; obtiene una palabra cero
MOV     ES,DI                ; ajusta el párrafo a (= 0)
MOV     DI,36                ; ajusta la dirección relativa a número
MOV     SI,XXXX              ; ajusta la dirección relativa de
MOV     CX,2                 ; número de palabras
CLD                          ; ajusta el sentido hacia adelante
```

; ahora hacemos el movimiento con las interrupciones suspendidas

```
CLI                      ; suspende las interrupciones
REP     MOVSW                ; movimiento repetido de palabras
STI                      ; reactiva las interrupciones
```

Le he mostrado dos formas diferentes para cambiar un vector de interrupción siguiendo el método de «hágalo usted mismo». El otro método es permitir que lo haga el DOS, empleando la rutina de servicio del DOS número 37, que fue diseñada con este propósito. Hay dos ventajas muy importantes al permitir que el DOS varíe las

interrupciones en lugar de hacerlo por nosotros mismos. Una ventaja es que el DOS realiza la tarea de poner el vector en su lugar de la forma más segura posible. La otra ventaja es de más alcance. Con la aparición del *chip* procesador 80286 en el modelo AT, la familia PC está empezando a pasar al campo en que elementos tan familiares como los vectores de interrupción y los registros de segmento no son lo que solían ser. Usar un servicio del DOS, para poner un vector de interrupción en lugar de ponerlo nosotros mismos, es una de las formas con la que podemos reducir el riesgo de que nuestros programas sean incompatibles con las nuevas máquinas o los nuevos sistemas operativos.

Aquí hay un ejemplo de cómo usar el servicio 37 del DOS para poner un vector de interrupción:

MOV	DX,XX	; carga la parte de dirección relativa del vector
MOV	DS,YY	; carga la parte de segmento del vector
MOV	AH,37	; pide la función de ajuste de interrupción
MOV	AL,9	; cambia la interrupción número 9
INT	33	; interrupción de llamada a función DOS

Este ejemplo muestra, de la forma más simple posible, cómo se puede utilizar el servicio del DOS. Sin embargo, ilustra una dificultad importante y sutil: se tienen que cargar una de las direcciones en el registro DS (segmento de datos) —lo que bloquea el acceso normal a los datos a través del registro DS—. Para evitar este problema de una manera útil, he aquí una forma de hacerlo. He aquí un ejemplo real de mis programas «Norton Utilities»

PUSH	DS	; salva el segmento de datos en curso
MOV	DX,OFFSET PGROUP:XXX	; obtiene la dirección relativa del vector
PUSH	CS	; mueve nuestro propio segmento de código...
POP	DS	; ...al segmento de datos
MOV	AH,37	; pide la función de ajuste de interrupción
MOV	AL,9	; cambia la interrupción número 9
INT	33	; interrupción de llamada a función DOS
POP	DS	; realmacena nuestro segmento de datos original

Referencia de las direcciones bajas de memoria

Muchas de las operaciones del PC están controladas por datos que se encuentran en las posiciones bajas de la memoria, particularmente en las dos áreas adyacentes de 256 bytes, que empiezan en 400 y 500 hex.

Los datos se almacenan en esas áreas desde el BIOS durante el proceso de puesta en marcha. Aunque el control de estos datos se

supone que está reservado al BIOS, los programas pueden inspeccionarlos, o cambiarlos. Aunque no desee usar la información del área de control del BIOS, es importante su estudio, porque revela gran cantidad de detalles sobre el funcionamiento interno de la familia PC.

Para evitar confusión sobre estas direcciones bajas de la memoria, tenga en cuenta que la dirección de memoria 400 puede también estar expresada en formato segmentado, como 0040:0000 ó 0000:0400. Las tres notaciones se refieren exactamente a la misma posición.

El área de información de control

Algunas de las posiciones de memoria de las áreas 400 y 500 hex son particularmente interesantes. La mayoría de ellas contiene datos vitales para la operación de varias rutinas de servicio del BIOS y del DOS. En algunos casos, los programas pueden restituir la información almacenada en esas localizaciones, invocando una interrupción del BIOS; en todos los casos se puede acceder a la información directamente. Para chequear fácilmente los valores de estas localizaciones en su propio ordenador, use el DEBUG o el BASIC.

Para usar el DEBUG, introduzca estos comandos:

```
DEBUG
D 0:XXXX L 1
```

En este ejemplo, XXXX representa la dirección en hexadecimal que quiere examinar. El L 1 le dice al DEBUG que visualice un byte. Para ver dos o más bytes, introduzca después de la instrucción L el número de bytes (en hexadecimal) que desee ver.

Para visualizar datos con el BASIC, puede utilizar el programa que se muestra abajo, haciendo las sustituciones necesarias para *dirección.en.hex* y *número.de.bytes.en.decimal*:

```
10 DEF SEG = 0
20 FOR I = 0 TO número.de.bytes.en.decimal - 1
30   VALOR = PEEK(&Hdirección.en.hex + I)
40   IF VALOR < 16 THEN PRINT "0"; ' es necesario para encabezar con cero
50   PRINT HEX$ (VALOR); " ";
60 NEXT I
```

Se han listado las direcciones más útiles en las siguientes páginas. Todas ellas están dadas en hexadecimal.

410 (una palabra de 2 bytes). Esta palabra almacena los datos de la lista de equipos, que es recogida por el servicio de listado de equipos, interrupción 17 (11 hex). El formato de esta palabra aparece

mostrada en la figura 3-2, establecida para el PC y el XT, y ciertas partes pueden aparecer con un formato diferente en los modelos posteriores, incluyendo el PCjr.

JR



412 (un byte). Este byte es utilizado únicamente en el PCjr para contar el número de errores detectados en la conexión del teclado por infrarrojos. Otros modelos utilizan este byte solo durante la inicialización. Un byte interesante, pero que no tiene ningún significado útil desde el punto de vista de programación.

413 (una palabra de dos bytes). Contiene el tamaño de memoria utilizable en Kb. En el PCjr devuelve la cantidad de memoria que queda libre después de reservar la memoria de pantalla. En otros modelos esta palabra tiene un significado muy diferente: representa el tamaño total de la memoria. Sin tener en cuenta el modelo, el uso de esta palabra tiene el mismo propósito: le dice cuánta memoria se puede utilizar. El servicio de interrupción del BIOS número 18 (12 hex) escribe el valor en esta palabra.

417 (dos bytes de bits de estado del teclado). Estos bytes son utilizados para controlar la interpretación de las acciones del teclado por las rutinas de la ROM-BIOS. Cambiando estos bytes, se cambia el significado de las pulsaciones de las teclas. Puede cambiar libremente el primer byte, en la localización 417, pero no es una buena idea cambiar el segundo byte. (**JR** Un tercer byte, referido únicamente al PCjr, está localizado en la 488.) Véanse páginas 154 y 230 para ver los valores de estos bytes.

419 (un byte). Este byte está reservado para controlar las entradas alternativas al teclado. Está destinada para usos futuros.

41A (una palabra de dos bytes). Esta palabra apunta a la cabecera del *buffer* del teclado del BIOS situado en la dirección 41E, sitio donde se almacenan las acciones de las teclas hasta que son utilizadas.

41C (una palabra de dos bytes). Esta palabra apunta a la cola del *buffer*.

41E (32 bytes, usados como 16 elementos de dos bytes). El *buffer* del teclado es utilizado para almacenar las 16 últimas acciones realizadas en el teclado, hasta que son leídas por los servicios del BIOS a través de la interrupción 22 (16 hex). Este es un *buffer* de cola de espera circular; ésta es la razón por la cual hay dos punteros para indicar la cabecera y la cola (en 41A y 41C). No es prudente enredar con estos datos.

43E (un byte). Este byte indica si los *diskettes* necesitan ser calibrados antes de buscar una pista. Los bits 0 al 3 corresponden a las unidades 0 a la 3. Si un bit vale 0, es necesaria la recalibración. Generalmente encontrará que está puesto a 0 si ha

Bit															Significado	
F	E	D	E	B	A	9	8	7	6	5	4	3	2	1	0	
XX	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	Número de impresoras instaladas
.	X	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1 si está instalada la impresora serie
.	.	X	.	.	.	.	.	.	.	.	.	.	.	.	.	1 si está instalado el <i>joystick</i>
.	.	.	XXX	.	.	.	.	.	.	.	.	.	.	.	.	Número de puertas serie RS-232
.	.	.	.	.	X	.	.	.	.	.	.	.	.	.	.	0 si está instalado el <i>chip</i> de DMA; el
.	.	.	.	.	.	X	.	.	.	.	.	.	.	.	.	DMA está instalado en todos los
.	.	.	.	.	.	.	.	XX	.	.	.	.	.	.	.	modelos, menos en el PCjr
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	Número de unidades de <i>diskette</i> - 1:
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	00 = 1 unidad; 01 = 2 unidades;
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	10 = 3 unidades; 11 = 4 unidades
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	(véase bit 0).
.	.	.	.	.	.	.	.	.	XX	.	.	.	.	.	.	Modo video inicial: 01 = color a 40
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	columnas en el PCjr; 10 = color a 80
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	columnas; 11 = 80 columnas mono-
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	cromo para el resto de los modelos
.	.	.	.	.	.	.	.	.	.	XX	.	.	.	.	.	Placa de RAM del sistema: 00 = 16K;
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	01 = 32K; 11 = 64K (normal para to-
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	dos los modelos, a pesar de la proba-
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	ble instalación de placas del sistema
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	de 256K)
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	No utilizado; podría ser 0
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1 si está presente alguna unidad de
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	disco (si ocurre, véanse los bits 7 y 6)

habido algún problema durante la utilización más reciente de la unidad del disco. Por ejemplo, el bit de recalibración será 0 si trata de solicitar el directorio (DIR) de una unidad sin *diskette*, y pulsa C en respuesta a la pregunta:

ERROR: Unidad no preparada, leyendo la unidad B
Cancelar, Reintentar, Ignorar?

43F (un byte). Este byte devuelve el estado del motor del *diskette*. Los bits 0 al 3 corresponden a las unidades 0 a la 3. Si el bit es 1, el motor del *diskette* está funcionando.

440 (un byte). Este byte almacena la cuenta atrás hasta que el motor del *diskette* se detiene. La cuenta se pone a 37 (aproximadamente 2 segundos) al principio de cada operación del *diskette*. Con cada señal de reloj, la cuenta es decrementada. El motor del *diskette* se detiene cuando la cuenta llega a cero.

441 (un byte). Este byte indica el estado del *diskette*, cada bit representa un tipo de error particular (☛ véase la figura 3-3). Un bit

Bit								Significado
7	6	5	4	3	2	1	0	
X	.	.	.	.	.	.	.	<i>Diskette</i> fuera de tiempo: falló la respuesta en tiempo
.	X	.	.	.	.	.	.	Fallo en la búsqueda de la pista
.	.	X	.	.	.	.	.	Fallo del <i>chip</i> controlador del <i>diskette</i>
.	.	.	X	.	.	.	.	Comprobación de la redundancia cíclica (CRC): error en los datos
.	.	.	.	X	.	.	.	Error en el DMA del <i>diskette</i>
.	.	.	.	.	X	.	.	Sector no encontrado: <i>diskette</i> dañado o no formateado
.	.	.	.	.	.	X	.	No encontrada la marca de dirección en el <i>diskette</i>
.	.	.	.	.	.	.	X	El comando del <i>diskette</i> solicitado es inválido

Figura 3-3. Byte de codificación del modo video, situado en la dirección 449 hex

de valor 1 señala que ha ocurrido un error; un valor de 0 indica que no ha ocurrido ningún error.

442 (siete bytes). Estos siete bytes almacenan la información del estado del controlador del *diskette*.

En la dirección 449 hex comienza un área de 30 bytes, utilizada para el control del video. El BIOS utiliza esta área para guardar la información crítica del video. Los programas pueden inspeccionar cualquiera de estos datos, sin problemas, pero en la mayoría de los casos es arriesgado modificarla. El cambio de alguno de estos datos puede causar interferencias, produciendo irregularidades en la operación del ordenador —mis propios experimentos han producido resultados asombrosamente raros—. Los únicos bytes que parecen ser seguros y útiles de cambiar son los campos de localización del cursor. (☛ Para leer más sobre los cursores, véase la dirección 450 H y la página 104.)

449 (un byte). Un valor de 0 a 10, o de 13 a 15 en este byte especifica el modo de video activo (☛ véase la figura 3-4). Este es el mismo código del modo de video utilizado en los servicios de video del BIOS. (☛ véase el capítulo 9 para encontrar más información sobre estos servicios y el capítulo 4 para leer información general concerniente a los modos del video.)

Los programas en BASIC pueden leer este byte para detectar el modo de video con estas instrucciones.

```
DEF SEG = 0
MOD0.VIDEO = PEEK(&H449)
```

* ajusta a cero el registro DS
* mira la posición hexadecimal 449

☛ Véase la página 89 para encontrar una discusión especial sobre los modos 4 y 5 en BASIC.

Código	Significado	Código	Significado
0	Texto a 40 columnas, sin color	9	Gráficos de resolución media, 16 colores (no en el adaptador de color/gráficos estándar)
1	Texto a 40 columnas, 16 colores	10	Gráficos de alta resolución, 4 colores
2	Texto a 80 columnas, sin color	13	Gráficos de resolución media, 16 colores
3	Texto a 80 columnas, 16 colores	14	Gráficos de alta resolución, 16 colores
4	Gráficos de resolución media, 4 colores	15	Gráficos de alta resolución especial, 4 colores
5	Gráficos de resolución media, sin color (4 grados de gris)		
6	Gráfico de alta resolución, 2 colores		
7	Modo de adaptador monocromo		
8	Gráficos de baja resolución, 16 colores (no en el adaptador de color/gráficos estándar)		

Figura 3-4. Byte de codificación del modo de video, situado en la dirección 449 hex

44A (una palabra de dos bytes). Esta palabra almacena la anchura de la pantalla en columnas de texto. La anchura de las columnas se almacena en el equivalente hexadecimal de 20, 40 u 80 columnas (el modo de video 8, gráficos de baja resolución, tiene una anchura de texto de 20).

44C (una palabra de dos bytes). Longitud de regeneración de la pantalla. Este es el número de bytes utilizado por una página de pantalla. Varía con el modo.

44E (una palabra de dos bytes). Dirección relativa del comienzo de la memoria de pantalla. De hecho, esta dirección indica la página que se está usando, dando su dirección relativa.

450 (ocho palabras de 2 bytes). Estas palabras dan la posición del cursor para las ocho pantallas, comenzando con la página 0. El primer byte de cada palabra da la columna (0 a 19, 39 ó 79) y el segundo byte da la fila (0 a 24). La posición del cursor se puede controlar modificando esta información. Para los lenguajes de programación que no implementan el control del cursor, ésta puede ser una manera de controlar el cursor sin crear un interfaz en lenguaje ensamblador con las rutinas del BIOS.

Cuando cambie el dato de este byte, observe que el cambio no tiene efecto inmediatamente, sino que espera a la siguiente salida de pantalla. Para comprobar esto, entre en el DEBUG, e introduzca este comando:

F 0:450 L 2 8 8

El cursor salta a la fila 8, columna 8 después de que pulse *Intro*. Hay que decir, necesariamente, que no es una buena técnica de programación, pero debe conocer cómo funciona.

460 (una palabra de dos bytes). Estos dos bytes almacenan el tamaño del cursor, basado en el número de líneas que tiene el cursor. El primer byte da la línea de exploración final; el segundo byte, la línea de exploración de comienzo. A diferencia de los campos de posición del cursor, el cambio de esos valores no altera el cursor automáticamente.

462 (un byte). Este byte almacena el número de página que se está visualizando en ese momento.

463 (una palabra de dos bytes). Esta palabra contiene la dirección del puerto del *chip* controlador del video 6845. Normalmente contiene la dirección 3D4 hex.

465 (un byte). Este byte contiene el modo de CRT.

466 (un byte). Este byte contiene la máscara de bits de la paleta de colores. ■ Para leer más sobre las paletas, véase la página 87.

467 (cinco bytes). Estos bytes son utilizados por el control del cassette.

46C (cuatro bytes almacenados como dos palabras de dos bytes, pero tratados mediante un número de cuatro bytes). Este área se utiliza como contador de un reloj maestro, que se incrementa con cada señal de reloj. Empieza a contar a las 0 horas. Cuando el contador alcanza el equivalente a 24 horas, se vuelve a poner a cero, y el byte de la dirección 470 hex se pone a uno. El DOS, o el BASIC, calcula la hora a partir de este valor y selecciona el tiempo, poniendo en este campo la cantidad apropiada. La rutina 26 (1A hex) del BIOS proporciona este valor.

470 (un byte). Este byte indica que ha ocurrido un paso por cero del reloj. Cuando el contador del reloj pasa la medianoche (y se pone a 0), este byte se pone a 1, lo que significa que la fecha debe de ser incrementada. El valor es puesto a 1 por la rutina de señal de reloj para indicar que ha pasado la medianoche. Se vuelve a poner a cero, cuando se lee el reloj, y se utiliza la interrupción 26 (1A hex). Esta puesta a cero automática se basa en la suposición de que cualquier programa que lea el reloj incrementará la fecha cuando lea esta señal.

□ *NOTA: Este byte se pone a uno a medianoche y no es incrementado nunca. No hay indicación acerca de si pasan dos medianoches antes de que se lea el reloj.*

471 (un byte). Este byte se utiliza para indicar una acción de interrupción desde teclado dentro del BIOS. Si el bit es 1, se ha presionado la tecla break.

472 (una palabra de dos bytes). Estos bytes se utilizan para señalar que está en curso una reinicialización producida desde el teclado. Cuando el sistema se reinicializa desde el teclado con la combinación de teclas Ctrl-Alt-Del, esta palabra contiene el valor 1234 hex mientras dura el proceso de la inicialización. Este hecho es una verdadera curiosidad.

473 (un byte). Reservado para IBM. Este byte se pone a 24 hex cuando IBM declara reparto de dividendos.

474 (cuatro bytes). Esta área se utiliza sólo en el PCjr para el control especial del *diskette*.

478 (8 bytes en dos campos de 4 bytes). Estos bytes se utilizan sólo en el PCjr para controlar las señales de temporización (para la impresora paralelo y la puerta serie (o impresora serie)).

485 (un byte). Este bit almacena el carácter que se debe repetir si pulsa la tecla repeat. Únicamente en el PCjr.

486 (un byte). Este byte se utiliza para temporizar la espera inicial antes de que empiece la acción de la tecla repeat. Únicamente en el PCjr.

487 (un byte). Este byte se utiliza para almacenar el código de la función Fn. Únicamente en el PCjr.

488 (un byte). Este byte es el tercer byte de estado del teclado que sólo se aplica al PCjr. (Los otros dos bytes de estado del teclado situados en las localizaciones 417 y 418 hex, y se utilizan también en el resto de los modelos, incluyendo el PCjr.) El significado de los bits para este byte se muestra en la página 154.

500 (un byte). Este byte es utilizado por el DOS y el BASIC para controlar la operación de escritura en pantalla. Hay tres valores hexadecimales posibles almacenados en esta localización:

- 00 Indica estado OK
- 01 Indica que se está efectuando una operación de impresión de pantalla
- FF Indica que un error ha ocurrido durante una operación de impresión de pantalla

504 (un byte). Este byte es utilizado por el DOS cuando un sistema de *diskette* único, tal como un XT o un PCjr, imita a un sistema de dos *diskettes*. El valor indica si la unidad real está actuando como unidad A o unidad B. Los valores usados son:

- 00 Actuando como unidad A
- 01 Actuando como unidad B

510 (una palabra de dos bytes). Esta área es utilizada por el BASIC para almacenar el valor del segmento de datos (DS) por

JR



defecto. Este es el puntero de segmento de datos por defecto del BASIC.

El BASIC nos permite usar nuestro propio valor de segmento de datos con la instrucción «DEF SEG = *valor*». (La dirección relativa del segmento se especifica mediante las funciones PEEK o POKE.) También se puede devolver el segmento de datos a su posición por defecto, usando la instrucción DEF SEG *sin* = *valor*. Aunque el BASIC no proporciona una forma simple para encontrar el valor por defecto almacenado en esta localización, se puede obtener utilizando esta pequeña rutina:

```
DEF SEG = 0
SEGMENTO.DATOS = PEEK(&H511) * 256 + PEEK(&H510)
```

□ *NOTA: El BASIC administra sus propios datos internos basándose en el segmento de datos por defecto. Intentar cambiarlo es como sabotear la operación del BASIC.*

512 (cuatro bytes). Esta área la utiliza el BASIC como un vector de interrupción, que apunta a la rutina de servicio de la interrupción del tic de reloj del BASIC.

□ *NOTA: Para funcionar mejor, el BASIC hace correr al reloj del sistema cuatro veces más rápido que su velocidad estándar; para ello, el BASIC debe reemplazar la rutina de interrupción de reloj del BIOS por la suya propia. La rutina de interrupción del BIOS estándar es llamada por el BASIC con la cadencia normal; esto es, una vez por cada cuatro señales rápidas. Véase la página 167.*

516 (cuatro bytes). Esta área es utilizada por el BASIC como un vector de interrupción que apunta a la rutina de manejo de la tecla *break* del BASIC.

51A (cuatro bytes). Esta área es utilizada por el BASIC como un vector de interrupción que apunta a la rutina de manejo de los errores de *diskette* del BASIC.

Area de comunicaciones entre aplicaciones

Aunque el control de la información del BIOS abarca la parte más grande y más importante del área del bloque 400, el área de comunicaciones entre aplicaciones, o ICA, también está localizada en esta zona. El ICA es un área reservada de 16 bytes, comprendida entre la posición 4F0 y la 4FF, y se utiliza para almacenar datos que pueden ser compartidos por diferentes programas. Es particularmente útil para los programas del DOS que se ejecutan por separado, pero que tienen que dejar información para otros programas posteriores.

El ICA no se utiliza demasiado. Entre los pocos programas, que se conoce, que lo utilizan, están algunas versiones de Comunicaciones Asíncronas de IBM, Lifetree's Volkswriter, y TimeMark. Debido a que cualquier programa puede almacenar datos en el ICA, en un momento determinado puede haber información de varios programas.

Si sus programas hacen uso del ICA, le recomiendo que incluya un *checksum* y también una firma, de forma que pueda identificar los datos del ICA que son suyos y los que han sido cambiados por otros programas.

□ AVISO: El ICA está localizado en los 16 bytes de la localización 4F0 a la 4FF. Un error tipográfico en algunas ediciones del manual de «Referencias técnicas de IBM» los sitúan entre la 500 y la 5FF. Esto es incorrecto.

Los identificadores de la versión de la ROM y de la máquina

Al estar los programas del BIOS en memoria fija, no pueden cambiarse fácilmente cuando se necesita hacer ampliaciones o correcciones. Esto significa que los programas en ROM deben haber sido comprobados muy cuidadosamente antes de congelarlos en los *chips* de memoria. Aunque hay una alta probabilidad de que existan serios errores en los programas contenidos en la ROM de un sistema, IBM ha seguido una trayectoria estupenda; hasta ahora, en los programas contenidos en ROM de la familia PC sólo se han encontrado unos pocos, y relativamente poco importantes, errores.

Las diferentes versiones del *software* en ROM podrían presentar un pequeño desafío a los programadores que describen las diferencias que afectan a las características operativas de sus programas. Pero un desafío mayor para los programadores es que algunos miembros de la familia (el PCjr y el AT, en particular) utilizan un conjunto ligeramente diferente de rutinas para la ROM-BIOS de las que vienen con el IBM PC estándar.

Para asegurar que los programas están trabajando con los programas de la ROM apropiados y con el ordenador correcto, IBM ha proporcionado dos marcas de identificación, que están permanentemente disponibles al final de la memoria en la ROM del sistema. Una marca identifica la fecha de salida de la ROM, y puede usarse para identificar la versión del BIOS, y la otra da el modelo de la máquina. Estas marcas están siempre presentes en las propias máquinas de IBM, y también las de los fabricantes de unos pocos miembros de la familia PC.

La fecha de salida de la ROM se puede encontrar en un área de 8 bytes, comprendida entre F000:FFF5 y F000:FFFC (dos bytes antes del byte de ID de la máquina). Está formada por caracteres ASCII, en el formato americano de la fecha; por ejemplo, 06/01/83 representa el 1 de junio de 1983. Esta marca de salida es una característica común de los ordenadores personales de IBM, pero está presente solamente en unos pocos compatibles de IBM. Por ejemplo, los ordenadores Compaq no lo tienen.

El único propósito de las fechas en la marca de salida es para identificar las diferentes versiones de la ROM (véase la figura 3-5). Sugiero que lo utilice sólo cuando haya encontrado un problema que requiera que sus programas trabajen de forma diferente, dependiendo de las distintas ROM. (Los programas, probablemente, suelen identificar el byte de ID de la máquina para responder a las características particulares de los diferentes modelos.)

Marca de salida	Máquina
04/24/81	PC original
10/19/81	PC revisado (contiene algunos errores)
08/16/82	XT original
10/27/82	Paso del PC al nivel BIOS XT
11/08/82	PC portátil original
06/01/83	PCjr original
01/10/84	AT original

Figura 3-5. Fechas de salida de las diferentes versiones de ROM

Puede mirar la fecha de salida con el DEBUG, utilizando los siguientes comandos:

```
DEBUG
D F000:FFF5 L 8
```

O desde BASIC utilizando esta técnica:

```
10 DEF SEG = &HF000
20 FOR I = 0 TO 7
30   PRINT CHR$(PEEK(&HFFF5 + I));
40 NEXT
50 END
```

Aquí tiene un ejemplo de lo que puede encontrar: Yo tengo tres PC y cada uno viene con diferente ROM. Uno tiene la versión 04/24/81, otro la versión 10/19/81 y el último la versión 10/27/82.

La versión de adaptación del BIOS está disponible bajo ciertas circunstancias; por ejemplo, la unidad de mejora de PC que transforma un PC a las especificaciones de XT viene con la adaptación 10/27/82. Ocasionalmente, la adaptación del BIOS está también disponible por separado.

El ID de la máquina es un byte localizado en F000:FFFE. La figura 3-6 lista los valores de ID publicados para los cinco modelos de IBM PC. Podemos esperar que, probablemente, esta secuencia continúe en los futuros modelos.

Tenga cuidado, ya que hay algunas inconsistencias en la forma de asignar los ID de las máquinas. FE fue el valor anunciado originalmente como identificador para el XT y, después, para el PC portátil; todavía algunos XT tienen actualmente la firma del PC: «FF.» En general, no se puede contar con que estas asignaciones de firma sean sólidas; IBM ha trastocado la definición un poco, o bien en la publicación de las firmas, o bien debido a que hayan sido cambiadas realmente. Yo creo, sin embargo, que hay una regla simple que se puede seguir en la interpretación de las firmas de la máquina. En aquellas situaciones en las que las diferencias entre los modelos sean lo suficiente significativas como para requerir que un programa necesite identificar la máquina sin equivocarse, se puede considerar que las firmas son sólidas; por ejemplo, es el caso entre el PCjr y el AT, en los cuales cada uno tiene sus propias características especiales. Pero cuando las variaciones entre los modelos de máquinas son menores, como, por ejemplo, entre el PC original, el PC estándar, el PC-2 (que acepta 256K de memoria en su placa del sistema), el XT y el PC portátil, entonces las firmas pueden variar. Para todos los propósitos prácticos se puede considerar que ambas firmas, la FF y FE, son la identificación de la máquina: más o menos PC estándar.

ID		Máquina
Dec	Hex	
255	FF	PC (el ordenador personal de IBM original)
254	FE	XT y PC portátil
253	FD	PCjr
252	FC	AT

Figura 3-6. Los ID de máquinas de los cinco modelos IBM PC

ID		
Dec	Hex	Máquina
45	2D	Compaq (equivalente al PC)
154	9A	Compaq-Plus (equivalente al XT)

Figura 3-7. Los ID de máquinas no oficiales de los dos modelos Compaq

Es posible que los ordenadores compatibles-IBM puedan ser identificados de la misma forma, pero no conozco la fiabilidad de la información pública. Mis propios programas identifican dos firmas para los dos primeros ordenadores Compaq, pero no se debe considerar esto como oficial.

Puede explorar el byte de ID de la máquina mediante el DEBUG, utilizando los siguientes comandos:

```
DEBUG
D F000:FFFE L 1           ' muestra un byte en la posición especificada
```

Un programa puede inspeccionar este byte utilizando técnicas como ésta:

```
10 DEF SEG = &HF000           ' define el segmento F000 en el registro DX
20 IF PEEK(&HFFFE) = 253 THEN PRINT "Debería ser un Junior"
30 IF PEEK(&HFFFE) = 254 THEN PRINT "Debería ser un XT"
40 IF PEEK(&HFFFE) = 255 THEN PRINT "Debería ser un PC"
50 IF PEEK(&HFFFE) = 252 THEN PRINT "Debería ser un AT"
60 END
```

LA ROM-BASIC

Ahora pasaremos al tercer elemento de la ROM: la ROM-BASIC. La ROM-BASIC actúa de dos formas. Primero, proporciona el núcleo del lenguaje BASIC, que incluye la mayor parte de los comandos y el cimiento fundamental, como, por ejemplo, la gestión de la memoria. Las versiones de disco del BASIC, que vemos en los ficheros de programas BASIC.COM y BASICA.COM, son esencialmente suplementos de la ROM-BASIC, y confían en que la ROM-BASIC haga la mayor parte del trabajo. El segundo papel de la ROM-BASIC es proporcionar lo que IBM llama «cassette» BASIC —que es el BASIC que se activa cuando se pone en marcha el ordenador sin disco.

Cuando se usa alguno de los intérpretes BASIC, tal como el BASIC de cassette, el cartucho de BASIC del PCjr o los BASIC de

disco (BASIC o BASICA), también se utilizan los programas de la ROM-BASIC —aunque no hay nada que permita decir que es así—. Por otra parte, los programas BASIC compilados no hacen uso de la ROM-BASIC.

Esta ROM-BASIC es característica sólo de la familia PC de IBM. Ninguno de los miembros de la familia PC extendida, tal como los ordenadores Compaq, tiene la ROM-BASIC; por ello, las partes equivalentes del BASIC están incluidas en los programas BASIC basados en disco.

LAS EXTENSIONES DE LA ROM

El cuarto elemento de la ROM tiene más que ver con el diseño del PC, que con el contenido de su memoria. El PC fue diseñado para permitir dos tipos de extensiones del *software* interno en la ROM: uno, para extensiones permanentes al *software* de la ROM-BIOS, y el otro, para extensiones previstas para los cartuchos de *software* removibles. Para cada uno de ellos están reservadas áreas especiales de memoria.

Las extensiones permanentes de la ROM-BIOS son programas que operan con la ROM-BIOS incorporada, pero añaden características no soportadas por la ROM-BIOS básica. Usualmente, éstos son programas de soporte para nuevos dispositivos periféricos. El mejor ejemplo de este tipo de extensiones de la ROM es el soporte ROM-BIOS para el disco duro de IBM, que fue introducido con el XT. Otra se encuentra en el Enhanced Graphics Adapter (EGA). Puesto que la ROM-BIOS original no podía anticipar la aparición de los programas de soporte para el *hardware* futuro, las extensiones a la ROM eran, obviamente, una adición necesaria y útil.

Las extensiones permanentes de la ROM-BIOS utilizan dos áreas de memoria. Una es la parte no utilizada del bloque de memoria F, que, desafortunadamente, puede variar de modelo a modelo. En la mayor parte de los modelos, el área de 24K comprendida desde el segmento F000 hasta el F600 está disponible (la ROM-BIOS del disco duro no-IBM de uno de mis PC se conecta en el segmento F400). El otro área de memoria para las extensiones de la ROM es el bloque C de memoria, cuyo párrafo de segmento va desde la C000 a la CFFE. La ROM-BIOS del disco duro del IBM XT se conecta en esta área, el párrafo de segmento C800, y el Enhanced Graphics Adapter de IBM se conecta en el párrafo C000. Aunque las extensiones de la ROM permanentes proporcionadas por IBM tienen localizaciones predecibles, hay siempre alguna posibilidad de conflictos entre extensiones del BIOS proporcionadas por otros fabricantes.

Normalmente, las extensiones de la ROM permanentes están instaladas en el ordenador, conectadas con parte de una tarjeta de expansión, o conectadas en un zócalo de la ROM, que hay disponible en la tarjeta de sistema del ordenador.

Los **cartuchos de *software***, por otro lado, se pretende que se puedan conectar y desconectar libremente. Generalmente, los cartuchos se utilizan de la misma forma que los *diskettes*: para cargar temporalmente programas que tienen un propósito específico. Un área de 128K de memoria, que comprende los bloques de memoria D y E, se reserva para que la utilicen los cartuchos de *software*.

Ambos tipos de extensiones de ROM se integran con el resto de la ROM, durante el proceso de puesta en marcha. Para encontrar las extensiones de ROM, la ROM estándar comienza en el bloque C000 y busca la *signatura* (55 AA hex) en cada bloque de 2K, que es la que identifica las extensiones de la ROM. Cuando aparecen estas *signaturas de identificación*, la rutina de puesta en marcha pasa temporalmente el control de la extensión de ROM, con objeto de que la extensión pueda hacer lo que sea necesario para fusionarse con la operación del ordenador. En este momento, la extensión de la ROM puede hacer lo que quiera, incluyendo apoderarse del control completo del ordenador. Algunos cartuchos de *software* hacen exactamente eso. Sin embargo, la cosa más normal que hace una extensión es simplemente comprobar el equipo que necesita (por ejemplo, una extensión ROM-BIOS para disco duro debe de disparar el disco duro y, metafóricamente, presentarse a través de las áreas de datos de la parte baja de la memoria con el resto del BIOS; así cada sección del BIOS conoce quiénes son sus compañeros de trabajo). Una vez se hace la inicialización, la extensión de la ROM acostumbra a devolver el control al BIOS principal, que entonces termina el proceso de puesta en marcha del ordenador.

Los cartuchos de *software*

Puesto que no se puede dar una descripción completa de las extensiones de la ROM sin discutir los cartuchos de *software*, dedicaremos una breve sección a proporcionar las descripciones esenciales sobre ellas. Encontrará información más detallada en «*Exploring the IBM PCjr*» y en el «Manual de Referencia Técnica de IBM».

Los cartuchos de *software* en ROM contienen programas, almacenados en *chips* de ROM, que se pueden conectar en algún modelo PC que los admita (como, por ejemplo, el PCjr). Cada cartucho de

software puede contener un mínimo de 2Kbytes, o un máximo de 64Kbytes, dependiendo del diseño del *hardware*.

Un cartucho puede aparecer en cualquiera de las seis posiciones de memoria. La posición que elige está escrita en el programa del cartucho. En notación de segmento, las seis posibles localizaciones son D000, D800, E000, E800, F000 y F800. Las cuatro direcciones de los bloques de memoria D y E son las localizaciones de los cartuchos convencionales. Las dos direcciones del bloque F son direcciones superpuestas a la ROM-BIOS, que pueden permitir que un cartucho reemplace temporalmente a la ROM-BIOS que está incorporada al ordenador.

Hay un formato estándar para la cabecera de los cartuchos que utiliza la ROM-BIOS para identificar los cartuchos instalados en la memoria y determinar su contenido. La información almacenada en la cabecera identifica indirectamente también el tipo de cartucho que está conectado y que será utilizado.

La cabecera del cartucho

Cada cartucho comienza con la signature de 2-bytes de la extensión de ROM estándar, 55 AA hex, seguida por un código de longitud de un byte. En los cartuchos y otras extensiones de ROM se da la longitud en unidades de 512 bytes, o 1/2K. Por ejemplo, un cartucho de juego de 8K tiene un código de longitud de 16, mientras que un cartucho de lenguaje BASIC de 32K tiene un código de longitud de 64.

Siguiendo al código de longitud, hay tres bytes que se reservan para una única instrucción de inicialización del cartucho. Los tres bytes que tiene este campo son suficientes para contener una instrucción de tres bytes, o menos; normalmente contienen una instrucción FAR RET de 1 byte, o una instrucción JMP de 3-bytes. La instrucción controla la inicialización que tiene que hacer el cartucho. Por ejemplo, el caso de un cartucho de programa escrito en BASIC (necesita, por tanto, el cartucho de lenguaje BASIC), y esos tres bytes contienen un código especial: la instrucción FAR RET estándar (CB hex), seguida por la signature *invertida* (AA 55 hex). Otros tipos de cartuchos, para evitar que puedan confundirse con un cartucho de programa en BASIC, deben tener valores diferentes al AA 55 hex en los últimos dos bytes de este campo.

A continuación del campo de inicialización, y comenzando en el séptimo byte del cartucho, hay una tabla con los contenidos del DOS, que permiten identificar los programas de comandos del DOS que deben de estar en el cartucho. Si están, se convierten efectivamente en

adiciones a los comandos internos, tales como DIR, COPY y TIME, que están a disposición del DOS. Si no están, el cartucho tendrá una tabla de contenidos *vacía*, en vez de *no* tener ninguna tabla de contenidos (esto se explicará en un momento.) El formato de la tabla de contenidos del DOS es una serie de entradas con nombres de comando, seguidas por un byte a cero.

Las entradas de los nombres de comando están formadas por tres campos cada una: un campo de 1 byte que registra la longitud del nombre; un campo que contiene el número de bytes que forman el nombre del comando, en letras mayúsculas y en caracteres ASCII, y un campo de instrucción de salto de 3-bytes, que produce el salto al programa que lleva a cabo el comando.

❑ **AVISO:** Algunos manuales de «Referencia Técnica de IBM» identifican incorrectamente el último campo de nombres de comando como una dirección relativa de 2-bytes; en realidad, es una instrucción de salto de 3-bytes.

A continuación de la cabecera está el contenido del cartucho —usualmente son programas escritos en lenguaje máquina—. Si el cartucho es un cartucho de programa en BASIC, el contenido es un

Offset	Contenido	Descripción
0	55 AA	Signatura
2	40	Longitud: 40hex = 64*1/2K = 32K
3	E9 1D 00	JMP 0023: salto al principio del código
6	05	Longitud (5) del siguiente nombre de comando
7	"BASIC"	Nombre del comando, en ASCII
12	E9 82 01	JMP 0191: salto al comienzo del BASIC
15	06	Longitud (6) del siguiente nombre de comando
16	"BASICA"	Nombre del comando, en ASCII
22	E9 78 01	JMP 0191: salto al comienzo de BASICA (el mismo comienzo que el BASIC)
25	00	Fin de la tabla de contenidos

Figura 3-8. Ejemplo específico de una cabecera de un cartucho, recogida del cartucho del lenguaje BASIC del PCjr. Observe que la tabla de comandos internos del DOS (DIR, TIME, etc.), que está almacenada dentro del interpretador de comandos, COMAND.COM, es bastante similar a esta tabla de comandos de DOS del cartucho.

programa en BASIC *tokenizado* y almacenado en el mismo formato que se utiliza para almacenar los programas en BASIC en el disco. Tenga en cuenta que el primer byte de un programa BASIC sirve para identificarlo como normal (FF hex) o protegido (FF hex).

COMENTARIOS

Los listados de programas en la ROM podrían llenar volúmenes enteros, y de hecho abarca una gran cantidad de espacio en el «Manual de Referencia Técnica de IBM». Aunque IBM desaprueba la utilización directa de la información encontrada en los listados de la ROM, particularmente del BIOS, puede ser divertido y muy esclarecedor hacerlo en alguna ocasión. Puesto que he hecho un gran esfuerzo en este libro para indicar cuáles son las rutinas de la ROM-BIOS que son seguras de utilizar, también le recomiendo que se lo lea a fondo antes de que se aventure usted mismo.

Video, conceptos básicos

Adaptadores de video 78

La memoria y los adaptadores de video 79

Creación de la imagen de la pantalla 80

Formatos de pantalla 82

Resolución de la imagen 84

Control del modo de video 85

Utilización del color 86

Modos de color-suprimido 89

Color en modo texto y gráficos 89

La memoria de imagen 95

Visualización de páginas en modo texto 97

Visualización de páginas en los modos gráficos 98

Visualización de caracteres en los modos de texto y gráficos 98

Caracteres mapeados en modo texto 99

Control de la pantalla 102

Control directo por *hardware* 105

Consideraciones de compatibilidad 108

Para mucha gente, la pantalla es el ordenador. Los programas se juzgan, a menudo, sólo por su calidad de imagen y diseño gráfico. En este capítulo se verán los tipos de pantallas que utiliza la familia del IBM PC y cómo se generan. Y, lo más importante, se aprenderá a cómo manipular los visualizadores de video para obtener los efectos.

ADAPTADORES DE VIDEO

Para producir la imagen de video, la mayoría de los miembros de la familia PC (incluyendo el PC, el XT y el AT) requiere un adaptador de video —una tarjeta especial que, normalmente, se conecta en uno de los conectores de expansión del ordenador—. Los adaptadores de video fueron diseñados para el PCjr, pero modelos tales como el PC portátil y el Compaq también incorporan adaptadores de video, aunque en estos casos se pueden cambiar los adaptadores.

El adaptador de video conecta el ordenador al monitor a través de un *chip* llamado controlador de CRT. El adaptador también tiene un conjunto de puertos de E/S programables, una ROM generadora de caracteres y memoria RAM para almacenar la información del visualizador.

Hay varios tipos de adaptadores de video, pero fueron diseñados después de que IBM adoptara originariamente los dos adaptadores para el PC: el adaptador color/gráficos (CGA) y el adaptador monocromo. Estos dos adaptadores serán discutidos más ampliamente que los otros.

Los generadores de video operan fundamentalmente según dos modos diferentes, llamados por IBM *modo texto* y *modo gráfico*. El modo texto sólo permite visualizar caracteres, aunque algunos de estos caracteres se pueden ajustar para producir dibujos simples con líneas (■ véase el apéndice C para obtener más información sobre los caracteres.) El modo gráfico se utiliza principalmente para producir dibujos complejos, pero puede reproducir caracteres de texto con una variedad de formas y tamaños igualmente buena.

El adaptador color/gráficos puede operar en ambos modos, texto y gráficos, para producir dibujos y caracteres en varios formatos y colores. Está diseñado para trabajar con todo tipo de pantallas, desde televisores estándar a monitores en color de alta resolución.

Por contraste, el adaptador monocromo sólo puede operar en modo texto, utilizando un conjunto de caracteres ASCII, alfanuméricos y gráficos, que tiene guardados, y visualizándolos en un solo color. Diseñado para aplicaciones de negocios, el adaptador mono-

cromo sólo trabaja con el monitor monocromo de IBM (o sus equivalentes), que es un monitor especial de alta resolución (véase la página 83 para obtener más información sobre monitores.) Algunos usuarios de gestión y profesionales prefieren una pantalla monocroma a una de color, porque es más fácil de leer. Pero eligiendo el monocromo, sacrifican los gráficos y el color, dos valiosas cualidades en cualquier pantalla.

Para superar estas limitaciones, algunos fabricantes de *hardware* han ofrecido variaciones del adaptador monocromo de IBM, tales como el popular adaptador de video Hércules, que combina satisfactoriamente las capacidades gráficas (pero no de color) del adaptador de color/gráficos con las del visualizador de texto de alta calidad del adaptador monocromo, y añade además características propias. La calidad de los gráficos resultantes es mejor que la que puede producir el adaptador de color/gráficos. El *Enhanced Graphics Adapter* (EGA) puede crear de forma similar gráficos en una pantalla monocroma.

Alrededor de dos terceras partes de los PC están equipados con el adaptador monocromo estándar y, por tanto, no tienen capacidad de gráficos, o color. Aunque hay ventajas reales para utilizar el color y los gráficos, la mayoría de los PC sobreviven perfectamente sin ellos. Cuando se proyecta aplicaciones para el ordenador, debe tenerse en cuenta que la mayoría de los ordenadores PC visualizan *sólo* el texto.

La mejor forma de comprender la capacidad de visualización del PC es cubrir las características del adaptador de color/gráficos original, notando donde difiere el adaptador monocromo (mayormente en pequeños detalles). Apuntaremos también dónde han sido hechas las extensiones al adaptador de color/gráficos en el *Enhanced Graphics Adapter* y en el PCjr.

La memoria y los adaptadores de video

La memoria de imagen está localizada físicamente, con el resto de la circuitería de video, en la tarjeta adaptadora. Sin embargo, para la CPU es una parte lógica del espacio de direcciones de la memoria principal del ordenador. Un espacio de direcciones de memoria de 128K está reservado para su utilización por el visualizador en los bloques de memoria A y B, entre las direcciones A0000 y BFFFF hex, pero los dos adaptadores de video originales sólo utilizan dos pequeñas partes de esta área de memoria. El adaptador monocromo se reserva 4K de memoria de visualización localizada en el segmento de dirección B000 hex. El adaptador color/gráficos original reserva 16K de memoria de visualización en la dirección B800. El espacio que

queda, particularmente el bloque de 64K desde A000 hasta B000, se reserva para su utilización por un visualizador avanzado; por ejemplo, el *Enhanced Graphics Adapter*.

El PCjr se desvía de la tradición de la familia, utilizando las localizaciones bajas de la memoria principal para su memoria de pantalla. Sin embargo, la circuitería especial del PCjr imita estrechamente el adaptador color/gráficos convencional. Esta circuitería, llamada matriz de puertas de video (*Video Gate Array*, VGA), hace que parezca que la memoria de visualización del PCjr esté localizada en la dirección B800 del adaptador color/gráficos. Las referencias al área B800 son reorientadas por la circuitería de la VGA a cualquier localización que esté en uso como memoria de pantalla. El PCjr puede utilizar cualquier parte de los primeros 128K de RAM de la memoria de video; la VGA utiliza la localización real. Para todos los propósitos de programación práctica, el PCjr se tratará como un PC equipado con un adaptador color/gráficos, que utiliza la memoria de visualización, comenzando en la dirección B800.



Creación de la imagen de la pantalla

Los adaptadores de color/gráficos y monocromo almacenan la información de visualización mediante *imagen mapeada en memoria*, llamada así porque cada dirección de la memoria de pantalla corresponde a una localización específica de la pantalla (■ véase la figura 4-1). La circuitería del visualizador lee repetidamente la información de la memoria y la sitúa en la pantalla. La información puede ser cambiada tan rápidamente como el ordenador pueda escribir nueva información de nuestros programas en la memoria. El controlador CRT es la unión entre la memoria de imagen y el monitor, transforma el flujo de bits que recibe de la memoria en puntos de luz en las localizaciones concretas de la pantalla.

Estos puntos, o destellos de luz, son llamado generalmente *pixels* y están producidos por un rayo de electrones que incide en la superficie fosforescente del CRT. El rayo de electrones está producido por un cañón de electrones que recorre la pantalla línea a línea. A medida que el rayo se mueve a través y hacia abajo en la pantalla, en un camino fijo llamado barrido de monitor, el controlador de CRT genera una señal de control de video que apaga y enciende los *pixels*, dibujando en pantalla el modelo de bits contenido en la memoria.

La circuitería de video refresca la pantalla 60 veces por segundo, haciendo que el cambio de imágenes aparezca claro y estable. Al final

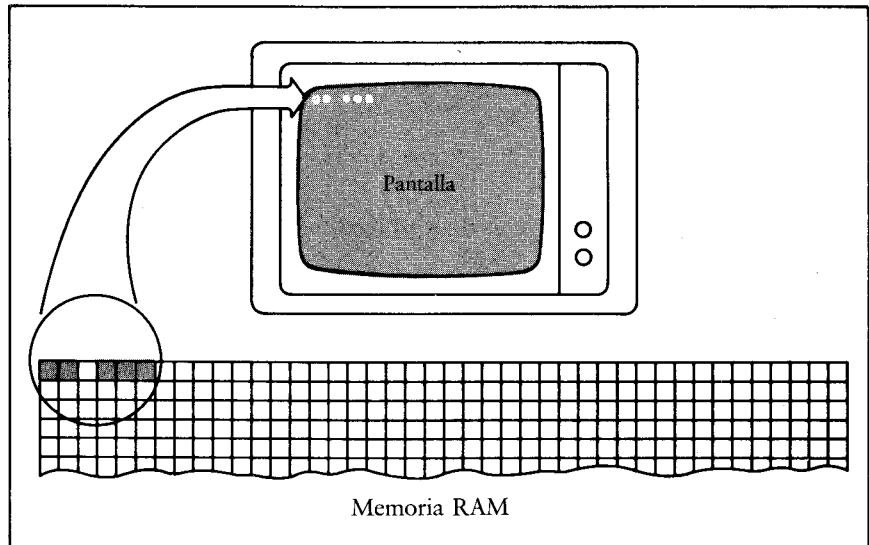


Figura 4-1. Memoria de imagen mapeada

de cada ciclo de refresco de pantalla, el rayo de electrones se debe mover de la esquina derecha inferior a la esquina izquierda superior de la pantalla para empezar un nuevo ciclo. Este movimiento es denominado retorno *vertical*. Durante el retorno, el rayo apaga y la información no puede ser escrita en la pantalla.

El período de retorno vertical (1.25 milisegundos) es importante para el programador por una razón que requiere explicación. El diseño especial de doble puerto de la memoria del adaptador de visualización da a la CPU y al controlador de CRT igual acceso a la memoria de imagen. Esto permite que la CPU y el controlador de CRT accedan a la memoria de imagen a la vez. Si la CPU quiere acceder a un byte de memoria, mientras el controlador de la CRT está escribiendo en la pantalla, puede aparecer brevemente en la pantalla un efecto de «nieve». Sin embargo, si instruimos a la CPU para que acceda a la memoria sólo durante el retorno vertical, cuando el controlador de la CRT no está accediendo a la memoria, entonces la «nieve» puede ser eliminada. En los sistemas que utilizan el adaptador color/gráficos, nuestros programas pueden comprobar un bit de estado, llamado *señal de sincronismo vertical*, en uno de los puertos de E/S del adaptador (localización 3DA hex). Este bit se pone a «uno» al principio de un retorno y se pone a «cero» al final. Durante estos 1.25 milisegundos de pausa podemos escribir tantos datos como sea posible en la memoria de video. Al final de este

repaso el controlador del CRT puede escribir estos datos en la pantalla sin «nieve». Esta técnica es útil para cualquier aplicación que requiera una rápida sucesión de imágenes claras.

FORMATOS DE PANTALLA

Originalmente había ocho formatos de pantalla, o modos, definidos para los ordenadores personales de IBM. Se han añadido otros siete, o más. Los modos de pantalla definen las características del visualizador, incluyendo la cantidad de texto que puede ser visualizado, la resolución, o detalles de los gráficos, y los colores. El adaptador color/gráficos (CGA) permite varias opciones de formatos diferentes en los modos de texto y gráficos. El adaptador monocromo ofrece sólo un único formato de texto en un color. El PCjr y el *Enhanced Graphics Adapter* (EGA) soportan una gran variedad de formatos.

Cada uno de los quince modos que será discutido está identificado por un número de 0 a 16 (● véase la figura 4-2). Los modos 0 a 3 son los modos de texto, y los modos 4 a 16 son los modos gráficos

Modo	Tipo	Tamaño	Colores	Adaptador	Visualización
0	Texto	40 × 25	16 (grises)	CGA, EGA, PCjr	Color mejorado
1	Texto	40 × 25	16 en primer plano, 8 de fondo	CGA, EGA, PCjr	Color mejorado
2	Texto	80 × 25	16 (grises)	CGA, EGA, PCjr	Color mejorado
3	Texto	80 × 25	16 en primer plano 8 de fondo	CGA, EGA, PCjr	Color mejorado
4	Gráficos	320 × 200	4	CGA, EGA, PCjr	Color mejorado
5	Gráficos	320 × 200	4 (grises)	CGA, EGA, PCjr	Color mejorado
6	Gráficos	640 × 200	2	CGA, EGA, PCjr	Color mejorado
7	Texto	80 × 25	b/n	EGA, A. monocr.	Monocromo
8	Gráficos	160 × 200	16	PCjr	Color mejorado
9	Gráficos	320 × 200	16	PCjr	Color mejorado
10	Gráficos	640 × 200	4	PCjr	Color mejorado
11	Aparentemente interno al EGA				
12	Aparentemente interno al EGA				
13	Gráficos	320 × 200	16	EGA	Color mejorado
14	Gráficos	640 × 200	16	EGA	Color mejorado
15	Gráficos	640 × 350	b/n	EGA	Monocromo
16	Gráficos	640 × 350	64	EGA	Color mejorado

Figura 4-2. Características de los quince modos de video

Monitores

El tipo de pantalla o monitor utilizado tiene un efecto importante en el diseño del programa. Algunos monitores no pueden producir color, o gráficos, y algunos producen una calidad de imagen tan pobre que sólo podemos usar el formato de visualización de texto de 40 columnas. Hay algunos tipos de monitor que se pueden utilizar con los ordenadores de la familia PC. Las dos categorías principales son el monitor monocromo y el monitor color, que se pueden dividir en cuatro tipos básicos:

Monitores monocromos de gestión directa: Estos monitores están diseñados para visualizar textos en alta resolución y caracteres gráficos, pero no gráficos de *pixels*. Los monitores monocromos de gestión directa sólo trabajan con el adaptador monocromo. Las pantallas gráficas que se visualizan en otros tipos de monitores no se verán en absoluto en el monitor de gestión directa, a menos que se utilice un interfaz especial, tal como la tarjeta de adaptación de Hércules.

Monitores monocromos de video compuesto. Estos monitores están entre los más extensamente utilizados y menos caros de los disponibles. Están conectados a la salida de video compuesto del adaptador color/gráficos y proporcionan una buena imagen de un color (usualmente verde o ámbar). El Compaq portátil y el IBM PC portátil utilizan este tipo de monitor. Un monitor monocromo compuesto puede visualizar gráficos, pero no colores. Algunos monitores monocromos proporcionan un soporte limitado de «color» con diversos grados de intensidad de gris, reemplazando a los colores. Sin embargo, la mayoría de ellos produce una imagen difícilmente legible, y a veces totalmente inservible cuando se les da una señal de color. No debe confundirse el monitor compuesto con el monitor monocromo de gestión directa. El monitor monocromo compuesto utiliza el adaptador color/gráficos, mientras que el monitor monocromo de gestión directa utiliza el adaptador monocromo.

Monitores de color de video compuesto y televisores. El video compuesto está producido por una señal combinada única, que viaja a través de la salida de video compuesto del adaptador color/gráficos. El monitor compuesto reproduce color y gráficos, pero tiene limitaciones: una visualización de 80 columnas es a menudo ilegible, sólo sirven ciertas combinaciones de color, y la resolución de los gráficos es de baja calidad; así que los gráficos deben simplificarse utilizando los modos gráficos de baja resolución.

Aunque el televisor estándar (en color, o en blanco y negro) es técnicamente un monitor de video compuesto, produce usualmente peor calidad de imagen que el monitor compuesto propiamente dicho. Las representaciones de texto deben constar de 40, o incluso 20, columnas para asegurar que la imagen es legible. Las TV están conectadas a la salida de video compuesto del adaptador color/gráficos, o su equivalente (en el caso del PCjr, hay una salida especial de TV, pero la señal compuesta debe ser convertida con un modulador de RF antes de introducirla en la TV).

Monitores de color RGB. Los monitores RGB están considerados como lo mejor de ambos mundos. Combinan la visualización de textos de alta calidad del monitor monocromo con gráficos y color en alta resolución. RGB significa rojo-verde-azul (*red-green-blue* en inglés), y a estos monitores se les llama monitores RGB porque utilizan cables separados para cada una de las señales de color (a diferencia de los monitores de video compuesto que utilizan un solo cable). Estos cables están conectados a la salida RGB del adaptador color/gráficos, o su equivalente. Un monitor RGB de calidad superior puede producir imágenes más claras y más legibles, seguido sólo por el monitor monocromo de IBM. La calidad de imagen y de color es usualmente muy superior a la que proporciona cualquier pantalla conectada a la salida de video compuesto.

para el adaptador color/gráficos. El modo 7 es el modo de texto monocromo que sólo puede ser utilizado con el adaptador monocromo de IBM (o sus equivalentes). Los modos 8 a 10 fueron introducidos con el PCjr (que utiliza también los modos 0 a 6) y no pueden ser utilizados con los adaptadores de visualización de IBM estándar. Los modos 13 a 16 se aplican al EGA (que también utiliza los modos 0 a 7).

El color puede ser usado en cualquier modo de visualización, excepto en el previsto para el adaptador monocromo (modo 7). A través de los modos disponibles con el adaptador color/gráficos, podemos elegir de 2 a 16 combinaciones de color, incluyendo una selección de tonos grises, llamados modos de color suprimido. Aunque no está permitida ninguna elección de color con el monitor monocromo, hay algunas variaciones en la visualización de caracteres que son ligeramente equivalentes al color: intensidad de brillo y oscuridad, subrayado y video inverso. ■ Estudiaremos la utilización del color con los visualizadores de texto y gráficos, así como las respuestas del adaptador monocromo al color. En la sección titulada *Utilización del color*, en la página 86, se describirán las opciones de color para los modos de video.

Resolución de la imagen

Las imágenes gráficas están construidas a partir de puntos individuales, llamados elementos de dibujo, o *pixels*. La resolución de la imagen está definida por el número de líneas, o líneas de barrido, de la parte superior a la inferior, y por el número de *pixels* de izquierda a derecha en cada línea. El número de líneas que un

monitor puede visualizar está determinado por el *hardware* y las señales de video, de las que tenemos poco, o ningún, control. Para cambiar la resolución de la pantalla, tenemos que cambiar el número de *pixels* de cada línea.

Los modos gráficos del PC tienen tres resoluciones —baja, media y alta— con 160, 320 y 640 *pixels* en cada línea. **JR** La baja resolución (160 × 200 *pixels*) fue introducida con el PCjr, y no está disponible con el adaptador color/gráficos original, o el *Enhanced Graphics Adapter*. Como los caracteres de texto también pueden ser visualizados en los modos gráficos, las resoluciones gráficas, media y alta, tienen un tamaño de texto equivalente (☛ véase la figura 4-3).

Resolución	Pixels	Caracteres
Bajo	160 × 200	20 × 25
Medio	320 × 200	40 × 25
Alto	640 × 200	80 × 25

Figura 4-3. Resolución de caracteres de texto dibujados en los modos gráficos

Una anchura de carácter que se adapta a un formato de 80 columnas por 25 líneas utiliza una resolución de 640 × 200 *pixels*, y un carácter de mayor anchura, que llena un formato de 40 columnas por 25 líneas, utiliza una resolución de 320 × 200 *pixels*. (El visualizador de 80 × 25 caracteres del adaptador monocromo muestra el texto más claro, porque sus caracteres están contruidos con una resolución de *pixels* más alta, 720 × 350).

Adviértase que los gráficos de baja resolución tienen su propio formato de texto de 20 columnas únicamente, que no tiene equivalente en los modos de texto estándar. **JR** El texto de 20 caracteres de anchura, junto con el modo de baja resolución, fueron introducidos con el PCjr, pero no existe en ninguno de los otros adaptadores.

Control del modo de video

Los modos de video están controlados por la ROM-BIOS a través de la interrupción 16 (10 hex), servicio 0. (☛ véase el capítulo 9.) El BASIC proporciona un control completo sobre los modos de video a través de la instrucción SCREEN, pero se refiere a ellos en su propia forma, utilizando los números de modo diferente al de rutinas de la ROM-BIOS. También se pueden controlar algunos de los modos de

Modo	Instrucción BASIC para cambiar de modo	Instrucción DOS para cambiar de modo
0	SCREEN 0,0:WIDTH 40	MODE BW40
1	SCREEN 0,1:WIDTH 40	MODE CO40
2	SCREEN 0,0:WIDTH 80	MODE BW80
3	SCREEN 0,1:WIDTH 80	MODE CO80
4	SCREEN 1,0 o SCREEN 4	n/a
5	SCREEN 1,1	n/a
6	SCREEN 2	n/a
7	n/a	MODE MONO
8	SCREEN 3	n/a
9	SCREEN 5	n/a
10	SCREEN 6	n/a

Figura 4-4. Comandos del BASIC y el DOS utilizados para cambiar los modos de video

video a través del DOS. Pero a nivel de comandos, el DOS funciona en el modo de texto y no hay ningún comando del DOS que cambie a alguno de los modos gráficos, como podemos ver en la figura 4-4.

UTILIZACION DEL COLOR

Hay una gran variedad de colores disponibles en cada modo de visualización, excepto en el modo proporcionado por el adaptador monocromo. Advierta que entre los diversos modos hay diferencias sustanciales en el número de colores disponibles. En esta sección describiremos las opciones de color en los diversos modos de video.

Los colores para las pantallas de visualización del PC están producidos por combinaciones de cuatro elementos: tres componentes de color —rojo, verde y azul— más un componente de intensidad, o brillo. Los modos de texto y gráficos utilizan las mismas opciones de color e intensidad, pero las combinan de diferente forma para producir sus imágenes coloreadas. Los modos de texto, cuya unidad básica es un carácter compuesto por varios *pixels*, usan un byte entero para poner el color, la intensidad y el parpadeo de los caracteres y su fondo. Los modos gráficos, que tienen una unidad básica mucho más pequeña (el *pixel*), usan sólo de uno a cuatro bits para definir el color y la luminosidad, porque el *pixel* no tiene características de parpadeo, o fondo. Se verá cómo poner los atributos de los modos de texto y gráficos en la página 90. Primero, una palabra sobre los colores mismos.

Los números de colores (0 al 15) usados por el BASIC, y utilizados en general para identificar los colores del PC, pueden obtenerse interpretando los cuatro elementos de color como bits de un número binario. Los colores y sus códigos de bits están listados en la figura 4-5. Cuando se utilice un modo de 16 colores, obtendremos todos los colores del 0 al 15. Cuando utilicemos un modo de ocho colores, obtendremos los colores del 0 al 7; esto es, todos los colores sin intensidad de brillo. Con un modo de cuatro colores, obtendremos una selección de cuatro colores, elegidos entre los 16, que se denomina *paleta*. En un modo de dos colores, obtenemos los colores 0 y 7, negro y blanco ordinarios.

Hasta ahora se ha descrito la paleta básica de 16 colores del PC estándar, que está constituida por los tres colores RGB y la intensidad (I). A este esquema de colores básico se le llama IRGB. Se ha añadido una paleta de 64 colores a la familia PC, pero sólo está disponible utilizando el *Enhanced Graphics Adapter* y el *Enhanced Color Display* (la combinación EGA/ECD). Esta paleta de 64 colores está construida a partir de los tres colores estándar (rojo, verde y azul), pero cada color tiene dos señales independientes: una de brillante y una de amortiguada. La notación para la paleta de 64

Intensidad	Rojos	Verde	Azul	Número	Descripción
0	0	0	0	0	Negro
0	0	0	1	1	Azul
0	0	1	0	2	Verde
0	0	1	1	3	Cyan (azul-verde)
0	1	0	0	4	Rojo
0	1	0	1	5	Magenta
0	1	1	0	6	Pardo (o amarillo oscuro)
0	1	1	1	7	Gris brillante (o blanco ordinario)
1	0	0	0	8	Gris oscuro (negro en algunas pantallas)
1	0	0	1	9	Azul brillante
1	0	1	0	10	Verde brillante
1	0	1	1	11	Cyan brillante
1	1	0	0	12	Rojo brillante
1	1	0	1	13	Magenta brillante
1	1	1	0	14	Amarillo (o amarillo brillante)
1	1	1	1	15	Blanco brillante

Figura 4-5. Matriz de colores completa de la familia PC, con los códigos de cuatro bits que los especifican

colores es RrCgBb, donde las letras mayúsculas identifican los colores brillantes. Adviértase que no se habla de dos niveles de intensidad, sino de dos señales de color separadas, que permiten un total de cuatro intensidades para cada uno de los tres colores. Para los rojos, las cuatro intensidades serían Rr (la mayor intensidad), R., r, y.. (sin rojo). Todas las posibles combinaciones de RrCgBb constituyen los 64 colores.

No se tratará la paleta de 64 colores de la combinación EGA/ECD en detalle, porque es bastante rara y especializada y no se adapta realmente a la línea principal de la familia PC. (Si se fuesen a tratar todos los elementos exóticos y particulares de la familia PC, se llenaría un portentoso libro de varias veces el tamaño de éste.) Otra combinación más especializada de adaptador y visualizador, de gráficos profesional de IBM tiene una paleta de 256 colores y una resolución extraordinariamente alta; pero está aún más lejos de la línea principal de la familia PC, así que tampoco se tratará. En cambio se volverá para un estudio más detallado de las paletas de color estándar.

Hay varias cosas que deben tenerse en cuenta cuando se eligen los colores. Si se activan los cuatro elementos de color (IRGB) producen luz blanca. Cuantos más elementos se activen, más brillantes serán los colores, pero también serán menos saturados. Para el ojo, los colores simples puros (rojo, verde y azul) son más intensos visualmente que los colores mixtos (cyan, magenta y amarillo), o las versiones llamadas «intensas» (con brillo) de los colores puros. Aquí hay otros tres factores que se deben considerar al elegir los colores:

- Algunas pantallas no responden al bit de intensidad. Esta deficiencia hace que el color 8 sea el mismo que el color 0, el color 9 el mismo que el 1, y así sucesivamente.
- Cuando una pantalla de visualización monocroma compuesta se utiliza con un adaptador color/gráficos, los colores distintos del negro (0) y el blanco (7) pueden producir información ilegible.
- Finalmente, en los programas que funcionen en un PC, o un XT, con el adaptador monocromo de IBM, debe tenerse en cuenta la forma inusual con que el visualizador monocromo trata los colores (■ véase página 92).

En las siguientes secciones hay importantes ejemplos relacionados con el color.

Modos de color-suprimido

En un esfuerzo por hacer que los modos gráficos sean compatibles con un gran rango de monitores, de color y monocromos, IBM incluye unos pocos modos que no producen color, llamados modos de color-suprimido. Hay tres modos de color-suprimido: modos 0, 2 y 5. En estos modos, los colores son convertidos en grados de grises, o del color que produzca la pantalla. En el modo 5 hay cuatro grados, y una gran variedad de grados en los modos 0 y 2. El color está suprimido en la salida compuesta para monitor del adaptador, pero *no* en su salida RGB. Esta inconsistencia es el resultado de una limitación técnica inevitable.

■ *NOTA: Para cada modo de color-suprimido, hay un modo de color correspondiente, así los modos 0 y 1 corresponden al texto de 40 columnas; los modos 2 y 3 al texto de 80 columnas, y los modos 4 y 5 a los gráficos de resolución media. El hecho de que los modos 4 y 5 inviertan la configuración de los modos 0 y 1, y los modos 2 y 3, donde el modo de color-suprimido viene primero, ha conducido a una complicación en el BASIC. El parámetro de grupo de la instrucción SCREEN del BASIC controla el color. La función de este parámetro es invertir los modos 4 y 5, de forma que la instrucción SCREEN, 1 activa el color en los modos de texto (0, 1, 2 y 3), pero suprime el color en los modos gráficos (4 y 5). Esta inconsistencia puede haber sido un error de programación al principio, pero ahora forma parte de la definición oficial de la instrucción SCREEN. La figura 4-6 muestra la sintaxis de la instrucción SCREEN para los modos 0 al 5.*

Color en modo texto y gráficos

Es necesario darse cuenta de las diferencias en la utilización del color entre los modos texto y gráficos, particularmente las inconsistencias aparentes en la forma en que son manejados los colores de los textos. En modo texto, se tiene control completamente independiente sobre el color de cada posición de carácter: se puede utilizar libremente la paleta de 16 colores del primer plano y la paleta de ocho colores de los fondos. En modo gráfico, se cuenta con un control completo sobre el color de cada *pixel* y sobre el color de algunas operaciones de dibujo de gráficos (como las proporcionadas por el BASIC, por ejemplo).

En teoría, los modos gráficos darían una gama más rica de color. Sin embargo, cuando se escriben textos en un modo gráfico, no se tiene control sobre el color del fondo: está siempre puesto el color de fondo universal. (● véase el tema del valor 0 de la paleta bajo los

Modo	Color suprimido	Color activo
0	SCREEN 0,0:WIDTH 40	
1		SCREEN 0,1
2	SCREEN 0,0:WIDTH 80	
3		SCREEN 0,1:WIDTH 80
4		SCREEN 1,0
5	SCREEN 1,1	

Figura 4-6. Parámetros de color de los modos 0 a 5. Fíjese que los modos 0 a 3 y los modos 4 y 5 tienen formas diferentes.

modos con cuatro colores, página 94). Esto significa que, aunque los modos gráficos proporcionan en principio más control del color, proporcionan menos control del color cuando están visualizando texto. Esta es una característica inherente de los servicios de escritura de texto en los modos gráficos. (☛ véase capítulo 9.)

Control del color en los modos de texto

En los modos de texto cada posición de un carácter en la pantalla está controlada por dos bytes adyacentes en la memoria (☛ véase página 60 para obtener más información acerca de la localización de estos bytes en la memoria). El primer byte contiene el código ASCII de los caracteres que serán visualizados. (☛ véase apéndice C para conseguir un cuadro de caracteres.) El segundo byte controla cómo aparecerá el carácter, especificando sus colores, y así sucesivamente. Este segundo byte se llama atributo del carácter.

Antes de seguir adelante, es necesario explicar un par de términos que pueden presentar alguna confusión. En la terminología del visualizador del IBM PC, los términos *color* y *atributo* son utilizados indistintamente. Aunque hay un significado preciso para estos dos términos, que son distintos hablando rigurosamente, a menudo se encuentran utilizados de manera imprecisa para indicar lo mismo. Para evitar la confusión, piense en ambas palabras como términos ligeramente vagos que se refieren a la forma en que las cosas aparecen en la pantalla y a la codificación de los datos en la memoria que controla la apariencia de los caracteres.

Hay tres componentes en el atributo de los caracteres de texto: el color de primer plano (el color del propio carácter), el color de fondo (el color del área no cubierta por el carácter) y el componente de

parpadeo del carácter. El color de primer plano puede ser cualquiera de los 16 colores del rango completo del PC. El color de fondo puede ser uno de los ocho colores: números de color de 0 al 7 (colores básicos sin intensidad de brillo).

Cada posición de carácter en la pantalla tiene su propio atributo de control, independiente del resto de los caracteres de la pantalla. Los ocho bits del byte de atributo actúan independientemente para controlar un elemento del atributo. El significado de los bits se muestra en la figura 4-7. El atributo utilizado por defecto por el DOS y el BASIC es 07 hex, blanco normal (7) sobre fondo negro (0), sin parpadeo.

Con el PCjr, cuando se utilizan caracteres de texto en un modo gráfico con 16 colores (modos 8 ó 9) se puede utilizar cualquiera de los 16 colores para el color de fondo, pero este color de fondo se aplicará a todos los caracteres de texto escritos en modo gráfico. Aunque esto puede parecer bastante limitado, es bastante útil. Por ejemplo, los caracteres que se visualizaban en la pantalla se hacían invisibles poniendo el color de primer plano igual que el color de fondo. Esta es la forma ideal para permitir que las palabras clave, u otra información confidencial, sean invisibles al entrar en la pantalla. (También hay un modo invisible para el adaptador monocromo; véase página 92.)

La calidad de color varía con el monitor. En algunos casos, los colores brillantes de alta intensidad son claramente legibles cuando se visualizan en un fondo del mismo color, pero sin la intensidad. Por otra parte, algunos visualizadores de color no actúan poniendo la intensidad. Con estos visualizadores, combinaciones legibles en el caso anterior, tal como el amarillo sobre fondo marrón, simplemente son indistinguibles.



Bit								Uso
7	6	5	4	3	2	1	0	
1	.	.	.	.	.	.	.	Parpadeo del carácter
.	1	.	.	.	.	.	.	Componente roja del color de fondo
.	.	1	.	.	.	.	.	Componente verde del color de fondo
.	.	.	1	.	.	.	.	Componente azul del color de fondo
.	.	.	.	1	.	.	.	Componente de intensidad del color del primer plano
.	.	.	.	.	1	.	.	Componente roja del color del primer plano
.	.	.	.	.	.	1	.	Componente verde del color del primer plano
.	.	.	.	.	.	.	1	Componente azul del color del primer plano

Figura 4-7. Código del byte de atributo del color

Aunque los modos gráficos pueden visualizar textos de forma bastante nítida, hay varias ventajas obvias para utilizar los modos texto para la visualización del texto. Quizá la ventaja más importante de los caracteres en modo texto es que pueden ser visualizados más rápidamente que los caracteres gráficos. Esto se debe, en parte, al hecho de que los caracteres en modo texto se toman de una tabla de caracteres, mientras que los caracteres gráficos deben ser dibujados bit por bit desde la memoria. Los modos de texto utilizan menos memoria que los modos gráficos; así tienen memoria extra disponible para visualizar páginas, permitiendo almacenar varias «páginas» de información de texto directamente en la memoria de visualización del video, y visualizarlas, una cada vez, en una sucesión rápida. Hay también más efectos especiales disponibles para los caracteres en el modo de texto. Por una parte, hay un mayor rango para elegir colores. Y por otra, los modos de texto pueden hacer parpadear los caracteres, mientras los modos gráficos no tienen capacidad de parpadeo en absoluto.

Control de atributos en modo monocromo

El modo monocromo (modo 7) utilizado por el adaptador monocromo de IBM tiene una posibilidad limitada de variaciones en el visualizador. Se utiliza el mismo esquema de codificación general de atributos para los caracteres monocromos que el utilizado en los modos 0 al 3.

Los bits de parpadeo, e intensidad, se utilizan en el modo monocromo. Sin embargo, sólo cuatro combinaciones de «color» de primer plano y de fondo producen resultados distintos:

- Blanco sobre negro normal, producidos seleccionando blanco (primer plano bits 111) sobre negro (fondo bits 000), o 07 hex.
- Caracteres subrayados, producidos poniendo el byte de atributo a 01 hex, que selecciona azul (primer plano bits 001), sobre negro (fondo bits 000).
- Video inverso, o negro (primer plano bits 000) sobre blanco (fondo bits 111) producido con 70 hex.
- Caracteres invisibles, creados utilizando negro (primer plano bits 000) sobre negro (fondo bits 000), o 00 hex.

El resto de las combinaciones muestra lo mismo que blanco sobre negro normal, 07 hex. Otras combinaciones de color que pueden parecer lógicas, tales como el invisible blanco sobre blanco o una

combinación video inverso/subrayado, no existen en modo monocromo; sólo existen los cuatro resultados mencionados. Adviértase que los bits de atributo de parpadeo e intensidad son independientes de estas cuatro combinaciones de «color».

Control del color en los modos gráficos

Hasta ahora se ha visto cómo poner color (y el equivalente monocromo del color) en los modos texto. Poner el color en los modos gráficos es un poco diferente. En los modos gráficos (modos 4 a 6 y 8 a 10) cada *pixel* de la pantalla tiene asociado un color. El color se pone de la misma forma que se ponen los atributos en el modo texto, pero hay importantes diferencias. Primero, los *pixels* gráficos no pueden parpadear. Segundo, puesto que cada *pixel* es un punto discreto de color, no hay primer plano y fondo (cada *pixel* es simplemente un color u otro). Cuando se escribe texto en modo gráfico, se utiliza un color para los *pixels* del fondo y cualquier otro color para los *pixels* que forman el carácter.

□ *NOTA: Al usar el modo gráfico en BASIC, da la impresión de que hay un color de fondo para los gráficos. Pero esto es simplemente un convenio que adopta el BASIC: algunos pixels a los que no se les ha asignado explícitamente el color del «primer plano» tienen el color del «fondo». Los servicios de video de la ROM-BIOS (véase capítulo 9) también hacen uso de este convenio para el color de fondo.*

Para cada modo gráfico, hay conjuntos de colores predefinidos, conocidos como paletas. Las paletas estándar pueden ser cambiadas en el PCjr, o el EGA, pero no en el adaptador color/gráficos original. Una vez fijada una paleta, cada color de *pixel* se puede seleccionar entre los colores disponibles, dando valores a los bits de color asignados a cada *pixel*. En un modo de dos colores hay un bit para cada *pixel*; el valor del color del *pixel* se da como 0 ó 1. En un modo de cuatro colores hay dos bits, y los valores del color varían de 0 a 3. En un modo de 16 colores hay cuatro bits, y los valores del color van de 0 a 15. Los valores de color utilizados para definir un *pixel* no son necesariamente los mismos que los números (0 a 15) usados para identificar los colores reales que aparecen en la pantalla.

En el modo 6 de dos colores hay una sola paleta estándar, mostrada en la figura 4-8. En los modos 4 y 5 de cuatro colores hay dos paletas estándar: la paleta 0, mostrada en la figura 4-9, y la paleta 1, mostrada en la figura 4-10. Dos cosas hay que resaltar sobre estas

Bit	Valor	Color
0 0	0	Negro
0 1	1	Blanco

Figura 4-8. Paleta estándar para el modo gráfico de dos colores (modo 6)

Bit	Valor	Color
0 0	0	Negro (por defecto; puede ser cambiado por otro color)
0 1	1	Verde
1 0	2	Rojo
1 1	3	Marrón

Figura 4-9. Paleta 0, una de las dos paletas estándar para los modos gráficos de cuatro colores (modos 4 y 5)

Bit	Valor	Color
0 0	0	Negro (por defecto; puede ser cambiado por otro color)
0 1	1	Cyan
1 0	2	Magenta
1 1	3	Blanco normal

Figura 4-10. Paleta 1, una de las dos paletas estándar para los modos gráficos de cuatro colores (modos 4 y 5).

paletas. Primero, el valor de la paleta 0 puede ser cambiado del negro (color 0) a cualquier color. Segundo, el valor 0 de la paleta es el color del «fondo» y el valor 3 de la paleta es el color del «primer plano», cuando se escriben caracteres de texto. En el modo 10 de cuatro colores hay una paleta estándar.

Esta paleta empareja los valores de paleta 0 a 15 a los números de color reales, como cabía esperar. Recuerde: los modos 8 y 9 están disponibles con el PCjr, y el 13 y 14 sólo con el EGA.

Remapeado de paletas en el PCjr y el EGA

Hasta este punto, se han estudiado los colores estándar que son producidos utilizando las paletas estándar. Con el adaptador color/gráficos original, las asignaciones de colores de la paleta son fijas y no pueden cambiarse. Sin embargo, en el PCjr, y en algún adaptador de video diseñado para proporcionar esta característica, tal como el EGA, las paletas pueden tener remapeados sus colores. Al remapear un color, simplemente se reasigna el valor del color, de forma que al solicitar, por ejemplo, el color 1 (azul) pueda visualizarse el color 4 (rojo).

El mapeado de cualquier valor de paleta requerido, a otro número de color, está bajo el control de la paleta asignada por defecto en ese momento. Las paletas pueden cambiarse en BASIC con las instrucciones de paletas, o con los servicios de video de la BIOS (■ véase página 202).

LA MEMORIA DE IMAGEN

A continuación, se tratará el funcionamiento interno del mapa de pantalla. En esta sección se verá cómo está relacionada la información contenida en la memoria con la pantalla del visualizador. Debe tenerse en cuenta que para los modos de video que tienen su memoria de imagen en el bloque B (color/gráficos modos 0 a 6 y monocromo modo 7), podemos tener guardados nuestros programas de forma segura junto a la memoria de imagen. Esto es verdad incluso para el PCjr, que sólo *parece* usar el bloque B para los modos 0 a 6. Al principio, IBM no quería que nuestros programas manipulasen directamente la memoria de video, pero desde que la mayoría de los programas que valen la pena lo hacen, IBM se ha resignado a ello e intenta que todos los adaptadores de video presentes y futuros sean compatibles. Pero IBM sólo está guardando las apariencias. Para los nuevos modelos, mejorados, tales como los modos del 8 al 10 del PCjr y los modos del 13 al 16 del EGA, IBM está haciendo la memoria de video tan inaccesible como le es posible. En el caso del EGA, la memoria de visualización está localizada teóricamente en el bloque A, pero nuestros programas no pueden encontrarla en aquella dirección. Estaríamos locos si tratásemos de romper esta barrera.

El uso y codificación de la memoria de video varían de acuerdo a los modos de video, de 0 a 10, que sean utilizados. (Recuérdese que los modos 0 al 6 se aplican al adaptador color/gráficos de IBM original, y el modo 7 al adaptador monocromo de IBM. Los modos 8

al 10 fueron introducidos con el modelo PCjr, que también utiliza los modos 0 al 6; estos modos no pueden ser utilizados con el adaptador color/gráficos de IBM estándar, o con alguno de sus equivalentes. Los modos 11 al 16 se aplican sólo al EGA.)

En los modos 0 al 6 y también en el 8 el mapa de visualización ocupa 16Kbytes; en los modos 9 y 10, el mapa de visualización llena 32Kbytes; en el modo 7 del adaptador monocromo usa sólo 4Kbytes. El modo de texto en ambos adaptadores de video, el monocromo y el de gráficos, utiliza menos memoria que el modo gráfico, porque se necesitan sólo dos bytes para almacenar un carácter. (● Obtendrá más información sobre esto en la página 99.) Consecuentemente, un visualizador de texto de 80 por 25 caracteres requiere sólo 4.000 bytes. Un visualizador de gráficos, como se puede ver en la figura 4-11, puede requerir de 16K bytes a 32K bytes, dependiendo del número de colores que se utilice. En los modos gráficos de dos colores, cada *pixel* utiliza un bit. En los modos de cuatro y 16 colores, cada *pixel* requiere dos o cuatro bits para almacenar más colores. Esto significa que una pantalla mapeada, de 16 colores y de 320×200 , requiere 32K bytes completos (cuatro *pixels* por bute).

Modo	Memoria mínima K utilizados	Dirección de comienzo del párrafo	Adapt.
0	2	B800 (la localización varía en el PCjr)	CGA
1	2	B800 (la localización varía en el PCjr)	CGA
2	4	B800 (la localización varía en el PCjr)	CGA
3	4	B800 (la localización varía en el PCjr)	CGA
4	16	B800 (la localización varía en el PCjr)	CGA
5	16	B800 (la localización varía en el PCjr)	CGA
6	16	B800 (la localización varía en el PCjr)	CGA
7	4	B000	MA
8	16	Memoria principal del PCjr (la localización varía)	n/a
9	32	Memoria principal del PCjr (la localización varía)	n/a
10	32	Memoria principal del PCjr (la localización varía)	n/a
13	32	A800	EGA
14	32	A800	EGA
15	64	A000	EGA
16	32	A800	EGA

Figura 4-11. Cantidad de memoria necesaria para cada modo de video y su localización de comienzo en la memoria

Puesto que un visualizador de texto típico ocupa 4000 bytes (sólo 2000 bytes en el modo de 40 columnas), se queda algún espacio libre en la memoria de 16K bytes del adaptador color/gráficos. Se puede utilizar este espacio para guardar más texto, dividiéndolo en páginas.

Visualización de páginas en modo texto

En los modos de texto 0 al 3, la pantalla utiliza siempre menos de 16K. Los modos 0 y 1 utilizan 2K, y los modos 2 y 3 utilizan 4K. Para estos modos, los 16K de memoria disponibles están divididos en múltiples pantallas, llamadas *páginas*. En un momento dado, sólo una página es visualizada. La información se puede escribir en la página visualizada, o en cualquiera de las otras. Utilizando esta técnica, se puede escribir en una página invisible mientras se está visualizando otra página, y a continuación cambiar a la nueva página cuando llegue el momento apropiado. Cambiando pantallas de esta forma, se consigue que parezcan como generadas instantáneamente.

Las páginas de visualización están numeradas del 0 al 7 en los modos 0 y 1, o de 0 a 3 en los modos 2 y 3, con la página 0 comenzando al principio del área de memoria de visualización de 16K. Cada página empieza en un K par. Las direcciones de comienzo de las páginas se muestran en la figura 4-12.

Se selecciona la página cambiando la dirección de comienzo utilizada por el *chip* controlador 6845. Normalmente se hace esto usando el servicio de video 5 de la ROM-BIOS a través de la interrupción 16 (10 hex). (Véase capítulo 9.)

Página	Modos 0 y 1 Desplazamientos de 2K	Modos 2 y 3 Desplazamientos de 4K
0	B800	B800
1	B880	B900
2	B900	BA00
3	B980	BB00
4	BA00	
5	BA80	
6	BB00	
7	BB80	

Figura 4-12. Direcciones de comienzo para las páginas de visualización en los modos 0 al 3.

En cualquiera de estos modos, si todas las páginas no se utilizan, para visualización, la parte libre puede ser utilizada con otros propósitos, aunque no es normal (o aconsejable) hacerlo así. Cualquier otro uso de esta zona de memoria, potencialmente libre, sólo servirá para causar problemas en el futuro.

Visualización de páginas en los modos gráficos

Para el PCjr, el EGA y otro cualquier adaptador de video que utilice la memoria, el concepto de página es equivalente en los modos gráficos y en los modos de texto. Obviamente no hay ninguna razón para no tener páginas de gráficos si la memoria está allí para soportarlas.

El beneficio principal de la utilización de páginas para los gráficos, o el texto, es poder cambiar instantáneamente de una pantalla a otra sin perder el tiempo en construir la información a visualizar en la zona de trabajo. En teoría, las páginas múltiples en el modo gráfico se podrían utilizar para producir efectos de animación, con desplazamientos suaves y nítidos, pero no serían suficientes el número de páginas de visualización para llevar la animación muy lejos. De todas formas, el potencial para la utilización de las páginas en modo gráfico está disponible en los nuevos adaptadores de video.

Visualización de caracteres en los modos de texto y gráficos

Como ya se ha dicho, los modos de texto de los adaptadores monocromo y color/gráficos no almacenan una imagen de caracteres en la memoria de imagen, sino que almacenan sólo los valores ASCII de los caracteres y sus atributos de visualización. El carácter es dibujado en la pantalla por un generador de caracteres, que forma parte del adaptador. El adaptador color/gráficos tiene un generador de caracteres que produce los caracteres en un formato de bloques de 8×8 *pixels*, mientras que el generador de caracteres del adaptador monocromo utiliza un formato de bloques de 9×14 *pixels*. El formato mayor es uno de los factores que hacen que la salida de visualización del adaptador monocromo sea más fácil de leer.

Los caracteres ASCII estándar (CHR\$(1) al CHR\$(127)) representan sólo la mitad de los caracteres ASCII que se pueden utilizar en los modos de texto. También se cuenta con 128 caracteres gráficos disponibles, a través del mismo generador de caracteres (CHR\$(128) al CHR\$(255)). La mitad de ellos puede utilizarse para hacer dibujos simples de línea (semigráficos). En el apéndice C se da una lista

completa de los caracteres ASCII estándar y de los caracteres gráficos proporcionados por IBM.

Los modos gráficos también pueden visualizar caracteres, pero son producidos de forma bastante diferente. Los modos gráficos sólo pueden almacenar información bit a bit, y los caracteres no son una excepción: debe dibujarse un bit cada vez. La gran ventaja de un visualizador mapeado, respecto a los caracteres, es el hecho de que se pueden diseñar los propios caracteres. En el adaptador color/gráficos de IBM original, la tabla para los segundos 128 caracteres está localizada en RAM y, por tanto, puede ser modificada. Modificando la tabla, se puede visualizar un conjunto de caracteres modificado en lugar del conjunto estándar de IBM. Con el PCjr, los 256 caracteres están en RAM; así todos ellos pueden ser modificados.

Caracteres mapeados en modo de texto

En los modos de texto, el mapa de memoria empieza en la esquina superior izquierda de la pantalla, utilizando dos bytes por cada posición de pantalla. Los bytes de memoria para los caracteres siguientes se sitúan a continuación, en el mismo orden en que se leerían —de izquierda a derecha y de arriba abajo.

Los modos 0 y 1 son modos de texto con un formato de pantalla de 40 columnas por 25 filas. Cada fila ocupa $40 \times 2 = 80$ bytes. Una pantalla ocupa sólo 2Kbytes, lo que significa que 16K de memoria pueden acomodar ocho páginas. Si las filas están numeradas de 0 a 24 y las columnas numeradas de 0 a 39, la posición de cualquier carácter en la primera página de visualización está dada por la fórmula de BASIC:

$$\text{DIR.RELATIVA.CARACTER} = (\text{NUMERO.FILA} * 80) + (\text{NUMERO.COLUMNNA} * 2)$$

Puesto que el byte de atributos para cualquier carácter está en la siguiente posición de memoria al carácter ASCII, se puede localizar simplemente añadiendo un 1 a la posición de carácter.

Los modos 2, 3 y 7 son también modos de texto con 80 columnas en cada fila, en vez de 40. La disposición de los bytes es la misma, pero cada fila requiere el doble de bytes, o $80 \times 2 = 160$ bytes. Consecuentemente, el formato de pantalla de 80×25 utiliza 4Kbytes y los 16K de memoria pueden acomodar cuatro páginas. La dirección relativa de cualquier posición de pantalla en la primera página de visualización está dada por la fórmula de BASIC:

$$\text{DIR.RELATIVA.CARACTER} = (\text{NUMERO.FILA} * 160) + (\text{NUMERO.COLUMNNA} * 2)$$

Cuando se utiliza el adaptador color/gráficos, el principio de cada página de visualización de texto comienza tradicionalmente al principio de un Kbyte par. Puesto que cada página de pantalla en los modos texto utiliza sólo 2000 ó 4000 bytes, hay algunos bytes no utilizados al final de cada página. Así, para localizar cualquier posición de una página en modo texto, se utiliza esta fórmula general:

$$\text{POSICION} = (\text{SEGMENTO} * 16) + (\text{NUMERO.PAGINA} * \text{TAMAÑO.PAGINA}) \\ + (\text{NUMERO.FILA} * \text{ANCHURA.COLUMNNA} * 2) + (\text{NUMERO.COLUMNNA} * 2) + \text{CUAL}$$

donde

POSICION es la dirección en 20 bits de la posición buscada.
 SEGMENTO es la dirección de comienzo del segmento de memoria de video (p. ej., hex B000 o B800).
 NUMERO.PAGINA está comprendida en el rango 0-3 ó 0-7.
 TAMAÑO.PAGINA es 2K o 4K.
 NUMERO.FILA es de 0 hasta 24.
 ANCHURA.FILA es 40 u 80.
 NUMERO.COLUMNNA va de 0 a 39 o de 0 a 79.
 CUAL es 0 para carácter y 1 para atributo.

Mapeado de los pixels en los modos gráficos

Cuando se utiliza un modo gráfico, los *pixels* son almacenados como una serie de bits, con una correlación uno a uno entre los bits de la memoria y los *pixels* de la pantalla. Generalmente se utiliza uno de los tres esquemas para mapear la memoria de imagen en los modos gráficos.

El adaptador color/gráficos original organiza la pantalla en 200 líneas, numeradas de 0 al 199. El número de *pixels* de cada línea depende del modo que se utilice. Los modos 4, 5 y 9 son de resolución media, con 320 *pixels* en cada línea. Los modos 6 y 10 son de alta resolución y tienen 640 *pixels* por línea. El modo 8, que fue introducido en el PCjr y no está disponible para su utilización con el adaptador color/gráficos de IBM estándar, es de baja resolución, con 160 *pixels* en cada línea. Las columnas de *pixels* para los modos gráficos de baja, media y alta resolución están numerados de 0 a 159, 319 ó 639, respectivamente.

El almacenamiento de las filas está dividido en «bancos» de líneas, que ocupan posiciones de memoria contiguas. Para los modos 4, 5, 6 y 8 hay dos bancos: el primer banco almacena la memoria para las líneas pares, 0, 2, 4, ..., hasta la 198, y el segundo las

impares. Los modos 9 y 10 tienen cuatro bancos, con las líneas escalonadas similarmente:

banco 1.º	0, 4, 8, 12...196
banco 2.º	1, 5, 9, 13...197
banco 3.º	2, 6, 10, 14...198
banco 4.º	3, 7, 11, 15...199

Estos bancos de líneas son similares a las páginas, en modo de texto, en dos aspectos: las líneas de cada banco van una después de otra sin ningún intervalo libre en la memoria, y cada banco empieza al principio de un Kbyte par, dejando algunos bytes inutilizados al final de cada banco. Sin embargo, a diferencia de las páginas de texto, todos los bancos de líneas son utilizados a la vez por la pantalla de visualización. Cada banco tiene un tamaño de 8K; así los desplazamientos del principio de los bancos son 0, 8K, 16K y 24K.

Como se puede ver en la figura 4-13, la cantidad de memoria utilizada para soportar cada *pixel* varía con el modo. El modo 6 usa un bit, que puede seleccionar dos colores: los modos 4, 5 y 10 usan dos bits, y pueden seleccionar uno entre cuatro colores, y los modos 8 y 9 utilizan cuatro bits, que seleccionan entre 16 colores.

Excepto para el modo 10, que está tratado especialmente, los bits necesarios para cada *pixel* en cada fila se cogen de la memoria en orden consecutivo. Por ejemplo, en el modo 6, que utiliza un bit por *pixel*, los ocho bits del primer byte de la memoria de pantalla controlan los primeros ocho *pixels* de la pantalla.

El primer bit (el más significativo) controla el primer *pixel*, y así sucesivamente. En el modo 4, con dos bits por *pixel*, los ocho bits de cada byte controlan cuatro *pixels*. En el modo 8, con cuatro bits por *pixel*, cada byte controla dos *pixels*. Los tres formatos de bit se muestran en la figura 4-14.

Modo	Columnas	Colores	Bits	Bancos	Memoria (K)
4	320	4	2	2	16
5	320	4	2	2	16
6	640	2	1	2	16
8	160	16	4	2	16
9	320	16	4	4	32
10	640	4	2	4	32

Figura 4-13. Formatos y requerimientos de memoria para los modos gráficos

En el modo 10 el mapeado de bits es diferente (☛ véase la figura 4-15). Como en los modos 4 y 5, el modo 10 requiere dos bits por cada *pixel*, pero, a diferencia de los modos 4 y 5, la información de un *pixel* no está almacenada de forma adyacente en un mismo byte. Está almacenada en los bits correspondientes de dos bytes adyacentes. El bit del primer byte es el bit de mayor orden. Cuando se combina con el bit correspondiente del segundo byte, los dos bits producen un número de color de 0 a 3.

En los modos 4, 5, 6 y 8, cada línea de *pixels* utiliza 80 bytes; en los modos 9 y 10, cada línea utiliza 160 bytes.

CONTROL DE LA PANTALLA

En general, el control de la pantalla de visualización, como la mayoría de las otras operaciones del ordenador, se puede hacer de cuatro maneras:

- Utilizando los servicios del lenguaje de programación (por ejemplo, la instrucción SCREEN del BASIC).
- Utilizando los servicios del DOS (☛ véanse capítulos 16 y 17).
- Utilizando los servicios de video de la ROM-BIOS (☛ véase capítulo 9).
- Por manipulación directa del *hardware*, a través de la memoria, o de los puertos.

Bit							Pixel	Bit							Pixel		
7	6	5	4	3	2	1		0	7	6	5	4	3	2		1	0
<i>Modo 6</i>									<i>Modos 4 y 5</i>								
X	.	.	.	.	.	.	.	1	X X	.	.	.	.	.	.	.	1
.	X	.	.	.	.	.	.	2	.	.	X X	.	.	.	.	.	2
.	.	X	.	.	.	.	.	3	.	.	.	.	X X	.	.	.	3
.	.	.	X	.	.	.	.	4	.	.	.	.	.	X X	.	.	4
.	.	.	.	X	.	.	.	5	<i>Modos 8, 9, 13 y 14</i>								
.	.	.	.	.	X	.	.	6	X X X X	.	.	.	.	.	.	.	1
.	.	.	.	.	.	X	.	7	.	.	.	.	X X X X	.	.	.	2
.	.	.	.	.	.	.	X	8									

Figura 4-14. Mapa de bits de los primeros pixels de los tres formatos gráficos

1.º Byte								2.º Byte								Pixel
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	
X	.	.	.	.	.	.	.	X	.	.	.	.	.	.	.	1
.	X	.	.	.	.	.	.	.	X	.	.	.	.	.	.	2
.	.	X	.	.	.	.	.	.	.	X	.	.	.	.	.	3
.	.	.	X	.	.	.	.	.	.	.	X	.	.	.	.	4
.	.	.	.	X	.	.	.	.	.	.	.	X	.	.	.	5
.	.	.	.	.	X	.	.	.	.	.	.	.	X	.	.	6
.	.	.	.	.	.	X	.	.	.	.	.	.	.	X	.	7
.	.	.	.	.	.	.	X	.	.	.	.	.	.	.	X	8

Figura 4-15. Mapa de bits de los primeros pixels del formato del modo 10

Los servicios de video disponibles a través de los lenguajes de programación, del DOS, y de la ROM-BIOS sitúan automáticamente los datos de salida por pantalla en la memoria de imagen; cada tipo de servicio ofrece varios niveles de control. Los 16 servicios de la ROM-BIOS son particularmente potentes, proporcionando casi todos los servicios necesarios para generar la imagen en la pantalla, el control del cursor y la manipulación de la información de la pantalla. (Los 16 servicios están completamente descritos en el capítulo 9.) Para controlar al máximo la pantalla de video también es posible ignorar los servicios del *software* y situar los datos directamente en la memoria de video cuando se crea que se tiene una buena razón para ello.

Antes de optar por la salida directa, debe saber que interfiere con los sistemas de partición de pantalla en ventanas y los más avanzados entornos operativos multitarea. Muchos programas importantes de la familia PC manejan la pantalla directamente (tantos, de hecho, que se ha convertido en un estándar y forma aceptada de creación de imágenes. Así, incluso aunque a largo plazo probablemente no es aconsejable generar directamente la imagen de video, parece que todos lo hacen).

Básicamente, no se pueden mezclar programas que escriban directamente en la memoria de pantalla y sistemas de ventanas, porque los dos programas se pelearían por el control de la misma memoria, y cada uno echaría a perder los datos del otro. Pero puesto que tantos programas manejan directamente la memoria de video, el sistema de ventanas multitarea propio de IBM, Topview, es muy potente acomodando programas que escriben directamente en la memoria de video. Un sistema como el Topview puede hacer esta acomodación guardando simplemente una copia separada de la

Acerca del cursor

El parpadeo del cursor es una característica de los modos de texto que se utiliza para indicar la localización activa en la pantalla. El cursor es un grupo de líneas que llena la anchura total de la casilla del carácter.

El tamaño de la casilla del carácter varía con el adaptador de video; el adaptador monocromo utiliza un formato de 9 *pixels* de ancho por 14 líneas de alto, y el adaptador color/gráficos utiliza un formato de 8 *pixels* por 8 líneas. (Las líneas extra en el modo monocromo permiten el dibujo del carácter con más detalle, como se verá en el apéndice C.)

El formato del cursor, por defecto, ocupa todas las líneas, pero se puede cambiar para dibujar sólo algunas líneas. Por ejemplo, el cursor se puede ajustar para que empiece y acabe en cualquier conjunto de líneas o incluso utilizar parte de las líneas de abajo y parte de las de arriba. Esto nos permite hacer un cursor entero localizado en cualquier sitio de la casilla del carácter, o un cursor de dos partes localizado en la parte superior y la inferior de la casilla del carácter. (☛ Véase, página 194, la casilla del carácter y la relación de las líneas de exploración con los caracteres.)

Puesto que el parpadeo del cursor utilizado en los modos de texto es una característica creada por *hardware*, el *software* sólo tiene un control limitado sobre él. Se puede cambiar su formato y localización en la pantalla de varias formas. Para leer o cambiar la localización del cursor, se pueden utilizar los servicios de la ROM-BIOS (☛ véase capítulo 11) o se puede leer o escribir directamente en la memoria (☛ véase la discusión sobre la localización 450 hex en la página 63). Asimismo, se puede leer y cambiar el formato del cursor utilizando los servicios de la ROM-BIOS, o se puede leer el formato directamente inspeccionando la memoria (☛ véase la discusión sobre la localización 460 en la página 64). La mayoría de los lenguajes de programación ofrece también estos servicios.

Si desea ignorar el parpadeo de cursor controlado por *hardware* (y muchos de nosotros lo hacemos), es posible utilizar el atributo de visualización de video inverso (70 hex), en la posición que se sitúe el cursor real. Esto producirá un bloque de cursor que no parpadea. Otra forma de hacer esto es utilizar los caracteres ASCII de bloque, el CHR\$(219) o el CHR\$(254).

Hasta ahora se ha estado hablando sobre el cursor en el modo de texto. En los modos gráficos no se visualiza el cursor, pero se registra una localización de cursor lógica, que indica la posición activa de pantalla. Como en los modos de texto, para encontrar la posición del cursor, se pueden utilizar los servicios de la ROM-BIOS, o leer directamente el byte de posición (450 hex).

Para crear un cursor en los modos gráficos, algunos programas, incluyendo el BASIC, simulan el bloque del cursor utilizando un color de fondo distinto al de la localización del cursor, o utilizando los caracteres ASCII de bloque.

memoria de pantalla de cada uno de los programas; cuando el programa está funcionando, la copia se pasa a la memoria del visualizador, y cuando el programa se detiene, se vuelve a hacer una copia, actualizada, de la pantalla. Esta técnica permite al Topview hacer funcionar programas que trabajan directamente con la memoria de video, pero a un costo enorme: primero, aumenta los cálculos y la memoria utilizada; segundo, el programa no puede funcionar en segundo plano simultáneamente con otros programas, y tercero, la información del visualizador no puede ser manejada en ventanas (esto es, no puede moverse o ajustarse su tamaño).

Los programadores se enfrentan aquí con un conflicto: la salida directa a la pantalla tiene el beneficio de la velocidad y la potencia, mientras que la utilización de los servicios de la BIOS, o del nivel superior, para la salida de pantalla tienen la ventaja de la mayor flexibilidad para su adaptación a los sistemas de ventanas, nuevos adaptadores de visualización, etc. La solución que adopté para mis propios programas es utilizar ambas técnicas, activando una o la otra cuando era necesario.

Control directo del hardware

Gran parte de la información que se ha proporcionado en este capítulo, particularmente la información del mapeado interno de la memoria de video, está pensada para ayudar a escribir directamente la información de video en la memoria de imagen. Pero debe recordarse que hay un riesgo en cualquier tipo de programación directa, y se encontrará que es más seguro y fácil utilizar los medios disponibles de más alto nivel. La acción en bajo nivel, particularmente la manipulación directa, puede ser peligrosa. Hay sólo unos pocos ejemplos donde el control directo es seguro y fiable. Siempre que sea posible, señalaré estas circunstancias.

Puertos de E/S del adaptador monocromo

El adaptador monocromo usa cuatro puertos de E/S: los puertos de estado y control del CRT, y los registros del controlador de CRT 6845.

Puerto de control del CRT (3B8 hex). Se pueden modificar tres de los 8 bits de este puerto: alta resolución, video y bits de parpadeo. El bit de alta resolución siempre debe estar a uno para utilizar el adaptador monocromo. La modificación de los bits de video y parpadeo cambia la generación de video y el parpadeo del carácter. Enviando el valor 29 hex a este puerto, se pondrán los tres bits en su estado normal. (● Véase la figura 4-16.)

Bit								Uso
7	6	5	4	3	2	1	0	
.	.	.	.	.	.	.	X	Modo de alta resolución: debe estar puesto a 1
.	.	.	.	.	X	X	.	No utilizado
.	.	.	.	X	.	.	.	0 = señal de video deshabilitada; 1 = señal de video habilitada
.	.	.	X	.	.	.	.	No utilizado
.	.	X	.	.	.	.	.	0 = sin parpadeo; 1 = con parpadeo
X	X	.	.	.	.	.	.	No utilizado

Figura 4-16. Codificación de la puerta de control del CRT

Puerto de estado del CRT (3BA hex). Este puerto almacena el estado de la señal de sincronismo vertical en el bit 0 y los bits enviados a la pantalla en el bit 3. Aunque es posible leer estos dos bits, ninguno es particularmente útil. Los otros bits no se utilizan.

Controlador del CRT 6845 (3B4 y 3B5 hex). En el 6845 hay 18 registros internos programables. Especifican datos, como la temporización de las señales de sincronismo horizontal y vertical, el número de líneas de visualización y el número de caracteres por línea. Sólo cuatro registros pueden ser utilizados con seguridad: los registros 0A, 0B, 0E y 0F hex. Los registros 0A y 0B determinan las líneas en que el cursor empieza y termina, y los registros 0E y 0F, la posición del cursor en pantalla, con un valor en un rango de 0 a 1999. Ambas funciones están también disponibles a través de la interrupción 16 (10 hex) en los servicios de la ROM-BIOS. No deben modificarse los otros valores; puede ser peligroso. (Por ejemplo, se puede dañar el visualizador monocromo si se programa incorrectamente el controlador de video 6845.) Si se quiere conocer más datos sobre ello, puede leerse el «Manual de Referencia Técnica del IBM PC».

Puertos de E/S del adaptador color/gráficos

Para acomodar las funciones gráficas, el adaptador color/gráficos tiene más puertos de E/S que el adaptador monocromo. Se señalarán los aspectos más importantes de cada una de los siete puertos.

Registro de selección de modo (3D8 hex). Este byte se modifica para cambiar de un modo de visualización a otro (● véase la figura 4-17.)

Registro de selección de color (3D9 hex). Modificando este byte, se cambian los colores del margen de la pantalla en los modos de

texto y los colores del fondo y primer plano en los modos gráficos. (☛ véase la figura 4-18.)

Registro de estado (3DA hex). Este registro almacena la información útil para aquellos que prefieran una actualización de pantalla sin centelleo. Cuando el bit 0 está a 1, se puede acceder a la memoria de pantalla sin perturbar la visualización. Cuando el bit 3, sincronismo vertical, está a 1, el barrido está en retorno vertical y se puede actualizar la pantalla. Este registro tiene también dos señales de estado del lápiz óptico (☛ véase la figura 4-19.)

Bit								Uso
7	6	5	4	3	2	1	0	
.	.	.	.	.	.	.	X	0 = selecciona el modo de texto de 40 × 25; 1 = selecciona el modo de texto de 80 × 25
.	.	.	.	.	.	X	.	0 = selecciona el modo de texto; 1 = selecciona el modo gráfico de 320 × 200
.	.	.	.	X	.	.	.	0 = selecciona el modo de color; 1 = selecciona el modo de b/n
.	.	.	X	.	.	.	.	0 = señal de video deshabilitada; 1 = señal de video habilitada
.	.	X	.	.	.	.	.	1 = gráficos b/n de 640 × 200
.	X	.	.	.	.	.	.	0 = sin función de parpadeo; 1 = con función de parpadeo
X	X	.	.	.	.	.	.	No utilizado

Figura 4-17. Codificación de registro de selección de modo

Bit								Uso
7	6	5	4	3	2	1	0	
.	.	.	.	.	.	.	X	Selecciona el primer plano, fondo o margen azul
.	.	.	.	.	.	X	.	Selecciona el primer plano, fondo o margen verde
.	.	.	.	X	.	.	.	Selecciona el primer plano, fondo o margen rojo
.	.	X	.	.	.	.	.	Selecciona la intensidad
.	X	.	.	.	.	.	.	Selecciona paleta intensificada, alternada
.	X	.	.	.	.	.	.	0 = paleta 0; 1 = paleta 1
X	X	.	.	.	.	.	.	No utilizado

Figura 4-18. Codificación del registro de selección de color

Puertos de lápiz óptico (3DB y 3DC hex). Escribiendo en cada uno de estos puertos se activa o desactiva un interruptor basculante conectado a la entrada de lápiz óptico del 6845.

Bit								Uso
7	6	5	4	3	2	1	0	
.	.	.	.	.	.	.	X	1 = el acceso a memoria puede ocurrir sin interferencia con el visualizador
.	.	.	.	.	.	X	.	1 = disparo del lápiz óptico
.	.	.	.	.	X	.	.	0 = lápiz óptico activo; 1 = lápiz óptico inactivo
.	.	.	.	X	.	.	.	1 = el barrido está en retorno vertical
X	X	X	X	.	.	.	.	no utilizados

Figura 4-19. Codificación del registro de estado

El controlador de video 6845 (3D0 y 3D1 hex). El controlador funciona de la misma forma con el adaptador color/gráficos que con el adaptador monocromo.

El control del generador del video es complicado, y se ha hecho mucho más complicado por la constante ampliación de la lista de características de visualización del PC. Antes de incorporar a los programas cualquier cosa que se desee, es importante, y una buena idea, verificar los conocimientos que se poseen sobre el controlador de video experimentando con él.

CONSIDERACIONES DE COMPATIBILIDAD

Para que nuestros programas sean compatibles con todos los modelos de ordenadores personales de IBM, es necesario tener en cuenta varias cosas. Primero: un modelo PC estándar no puede crear gráficos si sólo está equipado con el adaptador monocromo. Además, los modos gráficos del 8 al 10 no pueden ser utilizados con la versión original, o mejorada, del adaptador color/gráficos; son parte de las mejoras de color del PCjr. Asimismo, los modos 13 al 16 pertenecen al EGA. Estas restricciones se aplican también al remapeado de las paletas de color; ya que esta capacidad está ligada al PCjr y al EGA. También es importante recordar que el adaptador monocromo de los modelos PC y XT trata los atributos de color del modo de texto de una forma especial (■ como se vio en la página 90).

Es una buena idea adaptar para un programa su uso del color, o la elección entre los modos de texto y gráficos, para acomodar al monitor monocromo de IBM, o a un monitor monocromo compuesto, que normalmente no reproduce bien el color. Debe tenerse en cuenta que los monitores monocromos compuestos se pueden utilizar a menudo con los PC, especialmente cuando se trabaja con texto.

Hay algunos PC equipados de esta forma, y es prudente que nuestros programas los acomoden.

Para soportar estos sistemas, nuestros programas deberían adivinar el modo de video y actuar acordeamente. Para los programas que ya está utilizando un interfaz de lenguaje ensamblador con el BIOS, la forma preferible para hacer esto es utilizar el servicio de video 15 de la BIOS (● véase capítulo 9). Para otros programas, el servicio 15 es un escollo. El problema puede ser evitado leyendo la posición de memoria 0000:0449, donde está almacenado el modo de video (● véase página 62). Es posible leer esta posición en BASIC de esta forma:

```
DEF SEG = 0  
MOD0.VIDEO = PEEK (&H449)
```

El modo de video 7 identifica el uso del monitor monocromo de IBM. No hay una forma automática para identificar el uso del visualizador monocromo compuesto; sin embargo, si un usuario ha utilizado el comando MODE del DOS para suprimir el color, nuestros programas pueden detectarlo en un modo de video 0 ó 2.

Cuando se desea considerar las compatibilidades de trabajo de nuestros programas con los ordenadores personales de IBM y los diferentes tipos de pantallas de visualización, se puede disponer de varios criterios de compatibilidad a considerar. Estos criterios no son completamente consistentes unos con otros, reflejando la inconsistencia de diseño de IBM y la variedad de formatos de visualización que pueden ser utilizados. Aun así, hay varias reglas de compatibilidad, que perfilaremos aquí.

Primero, la salida sólo de texto incrementa la compatibilidad. Hay algunos PC equipados con adaptadores monocromos que no pueden mostrar salida gráfica. Si está contrapesando una decisión de textos frente a gráficos en el diseño de un programa, hay dos factores a considerar: uno, a favor de la utilización del visualizador sólo de texto, y otro, en contra. Por una parte, como han demostrado claramente algunos programas, es posible crear dibujos muy efectivos utilizando los caracteres de texto estándar de IBM (● véase apéndice C para obtener más información sobre la utilización efectiva de los caracteres de texto para dibujo.) Por otra parte, cada vez es más común incluir capacidades gráficas en los ordenadores. Por ejemplo, el PCjr y el PC portátil de IBM, así como el modelo Compaq, cuentan con capacidades gráficas incorporadas. Así, en el futuro, la salida, exclusivamente de texto, perderá probablemente su importancia y se podrán diseñar gráficos directamente en nuestros programas, sin preocuparse de la compatibilidad.

Segundo, cuanto menos dependan nuestros programas del color, mayor es el rango de ordenadores con los que será compatible. Esto no significa que sea necesario evitar los colores para conseguir la compatibilidad, significa, simplemente, que, para un máximo de compatibilidad, nuestros programas deberían utilizar el color como una mejora y no como un ingrediente esencial. Si nuestros programas pueden funcionar sin color, serán compatibles con los ordenadores que utilizan visualizadores monocromos, incluyendo PC con adaptadores monocromos, así como el Compaq y los ordenadores PC portátiles de IBM con sus visualizadores monocromos dispuestos internamente.

Tercero, si fuese posible, nuestros programas tendrían que poder trabajar en 40 u 80 columnas de anchura cuando estuvieran en modo de texto para poder acomodarse al uso de TV o monitores. La anchura de 40 columnas incrementa la legibilidad, pero reduce la cantidad de información que puede ser visualizada. Cuando no es mucha la información a visualizar, debe considerar el uso obligatorio de la anchura de 40 columnas, pero, en conjunto, lo mejor es que nuestros programas se adapten a la anchura existente, o preguntar al usuario del programa qué anchura mostrar. Recuerde que la anchura de 40 columnas no funciona en el PC con adaptador monocromo.

Considerando estas reglas a la luz de las características particulares de los programas propios, debe sopesarse la ventaja de la amplia compatibilidad, contra la conveniencia de la simplicidad de la escritura de programas para un estrecho rango de visualizadores. Mi propia experiencia y juicio me dicen que los programadores se equivocan muy a menudo, optando por un estrecho rango de visualizadores, y por tanto, reduciendo seriamente la variedad de ordenadores sobre los que pueden ser utilizados sus programas. Queda advertido.

Los discos, conceptos básicos



Estructura física del disco	112
Formatos de disco del DOS	114
Formatos estándar del DOS	114
Formatos de cuádruple densidad	115
Formato del disco duro	117
Estructura lógica del disco	118
Organización de los discos	119
Distribución del espacio del <i>diskette</i>	120
Distribución del espacio del disco duro	122
La estructura lógica en detalle	123
El registro de arranque (<i>boot</i>)	123
El directorio	126
El espacio de datos	133
Tabla de localización de ficheros (FAT)	135
Comentarios	139

La mayoría de sistemas informáticos tiene alguna forma de almacenar la información permanentemente, bien en cintas magnéticas, bien en discos flexibles, o en discos duros. Estos dispositivos de almacenamiento se presentan en varios tamaños y capacidades, pero operan básicamente de la misma forma: codifican magnéticamente la información en sus superficies según modelos determinados por el propio dispositivo, o por el *software* que controla el dispositivo.

Cuando se introdujo la familia PC, en 1981, utilizaba básicamente un tipo de dispositivo de almacenamiento: el disco flexible de 5 ¼ pulgadas, doble densidad, simple cara, y sectorizado por *soft*, y con sólo 160Kbytes de capacidad. Desde entonces, IBM ha incrementado la capacidad de almacenamiento del *diskette* y para alguno de sus sistemas PC ha añadido discos duros de 10 y 20 megabytes. En el futuro podemos esperar que IBM y otros fabricantes continúen los avances en la tecnología del disco, incluyendo definitivamente discos duros de capacidad mayor y *minidiskettes* de 3 ½ pulgadas.

Aunque el tipo de dispositivo de almacenamiento es importante, lo que concierne a los programadores es la forma en que se dispone y maneja la información almacenada. En este capítulo se presta mayor atención a la forma en que se organiza y almacena la información en los discos flexibles, puesto que son los medios de almacenamiento más comunes en la familia PC. Aunque en principio se tratará sobre los discos flexibles, lo que realmente se estará haciendo será un bosquejo que representa a todos los dispositivos de almacenamiento que se engloban bajo la palabra disco. La información proporcionada en este capítulo es válida tanto para los discos RAM —esto es, la simulación de discos en memoria— como para los *diskettes* convencionales, discos duros, cartuchos y *minidiskettes*.

ESTRUCTURA FISICA DEL DISCO

Las unidades de disco y el sistema operativo del ordenador determinan la *capacidad* de los discos utilizados, pero la *estructura* del disco es esencialmente la misma, independientemente de la organización. Los datos están siempre registrados en la superficie del disco en una serie de círculos concéntricos, llamados *pistas*. Cada pista se divide posteriormente en segmentos, llamados *sectores*. (● véase la figura 5-1). La cantidad de datos que puede ser almacenada en cada cara de un disco depende del número de pistas (su densidad) y el tamaño de sus sectores. La densidad del disco puede variar considerablemente según la unidad utilizada: las unidades de doble densidad estándar pueden registrar 40 pistas de datos,

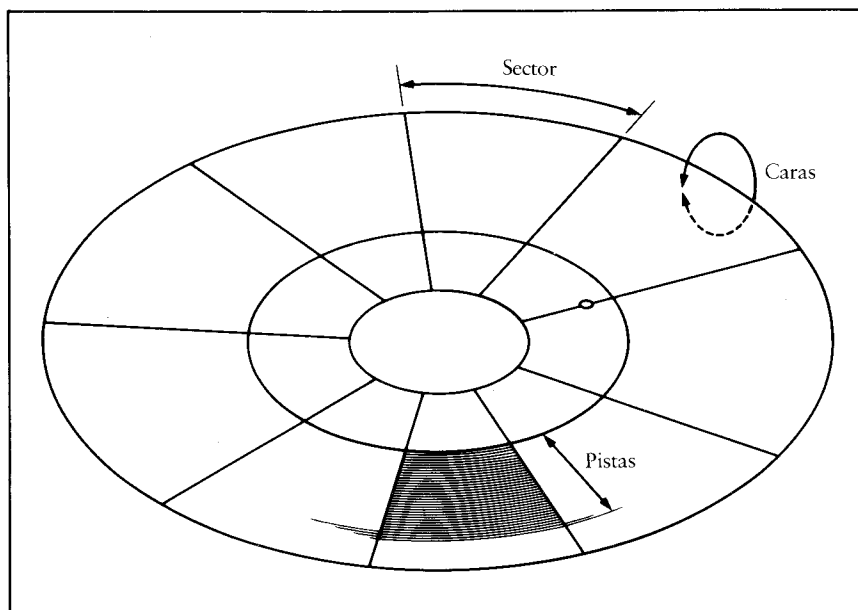


Figura 5-1. Estructura física del disco

mientras que las nuevas unidades de cuádruple densidad pueden registrar 80 pistas. Para los *diskettes* de 5 $\frac{1}{4}$ pulgadas estándar del PC, la localización de cada pista y el número de caras utilizables están impuestos por las características *hardware* de los discos y de las unidades de discos, y, por tanto, son fijas e inmodificables. Sin embargo, la localización, tamaño y el número de sectores por pista están bajo control del *software*. Es, por esto, por lo que los *diskettes* del PC son conocidos como sectorizados por *soft*. Las características de los sectores del *diskette* (su tamaño y el número por pista) se establecen al formatear cada pista. El formateo del disco se puede hacer mediante el sistema operativo, o con el servicio de formateo de la ROM-BIOS. La mayor parte de este capítulo se dedicará a presentar los formatos del DOS. Sin embargo, se pueden crear fácilmente nuevos formatos y utilizarlos como parte de un esquema de protección de copia, utilizando los servicios del *diskette* de la ROM-BIOS (véase el servicio 5 en la página 215).

Los *diskettes* de 5 $\frac{1}{4}$ pulgadas utilizados por el BIOS del PC estándar pueden tener sectores con tamaños de 128, 256, 512 ó 1024 bytes. El DOS, de las versiones 1.00 a la 3.2, ha utilizado habitualmente sectores de 512 bytes, y es muy posible que se sigan utilizando. Sin embargo, cualquier programa que dependa, o haga uso, de los sectores del DOS de 512 bytes debería permitir futuros cambios en el tamaño del sector, especialmente sectores de mayor tamaño.

Un *diskette*, por supuesto, tiene dos caras, y los sectores pueden estar situados en una o ambas caras. Los sistemas de disco duro pueden constar de uno o más discos, así que pueden tener más de dos caras. **XT** Por ejemplo, el disco duro de 10 megabytes introducido con el XT tiene dos discos y utiliza las cuatro caras disponibles.

FORMATOS DE DISCO DEL DOS

En las primeras versiones del DOS utilizadas por IBM sólo se podía utilizar un número limitado de formatos, aunque las unidades de disco pudieran leer o escribir en varios. Comenzando con el DOS 2.00 y continuando con todas las subsiguientes versiones, el DOS ha sido equipado tan sólo con unas pocas opciones de formateado estándar, pero permite que sea integrado cualquier formato físico de disco. (El formato lógico, como se verá más adelante, está restringido por el DOS a un conjunto estándar cerrado.) Esta integración es posible porque el DOS proporciona las herramientas necesarias para escribir un controlador de dispositivo (*device driver*) instalable (una rutina en lenguaje máquina que puede configurar una unidad de disco para leer o escribir en diferentes formatos, o permite conectar una unidad de disco no IBM al sistema). (● Véase apéndice A para más información sobre controladores de dispositivos instalables.)

Puesto que hay muchos formatos de disco potenciales, posiblemente no podemos considerar todos ellos. Examinaremos siete formatos de disco comunes, incluyendo cuatro formatos de *diskette* de 5 ¼ pulgadas normales, uno de *diskette* de 5 ¼ pulgadas especial, uno de mini-*diskette* de 3 ½ pulgadas, y uno de disco duro. Juntos, estos siete formatos comunes constituirán una guía suficiente como para trabajar con cualquier tipo de disco.

Formatos estándar del DOS

Se verán en primer los cuatro formatos más comunes del PC, los utilizados por IBM como formatos estándar para los *diskettes* de 5 ¼ pulgadas. Los cuatro formatos están derivados del número de caras y del número de sectores en cada pista: simple o doble cara, y ocho o nueve sectores (● véase la figura 5-2).

La razón por la que hay cuatro formatos estándar es bastante simple: IBM ha de tener la seguridad de que todas las versiones del DOS respetan todos los anteriores modelos. Los primeros PC vinieron equipados con unidades de disco de simple cara. Después, IBM introdujo unidades de doble cara y dejó de utilizar las de simple cara. Ahora, aunque relativamente pocos PC tienen unidades de

Nuestra notación	Caras	Sectores	Pistas	Tamaño nominal
S-8	1	8	40	160K
D-8	2	8	40	320K
S-9	1	9	40	180K
D-9	2	9	40	360K

Figura 5-2. Formatos estándar del DOS

simple cara, los formatos de simple cara son respetados por todas las versiones del DOS, para compatibilizarlos con las máquinas anteriores.

En las primeras versiones del DOS, cada pista tan sólo constaba de 8 sectores de 512 bytes cada uno, aunque se podían introducir sin ninguna dificultad más de diez sectores de ese tamaño. Después se aceptaron como seguros y fiables nueve sectores de 512 bytes, tomando como estándar este nuevo formato. Una vez más, los demás formatos fueron preservados para mantener la compatibilidad.

Los adelantos en el campo de los formatos están ligados a la historia y al desarrollo del DOS. La versión 1.00 original del DOS soportaba sólo lo que yo llamo formato S-8. La siguiente versión, 1.10 añadía el D-8. La versión 2.00 añadía los dos formatos de nueve sectores, S-9 y D-9. Con el DOS 2.10 no se añadieron nuevos formatos, mientras que el DOS 3.0 incorporaba el formato de cuádruple densidad, que se verá brevemente.

Aunque hay varios formatos, sólo dos son utilizados habitualmente: el S-8 y el D-9. El S-8 constituye el denominador común más bajo, por lo que ha sido utilizado tradicionalmente en programas comerciales, puesto que la utilización del S-8 garantiza que un *diskette* puede ser leído por cualquier versión del DOS. Sin embargo, esta práctica está cayendo en desuso, especialmente en compañías que venden grandes programas que necesitan más espacio de disco. El formato D-9 es el de mayor capacidad que puede utilizar la mayoría de las unidades de 5 $\frac{1}{4}$ pulgadas, por lo que la mayor parte de la gente lo utiliza para sus *diskettes* de trabajo. Los otros formatos, D-8 y S-9, no son tan comunes, pero son utilizados ocasionalmente.

Formatos de cuádruple densidad

Adviértase que existe un factor constante en los cuatro formatos estándar: es el número de pistas. Todos ellos tienen 40. La razón es

que las unidades de *diskette* de 5 $\frac{1}{4}$ pulgadas que se usan más a menudo con la familia PC están diseñadas para leer y escribir 40 pistas de datos. Pero algunas unidades de *diskette* de 5 $\frac{1}{4}$ pulgadas, y otras de 3 $\frac{1}{2}$ pulgadas, pueden registrar 80 pistas en el mismo espacio. Este tipo de unidad y los *diskettes* que utiliza se designan, a menudo, con el nombre cuádruple densidad. De los posibles formatos de cuádruple densidad, se van a estudiar los dos más comunes de la familia PC. Se le designará con los nombres QD-9 y QD-15. (● Véase figura 5-3.)

El formato QD-9 es muy parecido al D-9, excepto en que tiene 80 pistas de datos en lugar de 40. Como el formato D-9, el formato QD-9 tiene dos caras, con nueve sectores por pista en cada cara.

Aunque IBM ha evitado utilizar el formato QD-9, ha estado disponible en otros equipos, tales como el ordenador DG-1 de Data General, un primo de la familia PC. El DG-1 utiliza *minidiskettes* de 3 $\frac{1}{2}$ pulgadas en lugar de los *diskettes* estándar de 5 $\frac{1}{4}$ pulgadas, pero la estructura lógica de sus formatos es la misma. Aunque las unidades de *minidiskettes* sean de cuádruple densidad, sus discos pueden formatearse no sólo en el formato QD-9, sino también en los otros cuatro formatos, S-8, S-9, D-8 y D-9. Las unidades de cuádruple densidad pueden también estar conectadas a PC normales como equipamiento no estándar, utilizando un controlador de dispositivos de DOS. (● Véase apéndice A para más información sobre controladores de dispositivos.) Mucha gente cree que este formato llegará a tener un gran uso, lo cual podría ser muy interesante.

El formato QD-15, de alta capacidad utilizado por el AT, sigue la misma estructura básica expuesta: 80 pistas por cara, y sectores estándar de 512 bytes. La característica especial de QD-15 es que cada cara de cada una de las pistas almacena 15 sectores en lugar de ocho o nueve. La introducción de tantos sectores en una pista es sólo posible porque el AT utiliza *diskettes* especiales de alta capacidad, que tienen un recubrimiento magnético diferente a los *diskettes* ordinarios. Sólo estos *diskettes* especiales —que parecen iguales a los

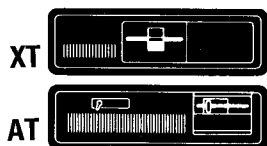


Nuestra notación	Caras	Sectores	Pistas	Tamaño nominal
QD-9	2	9	80	720K
QD-15	2	15	80	1200K

Figura 5-3. Formatos de cuádruple densidad

diskettes normales de 5 ¼ pulgadas— y las unidades especiales de alta capacidad pueden utilizar el formato QD-15.

Formato del disco duro



Los sistemas de disco duro de alta capacidad, tales como el disco duro de 10 megabytes de XT, o el disco duro de 20 megabytes del AT, presentan algunos problemas especiales.

Hay dos aspectos en cualquier disco: su formato físico y su formato lógico. El formato físico de un disco determina el tamaño del sector en bytes, el número de pistas (cilindros en los discos duros) y el número de caras. El formato lógico determina la forma en que será organizada la información en el disco y donde serán situados los diferentes tipos de información.

Al formatear un disco flexible con el DOS, o algún otro sistema operativo, se establecen los formatos físico y lógico del *diskette* o ignoramos cualquier distinción entre ellos. Por el contrario, el formato físico de un disco duro ya está establecido cuando se adquiere —está impuesto por el fabricante—. (● Véase figura 5-4.) Lo que no está presente en el formato físico impuesto en la factoría es la estructura lógica del disco; se debe establecer antes de que el sistema operativo pueda utilizarlo. Esto se hace en dos fases: primero, se divide el disco duro en particiones lógicas que alberguen los datos y programas de cada sistema operativo que se utilice (se pueden utilizar varios sistemas operativos con el sistema de disco duro; ● véase la página 124). Después, hay que definir la organización de las particiones, de forma que cada sistema operativo pueda localizar la información en su partición. Este proceso de «organización del disco» es lo que usualmente se llama formateado.

ESTRUCTURA LOGICA DEL DISCO

Sea cual sea el disco que se utilice, los discos del DOS están todos formateados lógicamente de la misma forma: las caras, las

Nuestra notación	Caras	Sectores	Cilin.	Tamaño nominal (megabytes)
XT	4	17	306	10
AT	4	17	615	20

Figura 5-4. Formatos físicos de los discos duros del XT y el AT

pistas y los sectores están identificados numéricamente utilizando la misma notación, y ciertos sectores están siempre reservados para programas e índices especiales que utilizan el DOS para gestionar las operaciones del disco. Antes de ver cómo organiza el DOS el espacio del disco, es necesario señalar, brevemente, la notación convencional utilizada por el DOS y el BIOS para localizar la información.

Como se ha visto en epígrafes anteriores, los formatos de *diskette* de 5 $\frac{1}{4}$ tienen 40 pistas, numeradas del 0 (la pista exterior) al 39 (la pista interior, más cercana al centro). Otros formatos de disco pueden tener más pistas. Por ejemplo, las pistas de los *diskettes* de cuádruple densidad están numeradas del 0 al 79; **XT** los cilindros del disco duro del XT están numerados del 0 al 305, **AT** y los cilindros del disco duro del AT están numerados del 0 al 614.

En un *diskette* de doble cara, las dos caras están numeradas como 0 y 1 (las dos cabezas de una unidad de disco de cara doble están también numeradas como 0 y 1). La única cara de un *diskette* de simple cara se designa con el número 0. **XT** El disco duro del XT tiene cuatro caras (y cuatro cabezas), numeradas del 0 al 3.

Los sectores en los discos flexibles están numerados del 1 al 8 o al 9. **XT** En el disco duro del XT están numerados del 1 al 17. Adviértase que los números de sector comienzan con el 1, mientras que los números de cara comienzan con el 0.

El BIOS localiza los sectores en un disco por un sistema de coordenadas en tres dimensiones, compuesto por un número de pista (también conocido como número de cilindro), un número de cara (también llamado número de cabeza) y un número de sector. El DOS, por otra parte, localiza la información por el número de sector y numera los sectores secuencialmente de fuera a dentro. (● Véase figura 5-5.) La secuencia empieza con el primer sector del disco: sector 1 de la cara 0 y pista 0, seguido por los sectores que quedan en la misma cara y pista. Para los *diskettes* de doble cara, el noveno sector de la cara 0 y pista 0 está seguido por el primer sector de la cara 1 y pista 0. El orden prosigue a través de todos los sectores de una cara y pista, después a través de la siguiente cara, en la misma pista (así todas las caras en una pista se encuentran antes de la próxima localización de pista).

Se puede señalar un sector determinado bien por sus coordenadas en tres dimensiones, bien por su orden secuencial. Todas las operaciones de la ROM-BIOS utilizan las coordenadas en tres dimensiones para localizar un sector. Todas las operaciones del DOS y herramientas tales como el DEBUG utilizan la notación secuencial del DOS. (● Véase el capítulo 14 cómo invertir la notación del DOS a la notación de la ROM-BIOS, y viceversa.

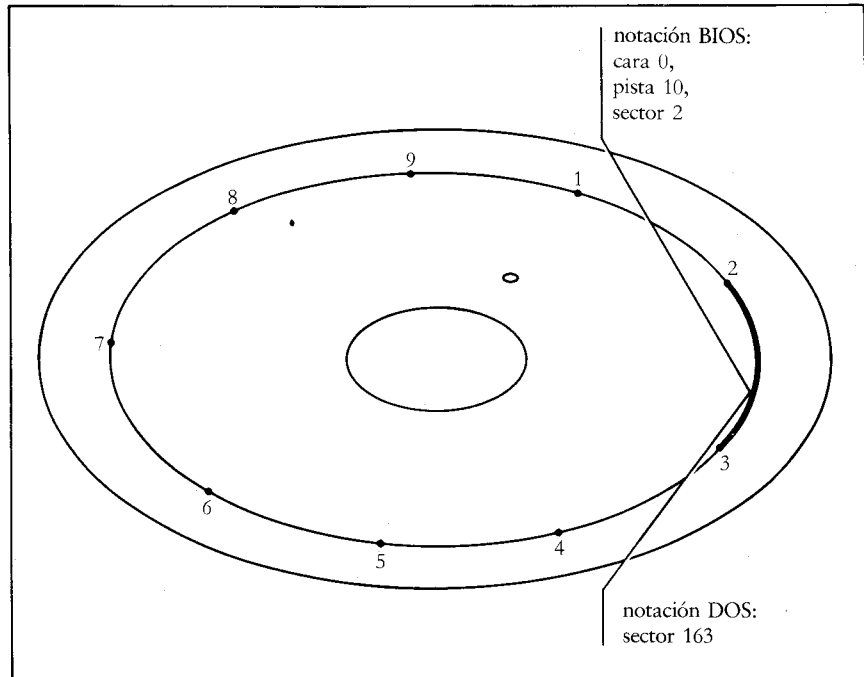


Figura 5-5. Notación de los sectores del DOS y de la ROM-BIOS

ORGANIZACION DE LOS DISCOS

Como ya se ha visto, cuando se dan instrucciones al DOS para formatear un *diskette*, divide cada una de las 40 pistas en ocho o nueve sectores de 512 bytes. En términos de capacidad de almacenamiento bruta, asciende a 368 640 bytes de espacio de datos en nuestros *diskettes* estándar D-9. Pero no todo el espacio puede ser utilizado para almacenar datos; una cierta cantidad se utiliza para almacenar índices e información de control del sistema que utiliza el DOS para encontrar la localización y relación entre sectores individuales. Además de dividir el disco en sectores, el DOS realiza otras varias operaciones cuando formatea un disco.

Distribución del espacio del *diskette*

El proceso de formateado divide los sectores de un disco en cuatro secciones para cuatro usos diferentes. Las secciones, en el orden en

que están almacenadas, son el registro de puesta en marcha, la tabla de localización de ficheros (FAT), el directorio y el espacio de datos (véase figura 5-7). El tamaño de cada sección varía según formatos, pero la estructura y el orden de las secciones no varía. Los discos duros, tales como el disco duro de 10 megabytes del XT, siguen la misma disposición básica, aunque los discos duros que pueden estar particionados presentan complicaciones adicionales, porque los tamaños de la partición afectan directamente al tamaño de cada sección. Véase en la página 124 la información sobre particiones en discos duros.

El **registro de puesta en marcha (boot)** es siempre un sector único situado en el sector 2 de la pista 0, cara 0. El registro de puesta en marcha contiene, entre otras cosas, un corto programa para comenzar el proceso de carga del sistema operativo. Todos los *diskettes* contienen el registro de puesta en marcha, aunque no tengan el sistema operativo. Aparte del programa de puesta en marcha, el contenido exacto del registro varía de un formato a otro.

Tabla de localización de ficheros, o FAT, va a continuación del registro de puesta en marcha, comenzando normalmente en el sector 2 de la pista 0, cara 0. La FAT contiene el registro oficial del formato del disco y los mapas de la localización de los sectores utilizados por los ficheros. El DOS utiliza la FAT para guardar un registro de la utilización del espacio de datos. Cada entrada de la tabla contiene un código específico para indicar el espacio que está siendo utilizado, el que está disponible y el espacio que está inutilizado (debido a defectos del disco). Al utilizarse la FAT para controlar todo el área utilizable de almacenamiento de datos, se conservan dos copias idénticas de ella, en previsión de que alguna se dañe. Ambas copias de la FAT pueden ocupar tantos sectores como

Formato	Sectores totales	Sectores ocupados				Sectores de datos
		Puesta en marcha	FAT	Directorio	Total	
S-8	320	1	2	4	7	323
D-8	640	1	2	7	10	630
S-9	360	1	4	4	9	351
D-9	720	1	4	7	12	708
QD-9	1440	1	10	7	18	1422
QD-15	2400	1	14	14	29	2371

Figura 5-6. Atribución de sectores de los formatos estándar de discos flexibles

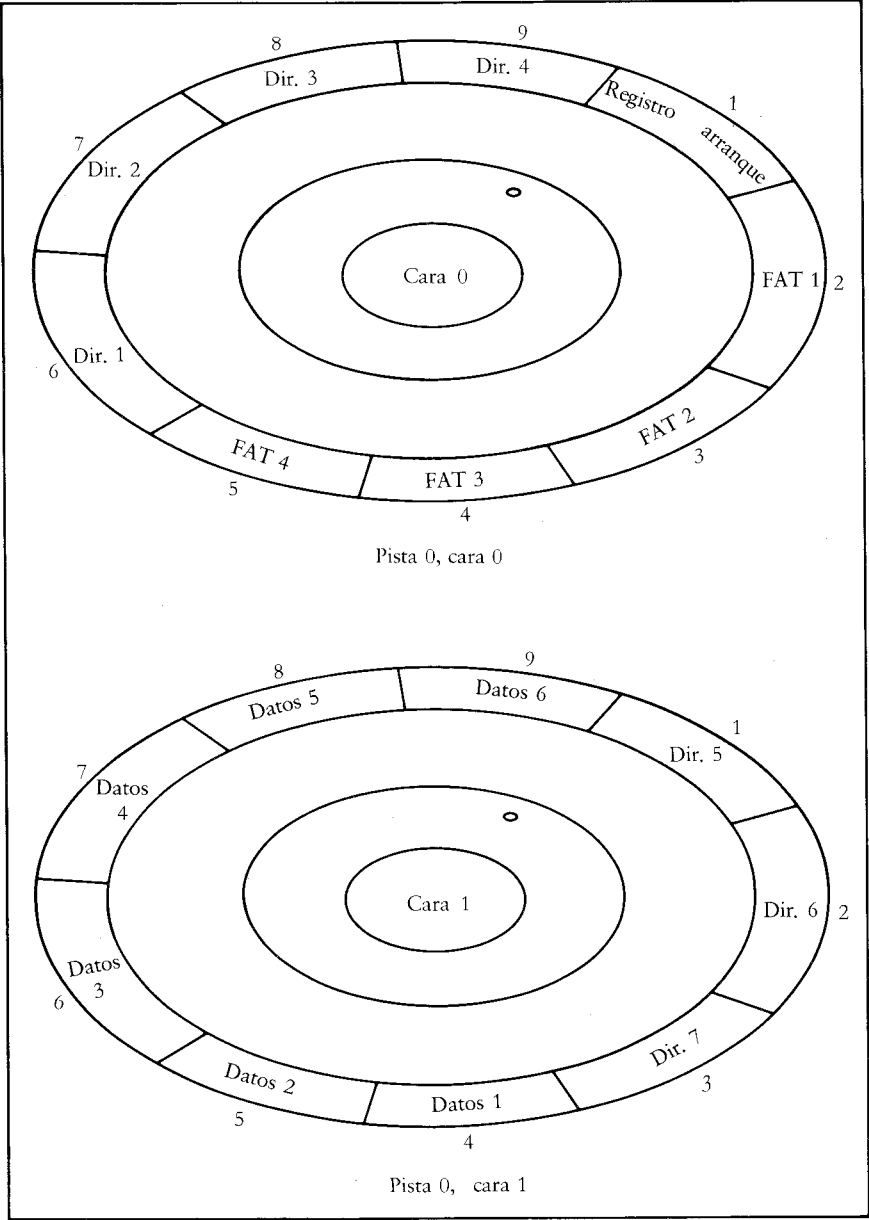


Figura 5-7. Las cuatro secciones lógicas de un diskette

se necesiten: 2 ó 4 en discos flexibles, 14 en los *diskettes* QD-15, más de 16 en el disco duro de XT y más de 82 en el disco duro del AT. En todos los tipos de disco duro el tamaño de la FAT varía con el tamaño de la partición.

El **directorio de ficheros** es el siguiente elemento del disco. Se utiliza como una tabla de contenidos, identificando cada fichero del disco con un elemento de directorio que contiene cierta cantidad de información, como el nombre y tamaño de los ficheros. Una parte de la entrada es un número que apunta al primer grupo de sectores utilizados por el fichero (este número es también la primera entrada de este fichero en la FAT). El tamaño del directorio varía según el formato del disco. Ocupa cuatro sectores en los *diskettes* de simple cara y siete en los de doble cara. En los discos duros, el directorio, como la FAT, varía según el tamaño de la partición.

El **espacio de datos**, que ocupa la mayor parte del *diskette* (desde el directorio hasta el último sector), se utiliza para almacenar datos realmente, mientras que las otras tres secciones se utilizan para organizar el espacio de datos. Los sectores del espacio de datos están organizados en unidades conocidas como *clusters*. El tamaño de un *cluster* varía según el formato. En *diskettes* de simple cara, los *clusters* ocupan un sector y en los de doble cara ocupan un par de sectores adyacentes. En los *diskettes* con una capacidad mayor pueden aparecer *clusters* que contengan varios sectores. **AT** Por ejemplo, el disco duro de 20 megabytes de AT utiliza un tamaño de *cluster* de cuatro sectores, **XT** y el disco duro de 10 megabytes del XT utiliza hasta ocho sectores por *cluster*.

Distribución del espacio del disco duro

Para sistemas de disco duro, la cantidad de espacio que el DOS otorga a la FAT, al directorio y al espacio de datos, depende del tamaño de la partición dada al DOS. El registro de puesta en marcha ocupa un sector independientemente del tamaño de la partición, así que no lo trataremos.

Para tener una idea de cómo el DOS distribuye el espacio en una partición, se va a ver tres tamaños de partición diferentes. Este ejemplo se refiere al disco duro de 10 megabytes del XT, que tiene 512 bytes por sector, 17 sectores por cilindro, 4 caras (cabezas) por cilindro y 306 cilindros por disco.

La tabla de la figura 5-8 muestra el espacio específico otorgado a las tres particiones del DOS: 305 cilindros (un disco, estilo XT entero), 100 cilindros y 5 cilindros. En general, estas figuras pueden ser interpoladas para determinar el espacio otorgado a los otros tamaños de partición.



	Tamaño de la partición (cilindros)		
	305 (todo)	100	5
Tamaño de la FAT (sectores)	8	5	1
Tamaño del directorio (sectores)	32	16	4
Número de elementos de directorio	512	256	64
Tamaño del <i>cluster</i> (sectores)	8	4	1
Número de <i>clusters</i>	2587	1699	333
Tamaño del espacio de datos (K)	10 348	3372	166.5

Figura 5-8. Distribución del espacio de las tres particiones del disco duro de 10 megabytes del XT

LA ESTRUCTURA LOGICA EN DETALLE

Conviene ahora estudiar un poco más profundamente cada una de las cuatro secciones del disco: el registro de puesta en marcha, el directorio, el espacio de datos y la tabla de localización de ficheros.

El registro de arranque (boot)

El registro de arranque consiste, primariamente, en un corto programa, en lenguaje máquina, que activa el proceso de carga del DOS en la memoria. Para realizar esta tarea, el programa comprueba primero si el disco está formateado por el sistema (si contiene los ficheros IBMBIO.COM y IBMDOS.COM) y entonces procede en consecuencia.

Se puede examinar el programa de arranque utilizando el programa DEBUG del DOS, que combina la posibilidad de leer datos de cualquier sector de un disco y la de desensamblar lenguaje máquina. Si desea conocer más a fondo el programa de arranque y no le asusta el parco formato del comando del DEBUG, trate de introducir estos comandos:

```
DEBUG
L 0 0 0 1           ; carga el primer sector
U 0 L 2             ; desensambla y lista los bytes primero y segundo
U 2E                ; desensambla y lista los bytes desde el 2E en adelante
```

Estos comandos le permiten ver las primeras instrucciones del programa de arranque del disco de la unidad A.

Particionamiento de un disco duro

Todo sistema operativo tiene su propia forma peculiar de formateado y gestión del espacio de almacenamiento del disco, que es incompatible con otros sistemas operativos. Aunque el DOS es, hasta ahora, el sistema operativo dominante en la familia PC, no es el único utilizado y, en unos pocos años, es bastante posible que el DOS pueda ser reemplazado por un nuevo sistema operativo, que bien pudiera ser el XENIX.

Dado que es posible que los diferentes sistemas operativos tengan que utilizar el mismo disco, se ha realizado un esquema para particionar o dividir un disco duro en secciones lógicas, de forma que varios sistemas operativos puedan utilizar cada uno una parte de él. Una partición es un conjunto de cilindros contiguos, cuyo tamaño está determinado por el usuario e implementado por el sistema operativo. Un disco duro debe ser particionado antes de que el sistema operativo pueda utilizarlo. Después de que es particionado, las particiones deben ser formateadas por separado, utilizando los procedimientos de formateado del sistema operativo de control. Normalmente, un sistema operativo utiliza sólo una partición. Sin embargo, ocasionalmente, algunos proveedores particionarán sus discos duros en el equivalente de unidades de disco separadas, de manera que el DOS utilice cada partición. Afortunadamente, esto es más una excepción que una regla.

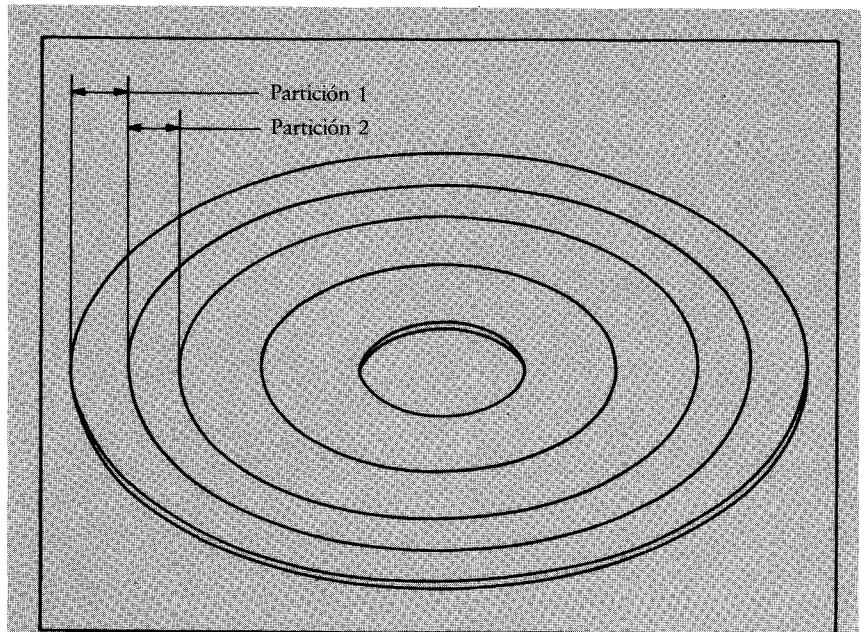
El programa FDISK del DOS se utiliza para crear las particiones y señalar una de ellas como la partición del DOS. Se puede especificar si se desea que la partición del DOS comprenda todo, o sólo una parte del disco. El sistema operativo puede adaptarse a cualquier tamaño de partición de disco que le sea asignado. Si no se prevé utilizar otro sistema operativo, es recomendable particionar el disco entero para el DOS.

Se puede variar el tamaño y número de las particiones tantas veces como se desee. Sin embargo, al cambiar la partición se destruye el contenido de la partición existente; así que cuando se varía el tamaño, o número de las particiones, habrá antes que guardar cualquier información que todavía sea útil, reparticionar el disco, y entonces volver a introducir esa información.

Formateado de partición

Un disco duro tiene dos niveles de estructura lógica. En un nivel se encuentra la división del disco en particiones: un fenómeno que es común a todos los sistemas operativos. En otro nivel, se encuentra el formato y localización específicos de la información que es almacenada en cada partición, que es diferente para cada sistema operativo. Una vez se ha establecido con el FDISK una partición del DOS, esta partición debe ser formateada mediante el comando FORMAT del DOS para establecer la estructura lógica que necesita el DOS para operar.

Como un disco del DOS contiene en su primer sector un registro de puesta en marcha que consta de un programa de inicialización y de cierta información general sobre el disco (■ véase página 120 para más detalles); el primer sector de un disco particionable tiene un registro de puesta en marcha *master*, que contiene un programa *master* de inicialización y un registro que indica cómo está particionado el disco.



Esta información del particionamiento incluye cuantas particiones hay (a menudo hay sólo una), el tamaño y localización de cada partición, el tipo de partición y qué partición está activa, además de otras informaciones. El programa *master* de inicialización es un programa corto que cede el control al programa de inicialización, o arranque, de la partición activa.

Hasta ahora, estas observaciones generales se refieren a cualquier disco que pueda ser particionado para trabajar con varios sistemas operativos. Para ilustrar el mecanismo de particionamiento, se tomará, como ejemplo, el disco duro de 10 megabytes del XT.

El registro *master* de arranque del disco duro del XT contiene una tabla de particiones con espacio para hasta cuatro. Cada partición está marcada en la tabla para indicar si está activa o no, y tiene un byte de ID para identificar el sistema operativo que la gobierna (01 hex identifica al DOS).

La localización y tamaño de la partición están almacenados en la tabla en el registro *master* de arranque, de dos formas equivalentes: la primera forma señala la localización de comienzo y final de la partición, utilizando los números del cilindro (o pista), la cabeza (o cara) y el sector, de los sectores primero y último de la partición; la segunda forma proporciona el número de sector del primer sector de la partición en relación con el primer sector del disco, seguido por el número total de sectores de la partición.

Cada partición ocupa un conjunto contiguo de cilindros, que comienza en el primer sector del primer cilindro del conjunto y termina en el último sector del último cilindro. Una excepción a esto se presenta en la partición que utiliza el primer cilindro del disco. En este cilindro la partición empieza en el segundo sector, porque el primer sector está ocupado por el registro *master* de arranque del disco.

❑ **NOTA:** El código que comienza en el byte 2E es el comienzo del programa de arranque, tan sólo en el caso del IBM PC estándar. Para todos los modelos, el segundo byte del primer sector almacena la localización del primer byte del código de arranque, por lo que si no se está utilizando un PC, cuando se introduzca la tercera línea de comandos, se deberá de utilizar el valor del byte mostrado en la instrucción JMP producido por la segunda línea de comandos, en lugar del valor 2E.

En todos los formatos de disco, excepto el S-8 y el D-8, se encontrarán en el registro de arranque algunos parámetros clave, que comienzan con el cuarto byte (■ véase figura 5-9). Estos parámetros son parte del bloque de parámetros del BIOS utilizados por el DOS para controlar cualquier dispositivo tipo disco. El resto del programa de arranque empieza en los primeros tres bytes (bytes 0, 1 y 2) y continúa en los bytes siguientes del bloque de parámetros del BIOS. Al final de los registros de arranque de las versiones DOS 2.00 y siguientes hay una signatura de 2 bytes, 55 AA hex.

Offset	Longitud	Descripción
3	8 bytes	ID del sistema (p. ej. IBM 3.1)
11	1 palabra	Número de bytes por sector (p. ej. 512, 0200 hex)
13	1 byte	Número de sectores por <i>cluster</i> (p. ej. 01 ó 02)
14	1 palabra	Número de sectores reservados al principio: 1 para <i>diskettes</i>
16	1 byte	Número de copias de la FAT: 2 para <i>diskettes</i>
17	1 palabra	Número de elementos en el directorio raíz (p. ej. 64 ó 112)
19	1 palabra	Número total de sectores en el disco (p. ej. 720 para el D-9)
21	1 byte	De formato (p. ej. FF, FE, FD o FC)
22	1 palabra	Número de sectores por FAT (p. ej. 1 ó 2)
24	1 palabra	Número de sectores por pista (p. ej. 8 ó 9)
26	1 palabra	Número de caras (cabezas) (p. ej. 1 ó 2)
28	1 palabra	Número de sectores especiales reservados

Figura 5-9. Parámetros del registro de arranque

El directorio

Los directorios de los discos se utilizan para almacenar la mayor parte de la información básica sobre los ficheros contenidos en el disco, incluyendo el nombre de los ficheros, su tamaño, el comienzo

del elemento de la FAT, la hora y fecha en que fueron creados y unas pocas características especiales del disco (☛ véase figura 5-10). La única información que no contiene el directorio es la localización exacta de los *clusters* individuales que componen el fichero; éstos están almacenados en la tabla de localización de ficheros.

Hay un elemento en el directorio para cada fichero del disco, incluyendo los ficheros de subdirectorio y la etiqueta de identificación de volumen del disco. Cada uno de los elementos tiene 32 bytes, por lo que un sector del directorio puede almacenar 16 elementos. Los *diskettes* de simple cara con cuatro sectores de directorio pueden almacenar 64 elementos. Los *diskettes* de doble cara con siete sectores pueden almacenar 112 elementos de directorio. Los subdirectorios son tratados como ficheros, y no existe ningún límite para el número de elementos de subdirectorio que pueden almacenar. (☛ Para más información sobre directorios, véase página 128.) Cada entrada de 32 bytes del directorio está dividida en ocho campos.

Campo 1: el nombre del fichero

Los primeros ocho bytes de cada elemento de directorio contienen el nombre del fichero, almacenado en formato ASCII. Si el nombre de fichero tiene menos de ocho caracteres, se completa por la derecha con espacios en blanco (CHR\$(32)). Las letras deberán de ser mayúsculas, puesto que las minúsculas no son reconocidas correctamente. Normalmente, no debería haber espacios en blanco intercalados con el nombre de fichero, como en AA BB. La mayoría de los programas de comandos del DOS, tales como el DEL o el COPY, no

Campo	Offset	Descripción	Tamaño (bytes)	Formato
1	0	Nombre de fichero	8	Caracteres ASCII
2	8	Extensión del nombre de fichero	3	Caracteres ASCII
3	11	Atributo	1	Bit codificador
4	12	Reservado	10	No utilizado; ceros
5	22	Hora	2	Palabra, codificada
6	24	Fecha	2	Palabra, codificada
7	26	Comienzo de entrada a la FAT	2	Palabra
8	28	Tamaño de fichero	4	Entero

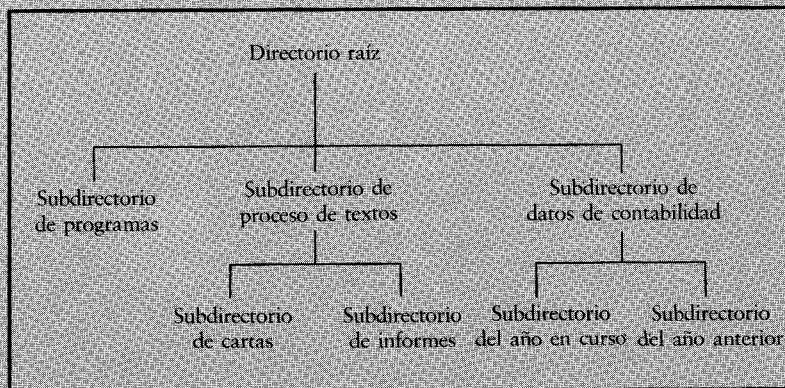
Figura 5-10. Las ocho partes de un elemento de directorio

Subdirectorios

Hay dos tipos de directorios: directorios raíz y subdirectorios. El contenido y utilización de cada tipo son esencialmente los mismos: ambos almacenan los nombres y localizaciones de ficheros del disco, pero sus características son diferentes. El directorio raíz (al que hasta ahora se ha designado con el nombre de directorio, simplemente) tiene un tamaño fijo y está almacenado en una localización fija del disco. Un subdirectorio es una adición al directorio raíz, no tiene tamaño fijo y se puede guardar en cualquier lugar del disco. Cualquier disco utilizado con la versión DOS 2.00, o posteriores, puede incluir subdirectorios.

Un subdirectorio se almacena en la zona de datos del disco, como cualquier otro fichero. El formato de campo y el contenido de un subdirectorio son idénticos a los de un directorio, excepto que los subdirectorios no están limitados en cuanto al tamaño. Como ocurre con cualquier otro fichero, el tamaño de un subdirectorio tan sólo está limitado por el propio espacio disponible en el disco. Los subdirectorios pueden ser creados y utilizados en cualquier tipo de disco. Sin embargo, puesto que los subdirectorios ocupan bastante espacio de datos, suelen utilizarse principalmente en discos duros de alta capacidad; su utilización en *diskettes* se evita generalmente.

Los subdirectorios están siempre conectados a un directorio padre, que puede ser el directorio raíz, u otro subdirectorio, y pueden ramificarse en varios niveles de directorios, formando una estructura de árbol.



El directorio padre tiene un elemento para cada uno de sus subdirectorios, que es igual a los elementos correspondientes a ficheros, excepto en que el byte de atributo lo marca como un subdirectorio y el campo de tamaño de fichero está puesto a cero. El tamaño real del subdirectorio queda determinado siguiendo su cadena de localización a partir de la FAT.

Cuando se crea un subdirectorio, se introducen en él dos elementos especiales, con "." y ".." como nombres de fichero. Estas actúan como

otros subdirectorios, pero “.” se refiere al propio subdirectorio, y “..” se refiere a su directorio padre. El número de *clusters* de comienzo determina la localización del propio subdirectorio, o la de su directorio padre. Cuando el número de *clusters* de comienzo es cero, significa que el padre de este directorio es el directorio raíz.

Cuando se reduce el tamaño de un fichero, el DOS puede generalmente reajustar el espacio, de manera que se aproveche el libre. Sin embargo, en el caso de los subdirectorios, los *clusters* vacíos que ya no se van a utilizar (porque los elementos que ocupaban aquel espacio se han borrado) no quedarán disponibles hasta que se elimine el subdirectorio entero.

reconocerá nombres de fichero con espacios en blanco intercalados. Sin embargo, el BASIC puede trabajar satisfactoriamente con estos nombres de fichero, y los servicios del DOS (véase capítulos 15 y 16) normalmente también pueden hacerlo. Esta capacidad posibilita la utilización de algunos trucos útiles, tales como crear ficheros que no puedan ser borrados fácilmente.

También pueden aparecer en el primer byte del campo del nombre de fichero tres códigos, que se utilizan para indicar situaciones especiales. El primer byte de los directorios que no se utilizan toma el valor 00 hex. Esto hace posible que el DOS sepa cuándo no hay ningún directorio activo más, sin tener que buscar hasta el final del directorio. Esta convención empezó con la versión DOS 2.00, y también aparece en las versiones posteriores, pero no en las primeras versiones del DOS-1.

El hecho de que el primer byte del campo del nombre de fichero sea E5 hex, indica normalmente que el fichero ha sido borrado. Sin embargo, puesto que las versiones DOS-1 no utilizaban el código 00 (nunca utilizado), el valor E5 hex en este campo puede indicar que un fichero ha sido borrado, o que no ha sido utilizado nunca.

Cuando se borra un fichero, sólo quedan afectadas dos cosas del disco. Al primer byte de nombre de fichero se le asigna el valor E5 hex, y la cadena de utilización de espacio del fichero se borra en la FAT; esto se verá con más detalle en la sección dedicada a la FAT. El resto de la información del directorio acerca del fichero se mantiene, como, por ejemplo, el resto de su nombre, su tamaño e incluso su número de *cluster* de comienzo. La información perdida puede ser recuperada mediante métodos adecuados, siempre que el elemento de directorio no se haya utilizado por otro fichero. Se advierte que cuando es necesario crear un nuevo elemento de

directorio, el DOS utiliza el primero disponible, reciclando rápidamente la entrada correspondiente a un fichero ya borrado, haciendo imposible la recuperación.

El tercer código que se puede encontrar en el byte del nombre de fichero es el carácter punto, 2E hex, que se utiliza para especificar un subdirectorio (véase página 128). Si el segundo byte es también 2E hex, indica que se trata del directorio padre del subdirectorio en uso, en cuyo caso el campo del *cluster* de comienzo (campo 7) contiene el número de *cluster* del directorio padre.

Campo 2: extensión del nombre de fichero

Inmediatamente a continuación del nombre de fichero se encuentra la extensión estándar del nombre de fichero, en formato ASCII. Son tres bytes y, como el nombre de fichero, se completa con espacios en blanco, si consta de menos de tres caracteres, hasta alcanzar ese número. Mientras un nombre de fichero debe tener al menos un carácter ordinario, la extensión puede consistir en tres espacios en blanco. Generalmente sirven las mismas reglas para extensión que para el propio nombre.

□ *NOTA: Cuando el directorio contiene etiqueta de volumen, los campos del nombre de fichero y la extensión son tratados como un campo combinado de once bytes. En este caso están permitidos los espacios en blanco intercalados. Normalmente, las letras minúsculas no se utilizan en las etiquetas, pero no hay ningún inconveniente en hacerlo.*

Campo 3: atributo del fichero

El tercer campo de la entrada de directorio consta de un byte, y cada uno de sus bits se utiliza para caracterizar el elemento de directorio. Los bits del byte de atributo se codifican por separado como bits del 0 al 7, como se muestra en la figura 5-11.

El **bit 0**, el de menor orden, caracteriza a un fichero como de sólo lectura. En este estado, el fichero está protegido contra cualquier operación de cambio o borrado del DOS. Hay que advertir que las versiones del DOS-1 ignoran este atributo, así que, aunque proporcionan cierta protección a los datos, es una protección débil.

Los **bits 1 y 2** caracterizan a los ficheros como ocultos, o ficheros del sistema. Los ficheros señalados como ocultos, o del sistema, o ambos, no pueden ser tratados mediante operaciones ordinarias del DOS, como el comando DIR. Se puede guardar cualquier programa

Bit								Valor		Significado
7	6	5	4	3	2	1	0	Dec	Hex	
.	.	.	.	.	.	.	1	1	1	Solo lectura
.	.	.	.	.	.	1	.	2	2	Oculto
.	.	.	.	.	1	.	.	4	4	Sistema
.	.	.	.	1	.	.	.	8	8	Etiqueta de volumen
.	.	.	1	.	.	.	.	16	10	Subdirectorio
.	.	1	.	.	.	.	.	32	20	Archivo
.	1	.	.	.	.	.	.	64	40	No utilizado
1	.	.	.	.	.	.	.	128	80	No utilizado

Figura 5-11. Los ocho bits de atributo del fichero

en fichero de este tipo sin más que introducir estos bits de atributo en el bloque de control de ficheros, o FCB (*file control block*) (véase página 323). Los ficheros IBMBIO.COM y IBMDOS.COM del DOS (que también pueden aparecer bajo los nombres IO.SYS y MSDOS.SYS) son, ambos, ocultos y del sistema. El atributo del sistema no posee un significado particular; existe para perpetuar una característica del CP/M, y no tiene absolutamente nada que ver con el DOS.

El bit 3 indica que el elemento de directorio es una etiqueta ID de volumen del disco. Las etiquetadas sólo son reconocidas correctamente en el directorio raíz, y sólo utiliza unos pocos de los ocho campos disponibles en el elemento. La etiqueta, propiamente dicha, está almacenada en los campos del nombre de fichero y la extensión, que son tratados como un campo unificado para este propósito. Los campos de tamaño y de *cluster* de comienzo no se utilizan, pero sí los de fecha y hora.

El bit 4, el atributo de subdirectorio, sirve para identificar elementos de directorio que identifican los subdirectorios. Dado que los subdirectorios están almacenados en el disco como ficheros de datos ordinarios, necesitan un elemento de directorio propio. Estos elementos utilizan todos los campos del directorio, excepto el campo de tamaño del fichero, que toma el valor cero. El tamaño real de un subdirectorio se determina siguiendo su cadena de localización, que hay que buscar en la FAT.

El bit 5, el atributo de archivo, fue creado para facilitar la realización de copias de seguridad de algunos ficheros que puedan estar almacenados en un disco duro. Este bit está a cero en todos los ficheros que no han cambiado desde la última copia de seguridad; el

bit está normalmente a uno en todos los ficheros de un *diskette*. El atributo de archivo no resulta particularmente útil en los *diskettes*.

Campo 4: reservado

Este área de 10 bytes está reservada para posibles usos futuros. Los 10 bytes contienen normalmente el valor 00 hex.

Campo 5: la hora

El campo 5 contiene un valor de 2 bytes, que señala la hora en que fue creado, o sufrió, su último cambio el fichero. Se utiliza en combinación con el campo de fecha, y los dos juntos pueden ser tratados como un único entero sin signo de 4 bytes.

Este entero de 4 bytes puede ser comparado con los de otros elementos de directorio, estableciendo relaciones de superioridad, inferioridad e igualdad. La hora, por sí misma, se trata como un entero de 2 bytes sin signo, con el byte menos significativo en primer lugar, que se obtiene a partir de la hora, los minutos y los segundos, utilizando esta fórmula:

$$\text{Hora} \times 2048 + \text{Minutos} \times 32 + \text{Segundos} / 2 = \text{Tiempo}$$

La hora está basada en un reloj de 24 horas, con un valor entre 0 y 23. Puesto que la palabra de dos bytes utilizados para almacenar el tiempo no tiene bits suficientes para almacenar todos los segundos, se guardan en unidades de 2 segundos de 0 al 29; un valor de 5, por ejemplo, representaría 10 segundos. La hora 11:32:10 sería almacenada como 23557.

Campo 6: la fecha

El campo 6 contiene un valor de 2 bytes que indica la fecha en que se creó el fichero, o en que se modificó por última vez. Se utiliza en combinación con el campo de la hora, y los dos juntos se pueden tratar como un único entero sin signo de 4 bytes que puede ser comparado con los otros elementos de directorio, estableciendo relaciones de superioridad, inferioridad e igualdad. La fecha, por sí misma, se trata como un entero de 2 bytes sin signo, con el byte menos significativo en primer lugar, que se obtiene a partir del año, del mes y del día, mediante esta fórmula:

$$\text{Fecha} = (\text{Año} - 1980) \times 512 + \text{Mes} \times 32 + \text{Día}$$

Adviértase que esta fórmula reduce el año, restándole 1980. Así, el año 1984 será calculado como el valor 4. Utilizando esta fórmula, 12 de diciembre de 1984 será almacenado, aplicando la fórmula, como 2444:

$$(1984 - 1980) \times 512 + 12 \times 32 + 12 = 2444$$

Aunque de esta forma se podrían manejar fechas de hasta el año 2108, el último que admite el DOS es el 2099.

Campo 7: número de cluster de comienzo

El séptimo campo consta de 2 bytes que indican el número de *cluster* de comienzo del espacio de datos del fichero. Actúa como el punto de entrada en la cadena de localización del fichero en la FAT. Para los ficheros sin espacio localizado y etiquetas de volumen, el número de *cluster* de comienzo es cero, en lugar del valor FFF hex que se utiliza en la FAT para indicar el final del fichero.

Campo 8: tamaño del fichero

El último campo del elemento de directorio indica el tamaño del fichero en bytes. Está codificado como un entero sin signo de 4 bytes, lo que permite disponer de ficheros de buen tamaño; de hecho, tanto como permita la capacidad del disco utilizado.

Para el DOS, el tamaño indicado por este campo es el verdadero tamaño del fichero. Sin embargo, a veces este valor almacenado es mayor que el tamaño real del fichero. Por ejemplo, algunos ficheros de texto ASCII creados por los procesadores de textos señalan el verdadero final de fichero con el carácter Ctrl-Z (CHR\$ (26), 1A hex). Para estos ficheros, el atributo de tamaño de fichero puede presentar un número mayor, como el siguiente múltiplo de 128 bytes. Esto ocurre con frecuencia en la mayoría de los programas editores de texto, que leen y escriben datos en bloques grandes, en vez de hacerlo tomando un byte cada vez. Es importante resaltar que, cuando el DOS está leyendo un fichero, establece el final de fichero cuando lee todo el fichero, según su tamaño, o cuando llega al final de la cadena de localización de la FAT (denotado por FFF hex), sea cual sea el que aparezca primero.

El espacio de datos

Todos los ficheros de datos y subdirectorios (que actúan como ficheros de datos) están almacenados en el espacio que ocupa la parte última y más grande del disco.

El espacio se otorga a los ficheros a medida que lo necesitan, un *cluster* cada vez. (Recuerde, un *cluster* está formado por uno o más sectores consecutivos; el número de sectores por *cluster* es una característica fija de cada formato de disco.) Cuando se crea un fichero, o cuando se amplía uno ya existente, el espacio de atribución del fichero aumenta. Cuando es necesario más espacio, se atribuye un *cluster* al fichero. En las versiones 1 y 2 del DOS se añade siempre al fichero el primer *cluster* disponible. Las versiones posteriores del DOS seleccionan los *clusters* mediante reglas más complicadas, en las que no se va a entrar.

En la mayoría de las ocasiones, cada fichero se almacena en un bloque espacio contiguo. Sin embargo, un fichero puede dividirse en varios bloques no contiguos, especialmente cuando se añade información a un fichero existente, o cuando se almacena un nuevo fichero en el espacio dejado por un fichero borrado. No es inusual que un fichero de datos esté disperso por todo el disco.

Esta especie de fragmentación del fichero ralentiza de alguna forma el acceso a los datos del fichero. Además es mucho más difícil «recuperar» un fichero que se ha borrado involuntariamente si está fragmentado, simplemente porque hay que realizar muchas más operaciones de búsqueda en cada uno de los sectores individuales que constituyen el fichero. Pero la fragmentación no tiene otros efectos. En general, los programas no se preocupan por el lugar donde están almacenados sus datos en el disco. Pero si se desea saber si un fichero está fragmentado, hay dos formas muy sencillas de averiguarlo. Se puede utilizar la opción /V del comando CHKDSK, o también un programa como las *Norton Utilities*, para visualizar un mapa gráfico de la localización de cada fichero en el disco.

Si los ficheros de un *diskette* están fragmentados, se puede reagrupar en un *diskette* vacío ya formateado. Naturalmente, el fichero puede fragmentarse otra vez si hay gran actividad de actualización en el disco. En un disco duro es más difícil eliminar la fragmentación. Pero no hay que preocuparse demasiado por esto. Se ha mencionado como mera información, pero, en la práctica, un fichero fragmentado no provoca ninguna complicación.

Haya o no haya ficheros fragmentados, será útil comprender cómo utiliza el DOS la tabla de localización de ficheros (FAT) para distribuir el espacio del disco y cómo la FAT forma una cadena de localización para conectar todos los *clusters* que constituyen un fichero.

Tabla de localización de ficheros (FAT)

La tabla de localización o asignación de ficheros (*File allocation table*) almacena un registro, que muestra cómo se está asignando el espacio del disco. Hay que distinguir entre cómo está organizado la FAT, que es relativamente simple e inmediato, y cómo está almacenada en el disco, lo cual es más complejo.

Como ya se ha reseñado, los formatos de disco estándar almacenan dos copias de la FAT, aunque pueden ser más de dos, o incluso sólo una copia. Cada copia de la FAT ocupa un sector en los *diskettes* de ocho sectores y dos sectores en los de nueve sectores. En los formatos de *diskette* de alta capacidad designados por QD-15, la FAT utiliza siete sectores.

En la mayor parte de los formatos de disco, el DOS realiza dos copias de la FAT, por si una de ellas se estropea o queda ilegible, pero no parece hacer uso de esta redundancia. El programa CHKDSK, que detecta la mayor parte de los errores que pueden producirse en la FAT y el directorio, no detecta si las dos FAT son diferentes.

Hay dos formatos de FAT: uno de 12 bits y otro de 16 bits. El formato de FAT de 12 bits es más común, y el más complicado de los dos. La FAT de 16 bits se utiliza sólo con discos que exceden la capacidad de 12 bits, como ocurre con el disco duro de 20 megabytes del AT. Se verá primero el FAT de 12 bits estándar, y después se explicará en qué se diferencia del FAT de 16 bits.

La FAT está organizada como una tabla de hasta 4096 números, que van desde el 0 al 4095 (0 al FFF hex), con un elemento para cada *cluster* en el espacio de datos. El número de cada elemento indica el estado y uso del *cluster* correspondiente. Adviértase que el rango de los números guardados en la FAT están definidos de forma que no excedan de tres dígitos hexadecimales. Esto es clave para comprender cómo está almacenada la FAT de 12 bits, como se va a ver brevemente.

Si el elemento de la FAT es 0, se indica que el *cluster* está libre y disponible para su uso. Si el elemento de la FAT es 4087 (FF7 hex), el *cluster* está declarado como inutilizable por un error de formateo; esto se llama también marca de pista-mala (*bad-track*). Los valores de la FAT del 4081 al 4086 (FF1 al FF6 hex) se reservan también para señalar la imposibilidad de utilizar un determinado *cluster*, pero no se utilizan.

□ **NOTA:** Es importante hacer una pausa aquí para advertir que no es inusual ni alarmante tener «bad tracks» marcados en un disco, particularmente en un disco duro. De hecho, es bastante común en un

disco duro tener unos cuantos remiendos. Por ejemplo, el disco duro del AT utilizado para escribir este libro tiene tres pequeñas áreas bad track. El procedimiento de formateado del disco detecta las pistas malas y las marca como tales en la FAT, como se acaba de ver. Después, el marcado de pistas malas le indica al DOS que debe saltarse esas áreas marcadas. Las pistas malas son también comunes en los discos flexibles; con éstos, a diferencia de lo que ocurre con un disco duro, existe la opción de desecharlos y sólo utilizar discos perfectos.

Los *clusters* están numerados por orden del 2 a un número que es superior en una unidad al número de *clusters* del disco (☛ véase figura 5-12). Una entrada de la FAT de 12 bits que contenga cualquier número entre 2 y 4080 (02 y FF0 hex) indica que el *cluster* correspondiente está siendo utilizado por un fichero. Un valor de la FAT de 4095 (FFF hex) indica que el correspondiente *cluster* contiene la última parte de los datos del fichero. Los valores entre 4008 y 4094 (FF8 al FFE hex) tendrían el mismo significado, pero no se utilizan.

Con todo esto en mente, se puede ver que las entradas de la FAT forman una *cadena de localización*; el elemento del directorio del fichero contiene el número de *cluster* de comienzo (☛ véase página 133) y las entradas de la FAT designan los demás *clusters* utilizados y el final del fichero (☛ véase figura 5-13). Cuando un fichero es borrado, todos los elementos de la FAT que determinan su cadena de localización de espacio son marcadas como disponibles (puestos a 0); pero los datos del fichero real en el espacio de datos no sufren modificación alguna, y la mayor parte de la información del elemento del directorio del fichero se conserva.

Aunque la FAT está organizada como una simple tabla de valores numéricos, está almacenada de una forma bastante compleja, con

Formato	Sectores	Sectores por cluster	Clusters	Rango del número de cluster
S-8	313	1	313	2 al 314
D-8	630	2	315	2 al 316
S-9	351	1	351	2 al 352
D-9	708	2	354	2 al 355
QD-9	1422	2	711	2 al 712
QD-15	2371	1	2371	2 al 2372

Figura 5-12. Número de clusters para los diferentes formatos del DOS

Elemento de la FAT	Valor		Significado	
	Dec	Hex		
Desde el directorio; comienzo de la cadena de localización de fichero	0	253	FD	El disco es de cara doble, doble densidad
	1	4094	EFE	Entrada no utilizada, disponible
	2	3	003	El siguiente <i>cluster</i> del fichero es el <i>cluster</i> 3
	3	5	005	El siguiente <i>cluster</i> del fichero es el <i>cluster</i> 5
	4	4087	FF7	El <i>cluster</i> no es utilizable; pista mala
	5	6	006	El siguiente <i>cluster</i> de fichero es el <i>cluster</i> 6
	6	4095	FFF	Ultimo <i>cluster</i> del fichero y final de esta cadena de atribución de espacio
	7	0	0	Entrada no utilizada

Figura 5-13. Cadena de atribución de espacio de un fichero en la tabla de atribución de ficheros

objeto de hacer la tabla tan compacta como sea posible. Para conseguirlo, hace uso de algunos trucos del formato de datos del 8088, específicamente del almacenamiento inverso. La FAT sacrifica la simplicidad en aras de la eficacia.

El rango de los números de *clusters* está definido de forma que los elementos de la FAT sean 4095 (FFF hex), o menos. Esto hace posible almacenar cada elemento de 3 dígitos hexadecimales en 12 bits, o 1 ½ bytes. Los elementos de la FAT se agrupan por pares, ocupando cada uno 3 bytes (0 y 1 ocupan los primeros 3 bytes, 2 y 3 los siguientes 3 bytes, y así sucesivamente). Los tres bytes se decodifican de la siguiente forma: si un par de elementos de la FAT está formado por 123 y 456 hex, los tres bytes que contienen serían, en hexadecimal, 23, 61 y 45. En sentido inverso, si los tres bytes son AB, CD, EF, los dos valores de la FAT son DAB y EFC. Como se aprecia en la figura 5-14, en los formatos S-8, 5-9 y D-8, el último número de *cluster* es impar y, consecuentemente, está emparejado con un elemento ficticio de la FAT.

Este sistema parece curioso cuando se utiliza en términos ordinarios, pero es rápido y eficiente cuando trabaja con instrucciones en lenguaje máquina. Dado cualquier número de *cluster*, se puede encontrar el valor de FAT multiplicando el número de *cluster* por 3, dividiendo por 2 y utilizando el número completo del resultado como un desplazamiento en la FAT. Cogiendo una palabra de esa ubicación, se tendrán los tres dígitos hexadecimales del elemento de la FAT, más un dígito hexadecimal extraño, que se puede hacer desaparecer con una de las rápidas instrucciones del lenguaje

máquina. Si el número de *cluster* es impar, se desecha el dígito de mayor orden; si es par, el dígito de menor orden. El valor obtenido de esta manera es el número siguiente del *cluster* del fichero, a menos que sea FFF, que indica el último *cluster* del fichero.

Este complejo esquema fue diseñado originalmente para los formatos de *diskette* de 8 sectores. No resulta tan eficaz cuando se utiliza para otros formatos, como los formatos de 9 sectores, donde la FAT llega a ser ligeramente mayor que un sector. En conjunto, sin embargo, es un método muy preciso y eficiente.

Los detalles reseñados hasta ahora son válidos para las FAT de 12 bits, que pueden alojar hasta 4080 *clusters*. Si un formato de disco tiene un número superior de *clusters*, es necesario utilizar la FAT de 16 bits.

La FAT de 16 bits trabaja de la misma forma que la de 12 bits, pero es más simple. Las entradas en la FAT de 16 bits son, obviamente, 4 bits más largas, lo que permite un mayor rango de números de *clusters*. Puesto que 16 bits son exactamente dos bytes, o una palabra, la FAT de 16 bits no necesita la disposición de almacenamiento comprimido utilizado con la FAT de 12 bits. En cambio, la FAT de 16 bits es una simple tabla de valores de 2 bytes, almacenados a continuación del otro.

Los valores especiales para la FAT de 16 bits (para señalar, por ejemplo, las pistas malas) son una extensión lógica de las utilizadas en las FAT de 12 bits, únicamente se ha añadido el valor F en la posición de mayor orden. Por ejemplo, el valor de final de fichero es el FFFF hex (en lugar del FFF), y el valor de *cluster* malo es FFF7 hex (en lugar de FF7).

Como ya se ha dicho, los *clusters* de datos están numerados desde el 2, mientras que cada FAT comienza con los elementos 0 y 1. Estos dos primeros elementos de la FAT, tanto en el formato de 12 bits como

AT



Formato	Byte de ID
D-8	FF
S-8	FE
D-9	FD
S-9	FC
QD-9	F9
QD-15	F9

Figura 5-14. Valores del byte de ID de los formatos de disco habituales

en el de 16 bits, no se utilizan para indicar el estado de los *clusters*; en cambio, están reservados para que el primer byte de la FAT pueda ser utilizado como byte de ID, indicando el formato del disco. Sin embargo, no se debe suponer que estos ID, por sí solos, identifican los formatos: no necesariamente. Si se considerasen todos los formatos de disco en uso, se encontrarían bastantes repeticiones. Cuidado.

Cualquier programa puede reconocer el formato de un disco leyendo e identificando el byte ID de la FAT. Sin embargo, la manera más usual de identificar el formato es utilizar la función 27 (1B hex) del DOS. (■ Para más información sobre esta función, véase página 317.)

Notas especiales de la FAT

Normalmente, los programas no atienden ni modifican la FAT de un disco; la FAT queda completamente bajo la supervisión del DOS. Las únicas excepciones son los programas que realizan funciones de atribución de espacio que no competen al DOS; por ejemplo, programas que recuperan ficheros borrados, como el programa Un-Erase de las *Norton Utilities*.

Es importante advertir que se puede dañar una FAT; por ejemplo, una cadena de localización puede ser circular, llegando a unirse a un punto anterior de la misma cadena, o dos cadenas pueden converger en un mismo *cluster*, o un *cluster* puede quedarse huérfano, lo que significa que está marcado como en uso, incluso aunque no sea parte de alguna cadena de localización válida. También puede perderse un indicador de fin de fichero (FFF o FFFF hex). Los programas CHKDSK y RECOVER del DOS están diseñados para detectar y reparar la mayor parte de estos problemas, de la mejor forma en que, razonablemente, sea posible.

■ Para notas especiales sobre la interacción de la cadena de localización de espacio en la FAT y el registro del tamaño del fichero del DOS, véase página 133.

COMENTARIOS

Aunque en este capítulo se ha proporcionado información detallada para la utilización directa del disco, sobre el registro de arranque, la FAT y los directorios, no es una buena idea utilizarla directamente, a menos que se tenga un motivo muy concreto. En efecto, excepto cuando sea completamente inevitable, como en un programa de protección de copias, es imprudente incorporar estos

conocimientos del formato del disco en los programas. En suma, la mejor cosa que se puede hacer es respetar la jerarquía estándar de operaciones y utilizar el más alto nivel de servicios que pueda satisfacer unas necesidades concretas:

- Primera posibilidad: Servicios de lenguaje (facilidades proporcionadas por el lenguaje de programación utilizado, como, por ejemplo, las instrucciones OPEN y CLOSE del BASIC).
- Segunda posibilidad: Servicios del DOS (descritos en los capítulos 15 y 16).
- Tercera posibilidad: Servicios de disco de la ROM-BIOS (descritos en el capítulo 10).
- Última posibilidad: Control directo (por ejemplo, programación directa del controlador del disco flexible (FDC) a través de los comandos enviados a través de los puertos).

La mayoría de las operaciones con discos en la familia PC puede realizarse satisfactoriamente con los servicios que proporciona su lenguaje de programación. Sin embargo, hay dos circunstancias obvias que pueden llamar a métodos exóticos. Uno, que ya ha sido mencionado, es cuando su programación incluye el control de un disco al mismo nivel que el control que ejerce el DOS. Esto sería necesario si se estuviera escribiendo un programa similar al CHKDSK del DOS, o a las *Norton Utilities*. La otra circunstancia tiene que ver con la protección contra copia. Todos los esquemas de protección copia, de una u otra forma, incluyen alguna variedad de E/S de *diskette* no convencional. Esto conduce, usualmente, a la utilización de los servicios de la ROM-BIOS, pero puede conducir a la medida extrema de programar directamente el controlador del disco flexible.

Protección de copias

Hay una gran variedad de esquemas de protección contra copia disponibles en el mercado, como uno muy poco sofisticado, que es parte del conjunto de *software* del autor, Access Tools para el IBM/PC. Sin embargo, cada uno puede inventar su propio esquema.

Hay docenas de formas de conseguir la protección anticopia. Quizá los métodos más comunes suponen tener que volver a formatear los sectores de ciertas pistas del disco, utilizando las rutinas de formateado de la ROM-BIOS, puesto que el DOS no puede copiar un disco que tenga sectores de tamaños extraños junto con sectores normales. Naturalmente, esta limitación del DOS ha impulsado a gran número de compañías a producir programas de copia que pueden leer y copiar sectores de cualquier tamaño, por lo que no resulta un método particularmente efectivo de protección de copias.

En un nivel más avanzado, hay dos conceptos que destacan en la protección contra copia. Primero, alguno de los esquemas de protección más exóticos y efectivos han estado basados en el descubrimiento de prestaciones no documentadas, ocultas en el controlador del disco flexible (FDC). Segundo, algunos esquemas de protección dependen voluntaria o involuntariamente de las características particulares de las diferentes unidades de *diskette*. O pueden depender de los detalles del *software* de control de la unidad, que pueden diferir dentro de la misma familia PC. Por ejemplo, el *software* de control de la unidad de disco del PCjr trabaja de una forma muy diferente a los de los PC y XT. El *software* del AT es también diferente. Así un programa de protección contra copia puede funcionar en un modelo de ordenador, pero ser totalmente inútil en otro modelo, incluso aunque la protección de copia no haya sido alterada. Al utilizar un esquema de protección contra copia, habrá que tener esto en cuenta.

No se puede dar ningún consejo que vaya a ser particularmente útil a la hora de confeccionar uno de estos esquemas; lo único que cabe decir es que la variedad y el ingenio son las claves de una protección de copias adecuada.

El teclado, conceptos básicos



- Operación del teclado 144
 - Comunicación con la ROM-BIOS 146
 - Traducción de códigos de exploración 147
 - Introducción directa de códigos ASCII 150
- Formato de los datos del teclado 150
 - Las teclas ASCII 151
 - Las teclas especiales 151
- Control del teclado 153
 - Los bytes de estado 153
- Comentarios 156
- Diferencias con el PCjr 157
 - Operación del teclado del PCjr 158
 - Byte de estado del teclado del PCjr 159
- Diferencias con el AT 161

Este capítulo trata principalmente sobre el teclado del IBM PC estándar, aunque a lo largo del texto se han añadido algunos comentarios sobre los teclados ligeramente diferentes del PCjr y el AT. No se tratarán a fondo los modelos peculiares del IBM PC, tales como el PC 3270, el AT y el PCjr, así como de algunos miembros no IBM de la familia PC, porque tienen teclados que no se corresponden exactamente con el teclado del PC estándar; son versiones mejoradas o reducidas del PC estándar. Por ejemplo, IBM cambió de lugar unas pocas teclas en el AT y añadió una nueva tecla y algo de *hardware* adicional, pero afortunadamente no cambió la mayoría de las características de operación. El PCjr tiene menos teclas que el teclado del PC estándar; no obstante, consigue una simulación convincente del teclado estándar. Afortunadamente, esta práctica de simulación del teclado normal del PC parece ser estándar entre los miembros de la familia extendida del PC, haciendo insignificantes las diferencias entre ellos y de poca importancia para los programadores.

La primera parte de este capítulo explica cómo interacciona el teclado con el ordenador a nivel de *hardware* y *software*. En la segunda parte se verá cómo la ROM-BIOS trata la información del teclado y la hace disponible para nuestros programas. ☛ Si piensa jugar con el control del teclado, le aconsejo que considere primero las recomendaciones de la página 156 y no aplique a sus programas la información de este capítulo, a menos que tenga una razón especial para hacerlo. Un ejemplo de la utilización apropiada de la información aquí expuesta es crear un programa que modifique la operación del teclado, tal como lo hace el popular y estimado programa ProKey de mejora del teclado. ☛ Si tiene alguna de tales operaciones en mente, eche un vistazo a los servicios del teclado de la ROM-BIOS en el capítulo 11.

OPERACION DEL TECLADO

El teclado del PC contiene el controlador de teclado 8048, que realiza gran variedad de trabajos, todos los cuales ayudan al aumento de la eficacia del sistema. La principal función del 8048 es sondear las teclas e informar a la ROM-BIOS, siempre que una tecla sea presionada, o soltada. Si alguna tecla permanece presionada durante un tiempo superior a medio segundo, el 8048 envía la acción repetida a intervalos específicos. El controlador del 8048 también tiene capacidades limitadas de diagnóstico y chequeo de errores, y tiene un *buffer* que puede almacenar 20 acciones de tecla para el caso de

Programas de mejora del teclado

Gracias al diseño flexible del *software* del PC, es posible crear programas que manipulan el teclado. Hay disponibles algunos programas de este tipo, pero el mejor conocido y más popular es el ProKey.

Los programas de mejora del teclado monitorizan los datos que vienen del teclado y los cambian de la forma que quieren. Típicamente, estos programas están repletos de instrucciones, llamadas macros de teclado, que les dicen qué pulsación de tecla buscar y qué cambios hacer. El cambio debe implicar la supresión de la pulsación de una tecla (actuando como si nunca hubiera ocurrido), y la sustitución de una pulsación de tecla por otra, o por una larga serie de pulsaciones de tecla. El uso más común de los macros del teclado es abreviar las frases que se introducen comúnmente; por ejemplo, es necesario instruir al ProKey para convertir una combinación de teclas, tal como Alt-S, en la salutación que utilizamos en nuestra correspondencia, tal como *sinceramente suyo*. Es posible utilizar los macros de teclado también para abreviar los comandos de programa, de forma que un comando con tres o cuatro pulsaciones de tecla pueda condensarse en una simple pulsación de tecla.

Los mejoradores de teclado trabajan combinando las potencias de dos recursos especiales: uno que es parte del DOS y otro que es parte de la ROM-BIOS del PC. El recurso del DOS permite al programa de mejora permanecer residente en la memoria del ordenador, mientras el control ordinario del ordenador es transferido a un programa convencional, tal como un proceso de texto. El recurso de la ROM-BIOS hace posible que los programas desvíen el flujo de información del teclado, de forma que pueda ser inspeccionado y cambiado antes de que pase al programa. Un programa como el ProKey utiliza el recurso de «programa residente» del DOS para permanecer activo en la memoria, mientras otros programas están funcionando; entonces utiliza el recurso de monitorización del teclado de la ROM-BIOS para preexaminar los datos del teclado y cambiarlos cuando sea necesario.

que el ordenador principal no pueda aceptarlas (esto raramente ocurre). **AT** El modelo AT utiliza un *chip* controlador de teclado diferente, el 8042, pero realiza esencialmente las mismas funciones que el 8048.

Cada vez que se pulsa o libera una de las teclas en el teclado del PC, los circuitos del teclado generan un número de 1 byte, llamado código de exploración de teclado (*scan code*), que identifica de forma única la tecla afectada. El teclado produce un código de exploración diferente para cada presión y liberación de tecla. Siempre que se pulse una tecla, el byte de código de exploración contiene un número en el rango de 1 al 83 (en el teclado del PC estándar). Cuando liberamos la misma tecla, el teclado genera un código de

exploración 128 (80 hex) unidades superior al código de exploración de pulsación de tecla, poniendo el bit 7 del byte del código de búsqueda a 1. Por ejemplo, cuando se pulsa la letra Z, el teclado genera un código de búsqueda 44; cuando se libera el teclado genera un código de búsqueda 172 (44 + 128). El diagrama del teclado de la figura 6-1 muestra las teclas del teclado estándar sus códigos de exploración asociados.

Cuando escribimos, el teclado no conoce el significado de las pulsaciones de las teclas: meramente informa de las acciones que tienen lugar. Es labor de las rutinas de teclado ROM-BIOS el traducir las acciones sobre las teclas en información con significado utilizable por los programas. Como veremos, el teclado se comunica con el ROM-BIOS mediante puertos e interrupciones.

Comunicación con la ROM-BIOS

Cada vez que cualquier tecla del teclado del PC es pulsada o soltada, la acción es comunicada a la ROM-BIOS del PC como una interrupción 9, la interrupción de acción del teclado. La interrupción 9 llama a una subrutina de manipulación de interrupciones, que responde leyendo el puerto 96 (60 hex) para buscar qué acción de tecla tuvo lugar. El código de exploración que estaba esperando es entregado a la BIOS, donde las rutinas de servicio de teclado lo traducen a un código de 2 bytes. El byte de menor peso de este código generalmente contiene el valor ASCII de la tecla, y el byte de mayor peso contiene generalmente el código de exploración de teclado. Las teclas especiales, tales como las teclas de función y las del teclado numérico, contienen un cero en el byte de menor peso y el código de

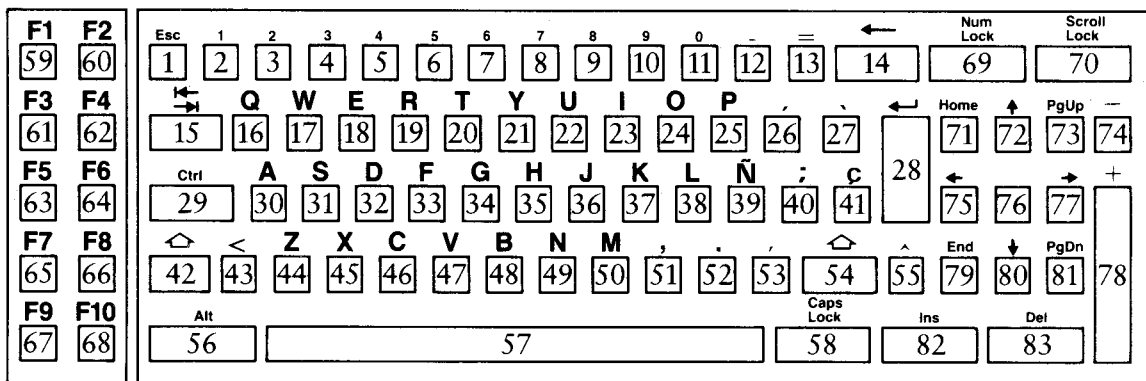


Figura 6-1. Disposición y códigos de registro del teclado estándar del PC

exploración del teclado en el byte de mayor peso (■ más sobre esto en la página 151).

Las rutinas BIOS sitúan entonces los códigos traducidos en una cola, que se guarda a partir de la posición 0000:041E de memoria baja. Los códigos se almacenan aquí hasta que son solicitados por un programa, tal como el DOS o BASIC, que espere leer una entrada del teclado.

Traducción de códigos de exploración

La tarea de traducción de códigos de exploración es moderadamente complicada, porque el teclado IBM tiene varias opciones de cambio que pueden modificar el significado de la pulsación de una tecla. Si pulsamos la tecla Shift y una c, obtenemos una C mayúscula; si pulsamos la tecla Ctrl y una c, generamos la señal Ctrl-C o «break». Estos son ejemplos de diferentes estados de cambio. Podemos modificar el estado de cambio mientras escribimos pulsando la tecla Shift, la tecla Alt o la tecla Ctrl. Cuando una de estas teclas se pulsa y no es soltada, la ROM-BIOS reconoce que todas las siguientes acciones sobre el teclado estarán influidas por el estado de cambio.

Las teclas de cambio permanente (toggle)

En adición a las teclas normales Shift, Ctrl, Alt, hay dos teclas de cambio permanente que también afectan al mecanismo de cambio del teclado: la tecla Caps Lock y la tecla Num Lock. Cuando se activa Caps Lock, cambia el significado de la tecla Shift para las teclas alfabéticas, pero no para el resto de las teclas. La tecla Num Lock cambia entre números y funciones de control del cursor en el teclado numérico.

La información de estado de las teclas de cambio o las teclas de cambio permanente es guardada por la ROM-BIOS en posiciones de memoria baja (hex 417 y y 418), donde podemos usarla o cambiarla. Cuando pulsamos una tecla de cambio o una tecla de cambio permanente, la ROM-BIOS ajusta un bit específico en uno de estos dos bytes. Tan pronto como la ROM-BIOS recibe el código de exploración de liberación de una tecla de cambio, conmuta el bit de estado devolviéndolo a su estado de cambio habitual.

Siempre que la ROM-BIOS recibe un código de exploración de una pulsación ordinaria del teclado, tal como la letra z o la flecha derecha, primero comprueba el estado de cambio, después traduce la tecla en el código de dos bytes apropiado. (■ Trataremos con más detalle los bytes de estado en la página 153.)

Las teclas de combinación

Mientras la rutina de la ROM-BIOS está traduciendo los códigos de registro, constantemente comprueba ciertas combinaciones de tecla de cambio; específicamente, las combinaciones Ctrl-Alt-Del, Shift-PrtSc, Ctrl-Num Lock y Ctrl-Break. Estas cuatro acciones de tecla, semicomandos, obligan a la ROM-BIOS a actuar inmediatamente y realizar una tarea específica, en vez de almacenar los caracteres.

Ctrl-Alt-Del obliga al ordenador a relanzarse, o recargar el programa de comandos. Ctrl-Alt-Del es probablemente usado más a menudo que cualquier otra combinación de teclas. Funciona con seguridad mientras funcione el servicio de interrupción de teclado. Si el servicio de interrupción no está trabajando, hay dos razones posibles: o el vector de interrupción de teclado (entre las posiciones de memoria hex 36 y 39) ha sido cambiado, o se ha ejecutado una instrucción de deshabilitación de interrupciones (CLI). En cualquiera de estos casos, el único recurso que queda es cortar la alimentación, esperar unos segundos y volver a conectar; el programa de arranque reajusta todos los vectores de interrupción y los servicios.

JR



□ *NOTA: Algunos programas pueden dejar deshabilitadas las interrupciones por error. Esto no es posible en el PCjr, puesto que la interrupción del teclado es una interrupción no enmascarable (NMI).*

Shift-PrtSc transfiere el contenido de la pantalla al dispositivo de impresión estándar. La operación se realiza a un nivel primitivo del BIOS a través de la interrupción 5. Para redirigir la salida de la impresora a dispositivos diferentes (que no es una cosa que se haga normalmente), debe cambiarse el vector de interrupción PrtSc para que apunte a una nueva subrutina. La rutina GRAPHICS.COM del DOS 2.00 y de las versiones subsiguientes intercepta la operación PrtSc chequeando primero el modo de video en que se encuentra. Si está en modo gráfico, una rutina toma el control y envía los datos de la pantalla, *pixel a pixel*, a una impresora gráfica compatible IBM (si está conectada). En caso contrario, se llama a la operación convencional y la información es enviada carácter a carácter.

Ctrl-Num Lock suspende la operación del programa hasta que ocurra otra pulsación de una tecla.

Ctrl-Break causa que el ordenador emita una señal *break* generando la interrupción 27. Si nuestros programas han establecido una nueva gestión de la interrupción 27, pueden interceptar la interrupción de *break* y actuar (o ignorarla) de acuerdo a los requerimientos del programa. Si nuestros programas no cambian la rutina de

interrupción, el DOS utiliza su rutina por defecto, y detiene el programa.

Estas son las únicas combinaciones de teclas que son especialmente significativas para la ROM-BIOS. Cuando se envía desde el teclado una combinación inválida, la ROM-BIOS simplemente la ignora y continúa con la siguiente pulsación de tecla.

Hay dos cosas más sobre el teclado del PC que necesitan discutirse antes de pasar a los detalles de la codificación del teclado: la acción de repetición de tecla y las teclas duplicadas.

Acción de repetición de tecla

El teclado del PC realiza la acción de repetición automática de tecla, un proceso que IBM llama *typematic*. La circuitería del interior del teclado del PC analiza el tiempo que se presiona cada tecla, y si una tecla es presionada más de medio segundo, genera automáticamente la acción de repetición de tecla diez veces por segundo. La acción *typematic* se transmite como una sucesión de códigos de exploración de tecla pulsada sin la intervención de los códigos de tecla liberada. Esto hace posible que el hábil programa de tratamiento de la interrupción 9 distinga entre la pulsación de una tecla y la acción *typematic*. Sin embargo, la ROM-BIOS no distingue siempre entre las dos. La rutina de gestión del teclado de la ROM-BIOS trata cada acción de repetición automática de tecla como si la tecla hubiera sido presionada cada vez, e interpreta la clave de forma adecuada. Por ejemplo, si se pulsa y se mantiene la tecla A el tiempo suficiente para que el teclado empiece a generar sucesivas señales de tecla pulsada, la ROM-BIOS creará una serie de Aes que serán pasadas a cualquier programa que esté leyendo datos del teclado. De otra forma, si se pulsa y mantiene la tecla shift —como se hace a menudo—, la ROM-BIOS reconocerá la primera señal de presión de la tecla shift y posicionará el teclado en estado shift, pero ignorará las siguientes señales de pulsación de la tecla shift generadas por el mecanismo de repetir, hasta que obtenga una señal de tecla shift liberada. Todo esto se condensa en el simple hecho de que la ROM-BIOS trata las acciones de repetición de tecla de una forma inteligente, activándolas o ignorándolas cuando lo necesita.

Teclas duplicadas

Otra cosa que deberá tenerse en cuenta es que hay teclas duplicadas en el teclado. Hay, por ejemplo, duplicados puntos, signos más, signos menos y dígitos (0 al 9), y dos teclas shift aparentemente idénticas.

La ROM-BIOS, de forma bastante inteligente, traduce estas teclas duplicadas en los mismos códigos de carácter; por ejemplo, ambas teclas más proporcionan el carácter más, CHR (43). La ROM-BIOS informa también a nuestro programa de las diferencias entre ellos, en caso de que ocurran. Las teclas de caracteres duplicados retienen sus códigos de exploración en el byte de orden alto; nuestros programas sólo necesitan chequear el código de exploración en este byte para ver qué tecla fue pulsada. En cuanto a las dos teclas de shift, cada una pone un bit diferente en el byte de estado del shift (localización 417 hex). Si se desea que los programas reconozcan qué tecla shift fue presionada, es necesario analizar el valor del bit apropiado. (☛ Véase el tema de la localización 417 hex en las páginas 60 y 153.)

Generalmente, lo mejor para nuestros programas es ignorar la distinción entre las teclas duplicadas, aunque algunos de los programas más sofisticados hacen uso de esta información con propósitos especiales. Entre ellos cabe destacar el Microsoft's Fligh Simulator y el Framework de Ashon-Tate.

Introducción directa de códigos ASCII

Debería mencionarse que el teclado del PC, en función con la ROM-BIOS, nos proporciona una forma alternativa de introducir casi todos los códigos de los caracteres ASCII. Esto se hace manteniendo pulsada la tecla Alt e introduciendo entonces el código decimal del carácter ASCII desde el teclado numérico del lado derecho del teclado. Este método permite que se introduzca cualquier código ASCII, del CHR\$(1) al CHR\$(255). El único código ASCII que no puede ser tecleado directamente es el CHR\$(0), porque está reservado para señalar los caracteres no ASCII, tales como las teclas de control del cursor y de función. ☛ En la próxima sección se verá esto con más detalle.

FORMATO DE LOS DATOS DEL TECLADO

Una vez ha sido interpretada una acción del teclado, se almacena como un par de bytes en el *buffer* de la ROM-BIOS. Al byte de menor orden se le llamará *byte principal* y al byte de mayor orden *byte auxiliar*.

Las teclas ASCII

Cuando el byte principal tiene un valor de carácter ASCII, del CHR\$(1) al CHR\$(255), se sabe que fue pulsado uno de los caracteres

del teclado estándar, o que fue introducido un carácter ASCII extendido, utilizando el método Alt-número mencionado anteriormente. (☛ Véase apéndice C para revisar el conjunto completo de caracteres ASCII.) En los caracteres ASCII estándar, el byte auxiliar contiene el código de exploración del teclado de la tecla pulsada. En circunstancias ordinarias, el código de exploración no tiene ningún uso (la función INKEY\$ del BASIC no informa del byte auxiliar). Sin embargo, puede utilizarse el byte auxiliar para distinguir entre caracteres del teclado duplicados con diferentes códigos de exploración. Cuando los caracteres ASCII han sido introducidos «artificialmente» mediante el método Alt-número, el código de búsqueda en el byte auxiliar es cero.

Las teclas especiales

Cuando el byte principal es cero (CHR\$(0)), significa que está siendo pulsada una tecla especial. Las teclas especiales incluyen las teclas de función, las teclas de función con shift, las teclas de control del cursor, tales como *home* y *end*, y algunas de las combinaciones de las teclas Ctrl y Alt. Cuando se pulsa cualquiera de estas teclas solas, o en combinación con otras teclas, el byte auxiliar contiene un valor que representa la tecla. Esto permite definir códigos propios de teclas especiales, sin interferir con los caracteres ASCII extendidos (CHR\$(128) al CHR\$(255)). Los 97 valores de teclas especiales están ordenados en la figura 6-2, en una mezcla de orden lógico y numérico.

☐ *NOTA: Puesto que un valor cero en el byte principal señala una tecla especial, el carácter ASCII cero, CHR\$(0), no puede ser gestionado. Por definición de IBM, CHR\$(0) puede introducirse desde el teclado como Alt-2. Esta combinación de teclas se reporta como tecla especial de valor 3 (byte principal=0, byte auxiliar=3). Nuestros programas pueden interpretar el valor del byte auxiliar de 3 como CHR\$(0), pero ningún lenguaje de programación, o programa, según mi experiencia, se preocupa de hacer esta interpretación.*

Los códigos del conjunto completo de caracteres de las teclas especiales son generados por la ROM-BIOS, pero los diferentes lenguajes de programación varían en la forma en que manejan los códigos. El BASIC, por ejemplo, toma una aproximación mezclada de las teclas especiales. Cuando se utilizan instrucciones ordinarias de entrada, el BASIC entrega los caracteres ASCII regulares al BIOS y filtra las teclas especiales. Alguna de estas teclas puede ser maneja-

Valor del byte auxiliar (dec)	Teclas pulsadas	Valor del byte auxiliar (dec)	Teclas pulsadas	Valor del byte auxiliar (dec)	Teclas pulsadas
59	F1	110	Alt-F7	44	Alt-Z
60	F2	111	Alt-F8	45	Alt-X
61	F3	112	Alt-F9	46	Alt-C
62	F4	113	Alt-F10	47	Alt-V
63	F5	120	Alt-1	48	Alt-B
64	F6	121	Alt-2	49	Alt-N
65	F7	122	Alt-3	50	Alt-M
66	F8	123	Alt-4	3	Sería el
67	F9	124	Alt-5		carácter nulo
68	F10	125	Alt-6		CHR\$(0); véase
84	Shift-F1	126	Alt-7		la nota de la
85	Shift-F2	127	Alt-8		página 151
86	Shift-F3	128	Alt-9	15	Reserve Tab
87	Shift-F4	129	Alt-0		(Shift-Tab)
88	Shift-F5	130	Alt-Guión	71	Home
89	Shift-F6	131	Alt- =	72	Flecha arriba
90	Shift-F7	16	Alt-Q	73	PgUp
91	Shift-F8	17	Alt-W	75	Flecha izquierda
92	Shift-F9	18	Alt-E	77	Flecha izquierda
93	Shift-F10	19	Alt-R	79	End
94	Ctrl-F1	20	Alt-T	80	Flecha abajo
95	Ctrl-F2	21	Alt-Y	81	PgDn
96	Ctrl-F3	22	Alt-U	82	Insert
97	Ctrl-F4	23	Alt-I	83	Delete
98	Ctrl-F5	24	Alt-0	114	Eco (Ctrl-PrtSc)
99	Ctrl-F6	25	Alt-P	115	Ctrl-flecha
100	Ctrl-F7				izquierda
101	Ctrl-F8	30	Alt-A	116	Ctrl-Derecha
102	Ctrl-F9	32	Alt-S		flecha
103	Ctrl-F10	32	Alt-D	117	Ctrl-End
104	Alt-F1	33	Alt-F	118	Ctrl-PgDn
105	Alt-F2	34	Alt-G	119	Ctrl-Home
106	Alt-F3	35	Alt-H	132	Ctrl-PgUp
107	Alt-F4	36	Alt-J		
108	Alt-F5	37	Alt-K		
109	Alt-F6	38	Alt-L		

Figura 6-2. Valor del byte auxiliar de las 97 teclas especiales del teclado estándar del IBM PC. El valor del byte principal es siempre 0

da con la instrucción ON KEY, pero se puede utilizar la función INKEY\$ del BASIC para obtener directamente la codificación de la ROM-BIOS para los caracteres del teclado y encontrar inmediatamente qué tecla especial fue presionada. Si la función INKEY\$ devuelve una cadena de 1 byte, está gestionando un carácter del teclado ASCII ordinario, o extendido. Si INKEY\$ devuelve una cadena de 2 bytes, el primer byte de la cadena es el byte principal de la ROM-BIOS y será siempre CHR\$(0); el segundo byte es el byte auxiliar, e indicará qué tecla especial fue presionada.

CONTROL DEL TECLADO

La operación del teclado y la recogida de datos del teclado que está supervisada por la ROM-BIOS hacen uso de un área de datos localizada en las posiciones bajas de la memoria, de la 417 hex a la 472 hex, así como las posiciones 412 y 488. Nuestros programas pueden hacer uso de estas posiciones para chequear el estado del teclado, o modificar la operación del teclado. Ahora se discutirán las posiciones cuya lectura puede ser útil para nuestros programas y las posiciones cuyo valor podemos cambiar con seguridad.

Los bytes de estado

Se comenzará con los dos bytes de estado del teclado estándar, posiciones 417 (mostrada en la figura 6-3) y 418 hex (mostrada en la figura 6-4). Estos bytes de estado están formados por bits significativos individualmente, que indican qué teclas de cambio y de cambio permanente están activas. Todos los modelos estándar de la familia PC tienen estos dos bytes. **JR** Actualmente, la única diferencia individual entre los modelos es el bit 2 del byte 2. Este bit, llamado bit de *click*, está implementado únicamente en el PCjr. Los otros aspectos del formato de los bits son comunes a todos los modelos PC estándar.

Estado de inserción

La ROM-BIOS analiza y almacena el estado de inserción en el bit 7 del byte 1. Todos los programas que conozco ignoran este bit y guardan su propio registro de si el estado de inserción está o no habilitado; así, aunque es posible, no es una práctica estándar el utilizar el bit de estado de inserción de la ROM-BIOS en nuestros programas.

Bit								Significado
7	6	5	4	3	2	1	0	
X	.	.	.	.	.	.	.	Estado inserción; 1 = activo; 0 = inactivo
.	X	.	.	.	.	.	.	Caps Lock: 1 = activo; 0 = inactivo
.	.	X	.	.	.	.	.	Num Lock: 1 = activo; 0 = inactivo
.	.	.	X	.	.	.	.	Scroll Lock: 1 = activo; 0 = inactivo
.	.	.	.	X	.	.	.	Alt shift: 1 = activo (Alt pulsada); 0 = inactivo
.	.	.	.	.	X	.	.	Ctrl shift: 1 = activo (Ctrl pulsada); 0 = inactivo
.	.	.	.	.	.	X	.	Shift normal: 1 = activo (Shift izquierdo presionado); 0 = inactivo
.	.	.	.	.	.	.	X	Shift normal: 1 = activo (Shift derecho presionado); 0 = inactivo

Figura 6-3. Codificación del primer byte de estado del teclado, en la localización 417 hex

Bit								Significado
7	6	5	4	3	2	1	0	
X	.	.	.	.	.	.	.	1 = Ins presionada
.	X	.	.	.	.	.	.	1 = Caps Lock presionada
.	.	X	.	.	.	.	.	1 = Num Lock presionada
.	.	.	X	.	.	.	.	1 = Estado de bloqueo activo (Ctrl-Num Lock)
.	.	.	.	X	.	.	.	1 = Sonido del teclado del PCjr activo
.	.	.	.	.	X	.	.	1 = <i>Clik</i> activo en PCjr
.	.	.	.	.	.	0	.	No utilizado
.	.	.	.	.	.	.	0	No utilizado

Figura 6-4. Codificación del segundo byte de estado del teclado, en la localización 418 hex

Estado del Caps-Lock

Algunos programadores activan el estado Caps-Lock, poniendo a uno el bit 6 del byte 1. Esto puede confundir o irritar a algunos usuarios del programa, así que no lo recomiendo. Sin embargo, funciona fiablemente y hay abundantes precedentes del uso de esta tarea.

Estado de bloqueo por teclado

El estado de bloqueo por teclado es una interesante característica del PC, aunque no tiene relación práctica con nuestros programas. Como se mencionó antes, una de las combinaciones de teclas especiales que supervisa la BIOS del teclado es el Ctrl-Num Lock. Cuando la BIOS detecta la combinación Ctrl-Num Lock, pasa a un estado conocido como bloqueo por teclado, poniendo a uno el bit 3 del byte de estado 1. Durante el bloqueo por teclado, el BIOS espera hasta que se pulse una tecla imprimible; no devuelve el control del ordenador al programa que esté funcionando hasta que esto ocurra. Esta característica se utiliza para suspender la operación del ordenador.

Durante el bloqueo por teclado, todas las interrupciones son manejadas normalmente. Por ejemplo, si la unidad de disco generó una interrupción (señalando la finalización de una operación de disco), el programa de gestión de la interrupción del disco recibirá la interrupción y la procesará normalmente. Pero, cuando el programa de gestión de la interrupción termina el trabajo, devolverá el control al lugar en que se encontraba cuando tuvo lugar la interrupción, es decir, al bucle de espera del bloqueo por teclado de la BIOS. Así, durante el bloqueo por teclado, el ordenador puede responder a las interrupciones externas, pero los programas están normalmente bloqueados por completo. El BIOS de teclado continúa manejando las interrupciones que señalan acciones de tecla, y cuando detecta una pulsación de tecla normal (por ejemplo, el espacio o una tecla de función, pero no una tecla de shift), termina el bloqueo por teclado, devolviendo finalmente el control al programa que estaba funcionando, permitiendo que continúe.

El estado de bloqueo por teclado no es de uso práctico para nosotros en programación, excepto porque proporciona una forma estándar de que los usuarios de nuestros programas suspendan la operación del mismo.

Debe tenerse en cuenta que el estado de bloqueo por teclado no es «infalible». Es posible que un programa continúe trabajando a través del bloqueo del teclado, actuando sobre una interrupción externa, tal como la interrupción de señal del reloj. Si un programa quiere realmente evitar que sea puesto en bloqueo, podría poner en marcha un programa de gestión de interrupción que trabajase a pesar del estado de bloqueo, o pudiera simplemente desactivar el estado de bloqueo en el caso que estuviese activado.

Estado de teclas de cambio permanente (toggle)

Téngase en cuenta que los bits 4 al 7 de cada byte se refieren a las mismas teclas. En el primer byte, los bits muestran el estado actual de las teclas de cambio permanente; en el segundo byte muestran si la correspondiente tecla está presionada.

Se puede leer el estado de cualquiera de estos bits para saciar la curiosidad, pero pocos, si hay alguno, serán útiles en sus programas. Con la excepción parcial del control del estado Caps-Lock, no creo que sea deseable modificar alguno de los bits de estado de cambio (bits 4 al 6 del byte 1). Y potencialmente es peligroso cambiar algunos de los bits indicativos de que la tecla está pulsada (bits 0 al 3 del byte 1; bits 4 a 7 del byte 2).

COMENTARIOS

Si se desea obtener una mayor profundidad en la comprensión de la operación del teclado del PC, debe estudiarse el listado del programa de la ROM-BIOS en el «Manual de Referencia Técnica de IBM». Cuando se haga esto, debe tenerse cuidado en evitar cometer una falta simple, que es común cuando alguien estudia por primera vez la ROM-BIOS, particularmente las interrupciones utilizadas. La ROM-BIOS trata dos interrupciones diferentes referidas al teclado: una, generada por el teclado (interrupción 9) y que guarda los datos del teclado en el *buffer* de las posiciones bajas de la memoria, y otra, generada para solicitar los servicios de teclado (interrupción 16 hex) y

Interrupción		Origen de la interrupción	Uso
Dec	Hex		
9	9	Teclado	Señales de acción del teclado
22	16	ROM-BIOS	Invoca los servicios estándar del teclado de la BIOS
27	1B	ROM-BIOS	Genera una interrupción cuando se presiona la combinación de la tecla break bajo el control de la BIOS; se invoca una rutina si ha sido creada
35	23	DOS	Si ha sido creada, se invoca una rutina de interrupción al presionar la combinación de la tecla break bajo el control del DOS

Figura 6-5. Interrupciones relativas a la acción del teclado

que pasa los datos del *buffer* al DOS y a nuestros programas. Es muy fácil confundir la operación de estas dos interrupciones, casi tanto como confundir las interrupciones *break*, 27 y 35 (1B y 23 hex). En la tabla de la figura 6-5 listan las interrupciones del teclado.

Un lema general que está presente en todo este libro aconseja no jugar rápido y despreocupadamente, sino jugar según las reglas. Esto significa, otra vez, escribir programas que sean generales a toda la familia IBM PC, más que aprovechar los rasgos de un solo modelo, y escribir programas que utilicen medios oficiales, tales como los servicios de la ROM-BIOS para manipular datos, en lugar de la programación directa del *hardware*. Estas reglas se aplican a la programación del teclado, tanto como se hace con cualquier otro tipo de programación.

DIFERENCIAS CON EL PCjr



El PCjr está diseñado para imitar, tan completamente como sea posible, la operación del PC original de 83 teclas. Pero, como se puede ver en la figura 6-6, el teclado original del Junior tiene sólo 62 teclas, lo que significa que no se corresponde exactamente con el teclado del PC. Este problema ha sido resuelto con una astuta imitación del teclado del PC.

El PCjr tiene 61 teclas en común con el PC, más una nueva tecla, la tecla Fn (función). Cada una de las 22 teclas que faltan es imitada de una u otra forma por varias combinaciones de teclas en el teclado del PCjr. Sin embargo, no es tan fácil como parece, porque las

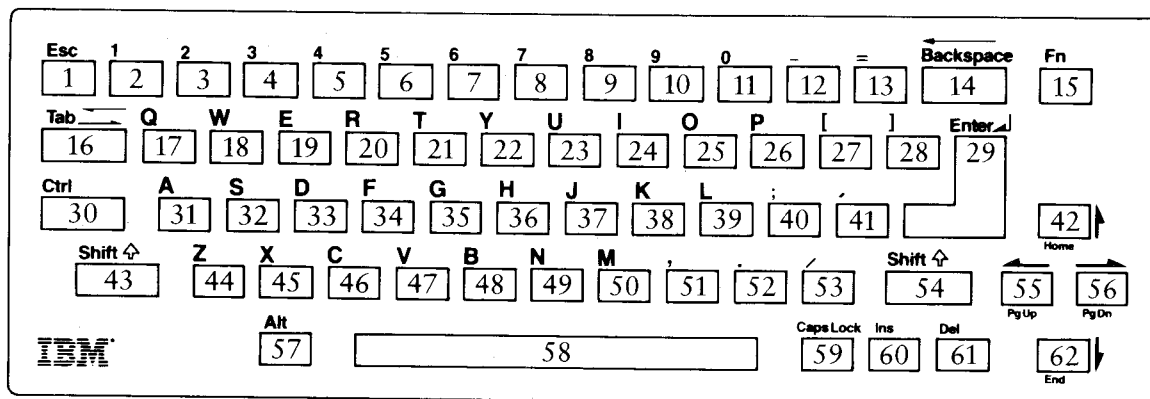


Figura 6-6. Distribución y códigos de exploración de teclado del IBM PC

equivalencias entre el PCjr y el PC no son particularmente directas. (☛ Véase figura 6-7.)

El PCjr tiene también cinco combinaciones de teclas especiales que son únicas, y no tienen equivalente en el PC. Estas están listadas en la figura 6-8. La quinta, Shift-Fn-Esc, se menciona raramente en la documentación del Junior.

Operación del teclado del PCjr

Los estados de operación del teclado del PC que se expusieron en la página 144 son seguidos estrechamente por el PCjr. Sin embargo, puesto que el teclado del Junior es diferente, se ha añadido un bloque de operaciones enteramente nuevo (pero familiar).

Cada acción de una tecla en el PCjr, como las acciones de teclas en el PC, causa una interrupción, pero es una interrupción especial: la

Tecla	Equivalente en el PCjr
F1 al F10	Tecla Fn, seguida por 1 al 0
	Alt-[
/	Alt-/
~	Alt-]
`	Alt-'
PrtSc	Fn-P
* (en la tecla PrtSc)	Alt-(no es la misma que el asterisco Shift-8)
Ctrl-PrtSc (eco)	Fn-E
Num Lock	Alt-Fn-N
Ctrl-Num Lock (pausa)	Fn-Q
Scroll Lock	Fn-S
Break	Fn-B
Home	Fn-flecha arriba
Ctrl-Home	Ctrl-Fn-flecha arriba
PgUp	Fn-flecha izquierda
Ctrl-PgUp	Ctrl-Fn-flecha arriba
5 (en el teclado numérico)	Sin sustitución
End	Fn-flecha abajo
Ctrl-End	Ctrl-Fn-flecha abajo
PgDn	Fn-flecha derecha
Ctrl-PgDn	Ctrl-Fn-flecha derecha
- (en el teclado numérico)	Fn-guión
+ (en el teclado numérico)	Fn- =

Figura 6-7. Teclas del PC equivalentes a las 22 que faltan en el PCjr

Combinación de teclas	Uso
Ctrl-Alt-Izquierda flecha	Desplazamiento de la pantalla de visualización a la izquierda
Ctrl-Alt-Derecha flecha	Desplazamiento de la pantalla de visualización a la derecha
Ctrl-Alt-Caps Lock	Activa y desactiva el sonido del teclado
Ctrl-Alt-Ins	Invoca los programas de diagnósticos
Shift-Fn-Esc	Hace activar las teclas de dígitos como teclas de función

Figura 6-8. Combinaciones de teclas especiales únicas del PCjr

interrupción 2, una interrupción no enmascarable (NMI); ésta llama a la interrupción 48 hex, que traduce los códigos de exploración de las 62 teclas en sus correspondientes códigos de exploración de 83 teclas. La rutina de la interrupción 48 hex genera también una interrupción 9 (simulando una interrupción del teclado del PC), y todo sigue desde ahí tan similar al PC estándar como resulta posible, con la ROM-BIOS trasladando los códigos de acción del PC a sus significados finales.

La razón de que todo este procesamiento sea necesario es porque el teclado del Junior no tiene la ayuda del microprocesador 8048. El teclado del PC es suficientemente inteligente para poder almacenar varias acciones de teclas, lo que permite enmascarar temporalmente las interrupciones del teclado, mientras que otras partes del ordenador necesitan atención. El PCjr no cuenta con esta característica, así que la interrupción por la acción del teclado es más urgente y domina a todas las demás interrupciones.

Una de las mayores diferencias entre el Junior y sus parientes más potentes es que la ROM-BIOS dirige la operación del teclado y de la unidad de disco. Surge un conflicto porque la ROM-BIOS favorece a la unidad de disco, puesto que los discos no tienen acceso directo a memoria. Consecuentemente, si la unidad de disco está en operación, el sistema entero está enmascarado y no puede tener lugar la entrada desde teclado (las comunicaciones asíncronas a través del puerto serie tampoco pueden tener lugar).

Byte de estado del teclado del PCjr

El PCjr tiene un tercer byte de estado del teclado, además de los dos bytes de estado estándar mencionados anteriormente. Está lo-

calizado en la dirección de memoria 488 hex. Este byte es específico de la operación del teclado de 62 teclas del PCjr.

El significado de los bits individuales en el byte de estado se muestra en la figura 6-9. Generalmente no ganará nada leyendo o cambiando estos bits, pero debería saber que se puede suprimir el proceso *typematic* (la operación de repetición de tecla), poniendo el bit 3 a 1. Adicionalmente es posible variar el tiempo de repetición, o el comienzo de repetición, poniendo los bits 2 y 1 a 1. Si se desea experimentar y suprimir la repetición de tecla en un PCjr, se puede insertar el disco de utilidades del DOS y tratar de introducir estos comandos (esto sólo funcionará en un PCjr):

```
DEBUG
F 0:488 L 1 08
```

Recomendaciones de programación del PCjr

El teclado del Junior está diseñado para emular el teclado del PC completo, y está claramente destinado para ser reemplazado, o aumentado, por otros dispositivos de tipo teclado, tal como un ratón, o un teclado completo. Este diseño lo hace particularmente poco capaz para integrar las peculiaridades del teclado del PCjr en nuestros programas.

Sin embargo, y como excepción, recomendaría que sus programas tuviesen en cuenta las peculiaridades del teclado del Junior: cuando esté seleccionando qué teclas especiales utilizar en un programa.

Bit								Significado
7	6	5	4	3	2	1	0	
X	.	.	.	.	.	.	.	Indicador de función
.	X	.	.	.	.	.	.	Señala la acción de la tecla break (Fn-B)
.	.	X	.	.	.	.	.	Función pendiente: Fn presionada
.	.	.	X	.	.	.	.	Enclavamiento en modo de función: hace las teclas numéricas teclas función
.	.	.	.	X	.	.	.	Controla la acción automática de repetición de tecla: 0 = habilitada, 1 = deshabilitada
.	.	.	.	.	X	.	.	Controla la acción de repetición de tecla a velocidad máxima o media: 0 = velocidad media, 1 = velocidad máxima
.	.	.	.	.	.	X	.	Controla el tiempo de espera antes de comienzo de la acción de repetición: 0 = habilitada, 1 = deshabilitada
.	.	.	.	.	.	.	X	Señala que la acción de repetición de tecla va a ser generada

Figura 6-9. Codificación del byte de estado del teclado del PCjr en la localización 488 hex

Algunos programas creados para el PC original tenían el uso de las teclas (especialmente el uso de las teclas de función) estrechamente adaptado al teclado completo del PC. Con la aparición del teclado del PCjr, es prudente considerar de nuevo la selección de las teclas, porque los dos teclados son diferentes. Se debe decidir elegir el mejor uso para el Junior o para el PC, un compromiso entre los dos, o ajustar la operación del programa a la máquina en que se esté trabajando. (● Véase página 67 para ver cómo encontrar el ID de la máquina.) La elección que se haga depende del alcance y del mercado potencial de su programa.

DIFERENCIAS CON EL AT



El teclado del AT también difiere del teclado del PC estándar. En contraste con el teclado del PCjr, las diferencias en el teclado del AT, que son visibles para el usuario y para nuestros programas, son muy ligeras. Sin embargo, por lo que concierne al *hardware* y al BIOS, el teclado del AT tiene en común con el teclado del PCjr, que no funciona como el teclado del PC, pero se ha hecho un mayor esfuerzo por simularlo. Puesto que las diferencias internas en el teclado del AT son esencialmente invisibles para nuestros programas, no será necesario cubrirlas con mucho detalle.

La distribución del teclado del AT es casi la misma que la del teclado del PC original, con unas pocas teclas cambiadas de posición y una tecla añadida. La nueva distribución de alguna de las teclas no requiere cambios en la forma en que se programa la utilización del teclado, o en la selección de teclas a utilizar, pero se observa que una tecla utilizada muy a menudo, la tecla Esc, ha sido desplazada a una localización totalmente nueva, del área superior izquierda al área superior derecha, una molestia para alguien que tenga que utilizar ambos teclados, el del PC y el del AT.

La nueva tecla del AT es la tecla Sys Req. Esta tecla no se utiliza en la operación del programa. En cambio fue creada para activar el cambio entre las tareas del sistema de programación cuando el AT está trabajando en modo multitarea y utiliza las capacidades especiales del microprocesador 286 del AT. Es imposible utilizar esta tecla en nuestros programas del PC.

El *hardware* que une el AT y el teclado del AT es bidireccional, de forma que el teclado pueda enviar información al AT y el AT pueda enviar comandos de vuelta al teclado, incluyendo comandos para activar las luces indicadoras del teclado. Sería disparatado ocuparse de estos comandos de control de teclado.

Es interesante conocer las variaciones francesa, española, alemana, italiana y británica del teclado del AT, pues se introdujeron al mismo tiempo que el orientado a Estados Unidos.

La segunda conjetura sobre los movimientos futuros de IBM es un negocio muy arriesgado, pero mi opinión es que la implantación del teclado del AT (y sus variaciones internacionales) pasarán a los estándares de todos los productos futuros de la familia PC introducidos por IBM. Si esto ocurre o no, no es de mucha importancia aquí, aunque, desde el punto de vista de la programación, el teclado del AT no es verdaderamente diferente del teclado del PC (por el contrario que el teclado del PCjr, que tiene verdaderamente algunas diferencias prácticas).

La generación del sonido



La física del sonido 164

Cómo produce el sonido el ordenador 166

Control del sonido con el temporizador 166

Control directo del altavoz 169

Volumen del altavoz y calidad de sonido 171

El sonido y el PCjr 171

El *chip* de sonido de TI 172

Control del *chip* de sonido 174

Salida de sonido del PCjr 177

Todos los miembros estándar de la familia PC pueden crear efectos sonoros simples, utilizando el temporizador programable del ordenador (el 8253-5) y el altavoz interno del ordenador. **JR** El PCjr también cuenta con multitud de posibilidades en cuanto a sonidos, entre los que destacan un *chip* de generación de sonido especial, y fuentes y salidas sonoras adicionales. Puesto que estas características son exclusivas del PCjr, se comentarán brevemente en la última parte del capítulo, reservando la primera parte para las características sonoras que son universales en la familia PC.

Para comprender cómo se pueden producir sonidos en un ordenador es necesario conocer algunos de los principios básicos del sonido, que se detallarán aquí.

LA FISICA DEL SONIDO

Los sonidos son, simplemente, pulsos o vibraciones regulares del aire. El sonido se produce cuando las partículas del aire son puestas en movimiento por una fuente sonora. Cuando la fuente sonora empuja hacia afuera, comprime las partículas de aire a su alrededor. Cuando tira hacia dentro, la presión liberada arrastra las partículas con ella. Una vibración compuesta por ambas acciones, presión y extracción, hace que las partículas de aire golpeen unas contra otras. Este movimiento desencadena una serie de reacciones que lleva a la vibración a través del aire, desde la fuente original. Tal movimiento se denomina onda de sonido.

El altavoz en los IBM PC se hace vibrar mediante impulsos eléctricos enviados a él por el ordenador. Puesto que los ordenadores trabajan normalmente con dígitos binarios, los voltajes que producen son alto o bajo. Cada transición de un estado de voltaje a otro empuja el cono del altavoz, o lo relaja. Se produce un sonido cuando el voltaje del altavoz pasa de bajo a alto y a bajo otra vez, haciendo que el altavoz se mueva hacia fuera y hacia dentro. Esta simple vibración, consistente en un pulso hacia fuera y un pulso hacia dentro, se llama ciclo, y se mide en hercios (un hercio es simplemente un ciclo por segundo). A través del altavoz del PC, un ciclo aislado de sonido se oye como un *click*. El sonido continuo se produce cuando se envía al altavoz un gran número de ciclos por segundo. Al incrementarse los ciclos por segundo, los *clicks* se funden entre ellos y se convierten en un tono de cierta frecuencia. Por ejemplo, si se pulsa el altavoz hacia dentro y hacia fuera 523 veces en un segundo (esto es, a una velocidad de 523 hercios), se oirá la nota musical conocida como Do control.

Nota	Frecuencia	Nota	Frecuencia	Nota	Frecuencia	Nota	Frecuencia
C ₀	16.35	C ₂	65.41	C ₄	261.63	C ₆	1046.50
C# ₀	17.32	C# ₂	69.30	C# ₄	277.18	C# ₆	1108.73
D ₀	18.35	D ₂	73.42	D ₄	293.66	D ₆	1174.66
D# ₀	19.45	D# ₂	77.78	D# ₄	311.13	D# ₆	1244.51
E ₀	20.60	E ₂	82.41	E ₄	329.63	E ₆	1328.51
F ₀	21.83	F ₂	87.31	F ₄	349.23	F ₆	1396.91
F# ₀	23.12	F# ₂	92.50	F# ₄	369.99	F# ₆	1661.22
G ₀	24.50	G ₂	98.00	G ₄	392.00	G ₆	1567.98
G# ₀	25.96	G# ₂	103.83	G# ₄	415.30	G# ₆	1661.22
A ₀	27.50	A ₂	110.00	A ₄	440.00	A ₆	1760.00
A# ₀	29.14	A# ₂	116.54	A# ₄	446.16	A# ₆	1864.66
B ₀	30.87	B ₂	123.47	B ₄	493.88	B ₆	1975.53
C ₁	32.70	C ₃	130.81	C ₅	523.25	C ₇	2093.00
C# ₁	34.65	C# ₃	138.59	C# ₅	554.37	C# ₇	2217.46
D ₁	36.71	D ₃	146.83	D ₅	587.33	D ₇	2349.32
D# ₁	38.89	D# ₃	155.56	D# ₅	622.25	D# ₇	2489.02
E ₁	41.20	E ₃	164.81	E ₅	659.26	E ₇	2637.02
F ₁	43.65	F ₃	174.61	F ₅	698.46	F ₇	2793.83
F# ₁	46.25	F# ₃	185.00	F# ₅	739.99	F# ₇	2959.96
G ₁	49.00	G ₃	196.00	G ₅	783.99	G ₇	3135.96
G# ₁	51.91	G# ₃	207.65	G# ₅	830.61	G# ₇	3322.44
A ₁	55.00	A ₃	220.00	A ₅	880.00	A ₇	3520.00
A# ₁	58.27	A# ₃	233.08	A# ₅	932.33	A# ₇	3729.31
B ₁	61.74	B ₃	246.94	B ₅	987.77	B ₇	3951.07
						C ₈	4186.01

Nota: Escala cromática de temperamento igual; A₄ = 440.

Graduación de tonos estándar americanos, adoptado por la Asociación Americana de Estandarización (ASA) en 1936

Figura 7-1. Frecuencias correspondientes a ocho octavas de notas musicales

Una persona media puede oír sonidos en un rango de 20 a 20 000 hercios. El IBM PC puede generar sonidos a través de su altavoz, cuyas frecuencias podrían variar teóricamente en un rango entre 18 y más de un millón de hercios, más allá del rango del oído humano. Para dar una idea de la magnitud de este rango de frecuencia, compárese con el de una voz humana normal que tiene un rango de sólo 125 a 1.000 hercios.

En el altavoz que incorporan los ordenadores personales IBM no tiene control de volumen y su objetivo no es la reproducción fiel de sonido. Como resultado, diferentes frecuencias producirán diferentes efectos; algunos pueden sonar más altos que otros, y algunos pueden tener una graduación de tonos más fiel. Esta fluctuación está producida por el diseño del altavoz, y no es algo que se pueda controlar.

COMO PRODUCE EL SONIDO EL ORDENADOR

Se puede generar sonidos a través del altavoz de dos formas, utilizando dos diferentes fuentes de sonido, o tan sólo una de ellas. Un método consiste en escribir un programa que actúe sobre el altavoz, manipulando los dos bits de altavoz en el *chip* interfaz de periféricos programable (el PPI). Utilizando este método, el programa controla la duración del pulso y la frecuencia sonora resultante. El otro método consiste en utilizar el temporizador programable interno del PC (el 8253-5) para enviar al altavoz una frecuencia precisa. La utilización del temporizador es un método más popular por dos razones: puesto que los pulsos del altavoz están controlados por el temporizador en lugar del programa, la CPU está libre para dedicar su tiempo a las otras demandas del sistema informático, y el temporizador no depende de la velocidad de trabajo de la CPU (que es más rápida en el AT y más lenta en el PCjr).

El método del programa y el método del reloj pueden utilizarse juntos, o separadamente, para crear una gran variedad de sonidos simples y complejos. Se explicará el control de sonido con el temporizador y el control directo del altavoz de forma más completa en las siguientes páginas, y se describirán algunas de las mejoras que tiene el PCjr con respecto a la familia PC.

Control del sonido con el temporizador

El temporizador programable 8253-5 es el corazón de la producción de sonido de los modelos PC estándar —además de ser también el corazón del reloj de tiempo real del sistema—. Aunque se va a estudiar principalmente su utilización como generador de sonido, el 8253-5 se denomina temporizador porque su función primaria es indicar el tiempo (en la misma forma que un metrónomo indica el tiempo a un músico).

Así es como trabaja. El 8253-5 obtiene una señal proporcionada por el reloj principal del ordenador (el 8284A), que oscila a una frecuencia de 1193180 veces en un segundo, o 1,193 megahercios (MHz). El temporizador está programado para producir una interrupción de reloj (interrupción 8) una vez cada 65536 ciclos del reloj principal, o alrededor de 18.2 veces en un segundo.

Esta interrupción de reloj se llama usualmente señal (*tic*) de reloj. Contando los tics del reloj, la ROM-BIOS calcula la hora del día.

La interrupción del tic de reloj de la ROM-BIOS se utiliza, a menudo, por los programas que guardan la hora, aunque algunos programas obvian esta interrupción y trabajan directamente con el

temporizador. Por ejemplo, el BASIC utiliza el temporizador directamente para contar la duración de un sonido, que se mide en tics de reloj. Sin embargo, puesto que la velocidad estándar de 18.2 tics por segundo, a menudo, no es suficientemente rápida para proporcionar la precisión que demandan algunos tipos de música, el BASIC reprograma el temporizador para funcionar cuatro veces más rápido, lo que hace que la interrupción 8 (el tic de reloj) se produzca 72.8 veces por segundo, en lugar de 18.2. Cuando el BASIC cuenta a cuádruple velocidad, es posible reproducir más fielmente el tiempo propio de una pieza de música.

□ *NOTA: El BASIC cuadruplica la velocidad de reloj durante la ejecución del comando de música. Evita la interferencia con la interrupción número 28 del tic de reloj del BIOS, que es vital para algunas otras funciones del sistema, modificando el vector de la interrupción 8 para apuntar a una rutina a la ROM-BIOS cada cuarto de señal. En el cuarto, el controlador de interrupción devuelve momentáneamente el control a la BIOS, habilitándola para incrementar su contador y activando la interrupción 28, después de lo cual se devuelve el control al BASIC.*

Programación del temporizador

La creación de sonidos con el temporizador implica dos pasos básicos: primero, hay que programar el temporizador para generar una frecuencia; después, se debe dirigir la salida del temporizador al altavoz. Estos dos pasos pueden ser realizados separadamente. El sonido se emite cuando han sido realizados los dos pasos, y el sonido termina cuando alguno de los dos pasos acaba.

El reloj puede ser programado para producir pulsos a cualquier frecuencia que queramos, pero, puesto que no se le proporciona información acerca del tiempo que tiene que durar el sonido, éste se producirá indefinidamente, a menos que se desactive. Así pues, los programas deben recibir información a este respecto a través de unas cortas instrucciones de temporización.

Para generar sonidos, el temporizador se programa de la misma forma en que el BASIC lo hace para generar tics, proporcionándole un número. Recibida una orden, el temporizador cuenta los pulsos de reloj del sistema (que oscilan alrededor de los 1,193 MHz) hasta que el total alcanza el número dado. Entonces produce un pulso (en lugar de una interrupción) y comienza a contar otra vez desde cero. En efecto, el temporizador «divide» la frecuencia del reloj entre el número dado para producir una frecuencia de salida. El resultado es

que el temporizador envía una serie de pulsos que producen un sonido de una cierta frecuencia cuando se conecta el altavoz.

La cuenta de control y la frecuencia resultante están en relación recíproca, como se muestra en estas fórmulas:

$$\begin{aligned} \text{Cuenta} &= 1193180 / \text{Frecuencia} \\ \text{Frecuencia} &= 1193180 / \text{Cuenta} \end{aligned}$$

De estas fórmulas se aprecia que un sonido de frecuencia baja (tono de graduación baja) está producido por una cuenta alta, y que un sonido de frecuencia alta (tono de graduación alta) está producido por una cuenta baja. Una cuenta de 100 produciría un tono de graduación alta, alrededor de 11931 ciclos por segundo, y una cuenta de 10000 produciría un tono de graduación baja, alrededor de 119 ciclos por segundo.

Se puede producir cualquier frecuencia, con las limitaciones de la aritmética de 16 bits. La frecuencia más baja es 18.2 hercios (con un divisor de 65 535, FFFF hex) y la más alta es 1.193 megahercios (con un divisor de 1). El BASIC almacena esto en un rango práctico de 37 a 32 767 hercios. El programa que se muestra al final de la página demuestra que el rango de frecuencia de altavoz interno es incluso menor que el que proporciona el BASIC.

Una vez se ha calculado la cantidad necesaria para la frecuencia deseada, se envía a los registros del temporizador 8253. Esto se hace con tres puertos de salida. El primer puerto advierte al temporizador que se le envía la cuenta, mandando el valor 182 (B6 hex) al puerto 67 (43 hex). Las dos salidas siguientes envían los bytes de orden bajo y alto de la cuenta, una palabra de 16 bits sin signo, al puerto 66 (42 hex) (el byte de orden bajo seguido por el byte de orden alto). Este programa en BASIC ilustra el proceso:

10 CUENTA = 1193280! / 3000	' nuestra frecuencia es 3000
20 CUENTA.BAJO = CUENTA MOD 256	' calcula el valor del byte menos significativo
30 CUENTA.ALTO = CUENTA / 256	' calcula el valor del byte más significativo
40 OUT 67, 182	' prepara el temporizador
50 OUT 66, CUENTA.BAJO	' carga el byte menos significativo
60 OUT 66, CUENTA.ALTO	' carga el byte más significativo

Activación del altavoz

Una vez se ha programado el reloj, todavía es necesario activar la circuitería del altavoz para utilizar la señal que está generando el temporizador. Como ocurre con la mayoría de las otras partes del PC, el altavoz se manipula enviando ciertos valores a un puerto específico, proceso que se ilustra en la figura 7-2. El altavoz es

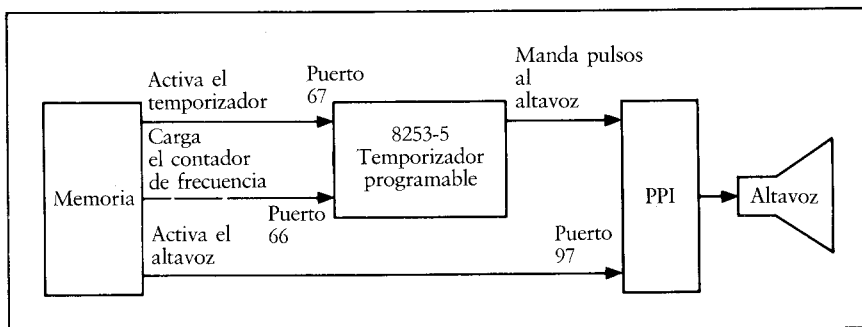


Figura 7-2. Generación de frecuencias con el temporizador 8253-5 y el altavoz

controlado por el interfaz de periféricos programable (PPI) y utiliza el puerto 97 (61 hex). Sólo dos de los ocho bits del puerto son utilizados por el altavoz: los bits de orden más bajos numerados con 0 y 1. Los otros bits se utilizan para otros propósitos, por lo que es importante no modificarlos mientras se trabaja con el altavoz.

El bit más bajo, bit 0, controla una señal de reloj utilizada para gestionar el altavoz. El segundo bit, bit 1, controla los pulsos del altavoz. Ambos bits deben de ser activados para hacer que el altavoz responda al temporizador. Podemos desactivarlos sin modificar los bits que no corresponden al altavoz, con una operación como ésta:

```

70 PUERTO.ANTIGUO = INP (97) ' asigna el valor del puerto 97 a PUERTO.ANTIGUO
80 PUERTO.NUEVO = ' activa los bits 0 y 1
  (PUERTO.ANTIGUO OR &H03)
90 OUT 97, PUERTO.NUEVO ' activa el altavoz
  
```

Control directo del altavoz

El temporizador controla al altavoz enviando señales periódicas que mueven el cono del altavoz hacia adentro y hacia afuera. Se puede conseguir lo mismo con un programa que envíe señales de activación y desactivación directamente al altavoz. Esto se logra poniendo el bit 0 de la puerta 97 (61 hex) a cero, para desactivar el altavoz, y entonces poniendo alternativamente el bit 1 a 0 y a 1, lo cual hace pulsar al altavoz. Cuando se utiliza ese método, la temporización del programa determina la frecuencia del sonido (cuanto más rápido se ejecuta el programa, más alto es el grado del tono). Este programa en BASIC demuestra cómo se hace (en el ejemplo, se supone que el puerto 97 (61 hex) tiene un valor de 76):

```
10 X = INP (97) AND &HFC ' cambia el valor del puerto, desactiva los últimos 2 bits
20 OUT 97, X               ' empuja el cono del altavoz
30 OUT 97, X + 2           ' relaja el cono del altavoz
40 GOTO 20
```

Las dos acciones de las líneas 20 y 30 activan y desactivan el altavoz. Cada una es medio ciclo, y las dos juntas producen un ciclo de sonido completo.

Este ejemplo se ejecuta con la máxima rapidez con la que el BASIC pueda procesarlo, produciendo una nota tan alta como sea posible. Si fuese necesario un rango mayor en una determinada aplicación, se utilizaría, probablemente, un lenguaje más rápido, y se introducirían, deliberadamente, tiempos de espera iguales a la mitad del tiempo de ciclo entre cada ciclo completo (mitad del tiempo del ciclo, porque cada operación de activación o desactivación es medio ciclo). Con cualquier lenguaje que se utilice, hay que incluir una cuenta de duración del sonido. Para producir diferentes sonidos de una frecuencia particular, tales como sonidos de tictac o zumbidos, no habrá más que variar los tiempos de espera entre pulsos.

A pesar de todas estas maravillosas posibilidades y la generación de sonidos a través del altavoz, o mediante una acción de programa directa, no es una buena forma de producir sonidos. La utilización del programa directo presenta tres grandes desventajas:

- Un programa requiere la atención constante de la CPU, así que el ordenador no tiene tiempo de realizar cualquier otro trabajo.
- La frecuencia está a merced de la velocidad del ordenador; esto es, el mismo programa produciría un sonido más bajo o más alto según el modelo fuese más lento o más rápido.
- Las interrupciones de los tics de reloj interfieren en la fusión del sonido, produciendo trinos. La única forma de evitar esto es suspender los tics de reloj, deshabilitando las interrupciones (y eso hace que el ordenador pierda la noción del tiempo).

La única ventaja reseñable para producir sonidos del método directo sobre el método del temporizador es la siguiente: con el control apropiado sobre los tiempos de espera de los programas es posible conseguir un rico sonido polifónico. Se advierte, sin embargo, que esto requiere una programación muy hábil y tediosa, y, a la postre, puede no haber valido la pena.

VOLUMEN DEL ALTAVOZ Y CALIDAD DE SONIDO

En el altavoz interno del ordenador no existe control del volumen de ningún tipo. Sin embargo —como en todos los altavoces—, puede ocurrir que la respuesta varíe según la frecuencia emitida, y algunas frecuencias pueden sonar más altas que otras. En el caso de un altavoz de poca calidad, como los instalados en la mayoría de los PC, la intensidad varía bastante con la frecuencia. Para comprobarlo se puede utilizar el siguiente programa (puede ser útil para elegir el mejor tono del sonido para un propósito determinado):

10	PLAY "MF"	' ejecuta cada sonido separadamente
20	FRECUENCIA = 37	
30	WHILE FRECUENCIA < 32000	' usa todas las frecuencias hasta 32000 Hz
40	PRINT USING "##,###"; FRECUENCIA	' muestra la frecuencia
50	SOUND FRECUENCIA, 5	' produce el sonido con una duración de 5
60	FRECUENCIA = FRECUENCIA * 1.1	' incrementa la frecuencia en $\frac{1}{10}$
70	WEND	

Se puede apreciar también que los altavoces de los distintos modelos del PC pueden no sonar de forma semejante, en parte porque los materiales de cada alojamiento del sistema resuenan diferentemente. Por ejemplo, el timbre del PCjr es bastante diferente del PC portátil y del de los modelos PC y XT. Estas variaciones de sonido no deberán ser, pues, motivo de sorpresa.

EL SONIDO Y EL PCjr



El PCjr tiene el *chip* temporizador programable estándar y un altavoz incorporado, al igual que el resto de la familia, pero también tiene otras fuentes de sonido y otras salidas para las señales sonoras.

La fuente de sonido mejor conocida del Junior es el *chip* generador de sonido TI SN76496A, una gran adición para los efectos especiales en juegos y aplicaciones educativas. Pero el Junior tiene también dos fuentes de sonido menos conocidas: la entrada del cassette de cinta magnética y la línea de audio (línea B30) del conector del canal de E/S. La selección entre estas cuatro fuentes de sonido está controlada por el interfaz de periféricos programable 8255-5, o PPI (☛ véase capítulo 1). Un programa puede controlar qué fuente se utiliza modificando los bits 5 y 6 del 8255, lo que se realiza a través del puerto 97 (61 hex). (☛ Véase figura 7-3.)

Bit 5	Bit 6	Valor	Fuente de sonido
0	0	0	Temporizador 8253 (fuente de sonido estándar de IBM)
0	1	1	Interfaz de cassette
1	0	2	Canal de E/S
1	1	3	<i>Chip</i> de sonido TI

Figura 7-3. Valores para las cuatro fuentes de sonido del PCjr de los bits 5 y 6 del puerto 97 del 8255-A

Las fuentes de sonido del cassette de cinta magnética y del canal de E/S pueden conectarse a un amplificador estéreo para producir una mejor calidad del sonido que el altavoz interno, pero, puesto que no son muy utilizadas, no serán consideradas en lo que queda de este capítulo. En cambio, sí se va a prestar atención, brevemente, al *chip* de sonido de TI.

El chip de sonido de TI

El *chip* de sonido de TI tiene cuatro generadores de sonido separados o voces. Tres de estas voces son completamente independientes, y generan tonos puros (como hace el temporizador). La cuarta voz es una fuente de «ruido» que genera sonidos de ruido irregular de gran variedad de formas. Las cuatro voces tienen un control de volumen independiente, proporcionando un conjunto, graduado uniformemente, de 15 niveles de volumen, más un volumen cero (desconectado). Cada una de las tres voces puras tiene una frecuencia de selección independiente. La voz de ruido tiene tres frecuencias preseleccionadas y una cuarta opción, que se apropia de la frecuencia de la tercera voz pura: se va a ver cada uno de los elementos de sonido por separado, explicando después cómo son utilizados juntos para programar el *chip* de sonido de TI.

Los generadores de tono

Cada uno de los tres generadores de tono, o voces, está controlado por un número de 10 bits que un determinado programa envía al *chip* de sonido de TI. El *chip* de sonido de TI sigue la misma filosofía para crear frecuencias que el temporizador programable: la frecuencia del reloj del sistema se divide por un número de cuenta, y el cociente determina la frecuencia. Sin embargo, los detalles son

diferentes. Para el temporizador, la cuenta de frecuencia es un número de 16 bits dividido por la frecuencia de reloj del *bus* de 1.193 megahercios. Para el *chip* de sonido de TI, la cuenta es un número de 10 bits dividido entre la treintaidosava parte de la frecuencia de reloj del sistema (3 579 MHz), que resulta ser 111 860 Hz.

La única limitación del número de control de 10 bits es que reduce el número de frecuencias disponibles en un factor de 32. Por ejemplo, si se utiliza una cuenta de 100, el *chip* de sonido de TI produce una frecuencia de 1118.6 Hz, y el siguiente divisor, 101, proporciona 1107.5 Hz; no es posible obtener las frecuencias intermedias. Por contra, el temporizador produciría 32 frecuencias dentro del mismo rango. En la práctica, esta limitación es sólo un problema para los sonidos musicales más delicados, tales como acordes de tres notas que pueden sonar de forma desafinada.

Atenuación

Cada voz del *chip* de sonido de TI tiene un control de nivel de sonido independiente, que está calculado en términos de decibelios de atenuación. Se utilizan cuatro bits para controlar el volumen. Estos bits, etiquetados del A0 al A3, pueden fijarse independientemente, o todos al tiempo, para producir 16 niveles de volumen, como se muestra en la figura 7-4. Cuando uno de los dos bits se pone a 1, el sonido es atenuado en una cantidad específica: 2, 4, 8 ó 16 decibelios. Cuando los cuatro bits se ponen a 1, el sonido desaparece casi completamente. Cuando los cuatro bits se ponen a 0, el sonido se produce en su máximo volumen. Aunque los niveles de sonido pueden calcularse, es más fácil elegir el sonido deseado por simple tanteo.

A	A	A	A	Valor	Atenuación (decibelios)
0	1	2	3		
.	.	.	1	1	2
.	.	1	.	2	4
.	1	.	.	4	8
1	.	.	.	8	16
1	1	1	1	32	Volumen mínimo

Figura 7.4. Especificaciones de los bits de atenuación

El generador de ruido

Hay dos modos para manipular el ruido, además de cuatro selecciones de frecuencia. Una, llamada ruido periódico, produce un sonido continuo; la otra, llamada ruido blanco, produce un sonido como un siseo. Estos dos modos están controlados por un bit conocido como bit FB. Cuando el FB está a 0, se genera el ruido periódico; cuando FB está a 1, se produce el ruido blanco.

Dos bits, conocidos como NF0 y NF1, controlan la frecuencia a que trabaja el generador de ruido. Tres de las cuatro posibles combinaciones de NF0 y NF1 determinan una frecuencia de ruido independiente, basada en el temporizador. La cuarta combinación se apropia de la frecuencia de la tercera de las tres voces puras producidas por los generadores de tono. La figura 7-5 muestra los valores de los bits de ruido y sus frecuencias asociadas.

Control del chip de sonido

Para programar el *chip* de sonido de TI se le transmiten direcciones del registro de 3 bits (que seleccionan las frecuencias de voz, el control de atenuación y el control de la voz de ruido), divisores de frecuencia de 10 bits, posicionamientos de atenuación de 14 bits, una selección de frecuencia de ruido de 2 bits y una selección del tipo de ruido de 1 bit (todos a través del puerto 192 (C0 hex)).

El *chip* de sonido de TI tiene ocho registros que controlan lo que hace. Tres bits, conocidos por los nombres R0 a R2, seleccionan los registros e identifican el parámetro introducido. Los valores del registro se muestran en la figura 7-6.

Toda la programación del *chip* de sonido de TI se lleva a cabo escribiendo en el puerto 192 (C0 hex). No se ha de leer este puerto: si se hace, la máquina quedará bloqueada. Este byte contiene el bit ID

NF0	NF1	Frecuencia del ruido
0	0	$1\ 193\ 180 \div 512 = 2330$
0	1	$1\ 193\ 180 \div 1024 = 1165$
1	0	$1\ 193\ 180 \div 2048 = 583$
1	1	Apropiado de la voz 3

Figura 7-5. Frecuencias del generador de ruido producidas por combinaciones de los bits NF0 y NF1

R0	R1	R2	Parámetro
0	0	0	Número de control de frecuencia de la voz 1 (10 bits)
0	0	1	Atenuación de la voz 1 (4 bits)
0	1	0	Número de control de frecuencia de la voz 2 (10 bits)
0	1	1	Atenuación de la voz 2 (4 bits)
1	0	0	Número de control de frecuencia de la voz 3 (10 bits)
1	0	1	Atenuación de la voz 3 (4 bits)
1	1	0	Control de la voz de ruido (4 bits; utilizados 3)
1	1	1	Atenuación de la voz de ruido (4 bits)

Figura 7-6. Los bits de identificación de parámetros

(identificándolo como el byte de comando), los tres bits de registro (R0 al R2) y algunos bits de datos. El byte de comando puede utilizarse para cargar los bits de control de frecuencia, atenuación, o ruido, junto con los bits de registro. Los formatos de los bits de datos variarán dependiendo de su propósito. En los casos de cuentas de frecuencia de voz pura, este primer byte está seguido por un segundo byte que contiene un bit ID, junto con los seis bits de frecuencia restantes que no pueden ser introducidos en el primer byte. Deliberadamente, por diseño, el *chip* de sonido de TI aceptará el segundo byte, una y otra vez, sin que el primer byte de comando lo preceda cada vez. Esto permite cambios rápidos de frecuencia sin utilizar el programa anterior, que sería necesario para cargar ambos bytes: las figuras 7-7 a la 7-10 muestran los formatos de bits de varias disposiciones de bytes.

❑ **NOTA:** Antes de que intente programar el *chip* TI, hay que advertir una diferencia irritante en los diseños de Texas Instruments e Intel. No utilizan la misma notación de orden de bits. Como resultado, en el «Manual de Referencia Técnica del PCjr», los bits del *chip* de sonido se determinan en un orden inverso al de la notación utilizada en el manual (y en este libro). Por ejemplo, lo que normalmente sería señalado como bit 7 de un byte, se encontrará denominado bit 0 MSB (por el bit más significativo). Si se utiliza el manual de referencia, habrá que seguir la notación MSB/LSB, ignorando la numeración de los bits, y enfadarse por el desconsiderado cambio de notación.

Podría parecer obvio, según este planteamiento, que la programación del *chip* de sonido de TI es engorrosamente compleja. Sin embargo, se reduce, en muchos casos, a contar bits con los dedos de

Bit								Uso
7	6	5	4	3	2	1	0	
1	.	.	.	.	.	.	.	Identifica el primer byte (byte de comando)
.	R0	R1	R2	.	.	.	.	Número de registro en el <i>chip</i> de TI (0, 2 ó 4)
.	.	.	.	F6	F7	F8	F9	4 de los 10 bits de la cuenta de frecuencia

Figura 7-7. Valores de los bits del primer byte de la cuenta de frecuencia

Bit								Uso
7	6	5	4	3	2	1	0	
0	.	.	.	.	.	.	.	Identifica el segundo byte (cuenta completada)
.	X	.	.	.	.	.	.	No utilizado, ignorado; puede ponerse a 0 ó 1
.	.	F0	F1	F2	F3	F4	F5	6 bits restantes de la cuenta de frecuencia

Figura 7-8. Valores de los bits del segundo byte de la cuenta de frecuencia

Bit								Uso
7	6	5	4	3	2	1	0	
1	.	.	.	.	.	.	.	Identifica el primer byte (byte de comando)
.	R0	R1	R3	.	.	.	.	Número de registro del <i>chip</i> de TI (1, 3, 5 ó 7)
.	.	.	.	A0	A1	A2	A3	4 bits de atenuación

Figura 7-9. Codificación de los bits de atenuación del primer byte de la cuenta de frecuencia

Bit								Uso
7	6	5	4	3	2	1	0	
1	.	.	.	.	.	.	.	Identifica el primer byte (byte de comando)
.	1	1	0	.	.	.	.	Número de registro del <i>chip</i> de TI (6)
.	.	.	.	X	.	.	.	No utilizado, ignorado; puede ponerse a 0 ó 1
.	.	.	.	.	FB	.	.	1 para ruido blanco, 0 para periódico
.	.	.	.	.	.	NF0	NF1	2 bits de control de la frecuencia de ruido

Figura 7-10. Codificación de los bits de control de ruido del primer byte de una cuenta de frecuencia

las manos y los pies. El inconveniente consiguiente es que los programas que utilizan el *chip* de sonido pueden ser utilizados satisfactoriamente en el PCjr, y no producirán música en los otros miembros de la familia PC (en definitiva, esto es algo que se deberá tener en cuenta antes de examinar todos los problemas del aprendizaje de la programación del *chip* de sonido).

Salida de sonido del PCjr

Las señales sonoras del PCjr tienen dos destinos: el altavoz interno y una de las salidas externas. Las salidas externas son tres de los zócalos de conexión de la parte de atrás del PCjr: la salida de audio A (que se conecta normalmente a un sistema hi-fi), la salida de transmisión directa D (que puede ser conectada a un monitor RGB, donde la señal de sonido es ignorada normalmente) y la salida de Televisión T (que puede conectarse a una TV a través de un modulador de RF).

El altavoz interno del PCjr, como el resto de altavoces internos del PC, sólo puede recibir sus señales de sonido del temporizador. Las salidas externas reciben su sonido de cualquiera de las cuatro fuentes generadoras. Es importante advertir que el sonido producido por el *chip* de sonido TI no puede ser dirigido al altavoz interno del ordenador. Esto es debido a que el altavoz interno está controlado de una forma completamente incompatible con las ricas capacidades del *chip* de sonido TI —increíble pero cierto.

Al estudiar en el «Manual de Referencia Técnica del PCjr» todo lo relacionado con este tema, se advertirá que puede utilizarse el bit 4 de la puerta 97 para controlar el altavoz interno. Poniendo este bit a 1, se inhabilita el altavoz; pero no debe hacerse. Por una parte, hay otras formas de controlar el altavoz, como se ha visto en el epígrafe anterior. Pero lo que es más importante, este bit tiene un uso radicalmente diferente en los otros modelos del PC: cuando este bit está a uno, se inhabilita el uso de la memoria, lo que bloquea el ordenador completamente.

La ROM-BIOS, conceptos básicos



Filosofía del BIOS 181

Interrupciones de servicios de ROM-BIOS 181

Características operativas de los servicios del BIOS 182

Creación de un interfaz en lenguaje ensamblador 184

Forma básica de una rutina de interfaz 185

El secreto de la afortunada programación de la familia PC se basa en la utilización eficaz del *software* que lleva implementado la máquina: los servicios de la ROM-BIOS. Conceptualmente, los servicios de la ROM-BIOS están intercalados entre el *hardware* y los lenguajes de alto nivel (incluyendo el sistema operativo). Trabajan directamente con el *hardware* del ordenador y los dispositivos periféricos, realizando algunas de las tareas más fundamentales del sistema, tales como leer y escribir bytes individuales de datos en la pantalla, o en el disco. Los servicios de lenguaje de programación y los servicios del DOS se construyen, a menudo, a partir de estas funciones básicas y se mejoran para hacer más eficaces determinados procesos particulares. Podemos mejorar nuestros programas de la misma forma conectándolos directamente con la ROM-BIOS, abriendo el camino de este modo a un conjunto de herramientas extraordinariamente potentes y utilizando nuestros ordenadores de la forma en que IBM aconseja que se utilicen.

Este último punto conviene recalcarlo. IBM se ha esforzado todo lo posible para crear un método claro y bien definido para la dirección de las operaciones del ordenador mediante los servicios de la ROM-BIOS. Como ocurre con cada nuevo modelo de PC que se diseña, IBM (y cualquier otro fabricante de ordenadores que esté incluido fielmente en la familia PC) se asegura de que sus servicios de la ROM-BIOS resulten totalmente compatibles con los de los otros miembros de la familia. Puesto que controlamos nuestros ordenadores mediante la ROM-BIOS, bien directa o indirectamente, estamos protegidos de cualquier problema de compatibilidad. Si pasamos por alto la ROM-BIOS y programamos directamente sobre el *hardware*, no sólo estamos abogando por los problemas, sino que también resultan severamente limitados el rango y la viabilidad de nuestros programas.

En los próximos cinco capítulos veremos las rutinas de servicio de la BIOS. Afortunadamente, las rutinas se dividen, como es natural, en grupos derivados de los dispositivos de *hardware* que llevan, así los servicios de video, servicios de disco y servicios de teclado pueden ser tratados separadamente. Pero antes de profundizar más en los servicios individuales, es necesario ver cómo se pueden incorporar en nuestros programas. Este capítulo asienta nuestro nivel de desarrollo, mediante la explicación de lo que sucede cuando se escribe una rutina de interfaz, el puente entre nuestro lenguaje de programación y los servicios de la ROM-BIOS. Primero, unas palabras sobre cómo opera la ROM-BIOS.

FILOSOFIA DEL BIOS

Todos los servicios de la ROM-BIOS se invocan mediante interrupciones. Las instrucciones de interrupción apuntan a una posición particular en la tabla de vectores de interrupción en las posiciones bajas de la memoria, que contienen un vector de interrupción: la dirección de la rutina de servicio almacenada en ROM. Este diseño hace posible que cualquier programa demande un servicio sin conocer la localización de memoria específica de la rutina de servicio de la ROM-BIOS. Permite también que los servicios sean trasladados, expandidos o adaptados, sin afectar a los programas que utilizan dichos servicios. Aunque IBM ha tratado de mantener las posiciones absolutas de memoria de algunas partes de la ROM-BIOS, sería tonto utilizar estas direcciones, puesto que siempre hay una posibilidad de que se puedan cambiar en el futuro. La forma estándar, preferida, y más fiable de invocar un servicio de la ROM-BIOS, es utilizar su interrupción mejor que su posición absoluta.

Los servicios de la ROM-BIOS podrían ser supervisados por una interrupción maestra, pero, en cambio, se dividen en categorías, según la temática de cada una, con su propio control de interrupción. El beneficio principal de este diseño es que permite que cada controlador de interrupción sea reemplazado con la mínima cantidad de problemas. Por ejemplo, si un fabricante de *hardware* creó de forma radicalmente diferente un visualizador de video, impresora, o cualquier otro elemento que requiera un programa del BIOS completamente nuevo para operarlo, el fabricante podría proveernos un nuevo programa del BIOS junto con el *hardware*. El nuevo programa del BIOS debe estar almacenado en RAM, mejor que en ROM, y sustituiría sólo aquella parte de la ROM-BIOS de IBM que se utilizó con el viejo *hardware*. Haciendo modular la ROM-BIOS, IBM facilita la mejora y extensión de las capacidades de nuestros ordenadores.

INTERRUPCIONES DE SERVICIOS DE LA ROM-BIOS

Hay en total doce interrupciones para la ROM-BIOS, repartidas en cinco grupos: seis de las doce interrupciones sirven a dispositivos periféricos específicos; dos dan cuenta del equipamiento del ordenador; una trabaja con el reloj de hora/fecha; una realiza la operación de escritura en pantalla, y finalmente dos interrupciones dirigen el ordenador a otro estado, activando la ROM-BASIC y la rutina de puesta en marcha del sistema. Como veremos, la mayoría

de las interrupciones están ligadas a un grupo de subservicios, que realizan en realidad el trabajo. Por ejemplo, la interrupción 16 (10 hex) de servicios de video tiene 16 subservicios, que hacen todo, desde poner el modo de video hasta cambiar el tamaño del cursor. Llamamos a un subservicio invocando la interrupción que lo gobierna y especificando el número de subservicio en el registro AH. Este proceso está explicado en el ejemplo del final de este capítulo.

CARACTERISTICAS OPERATIVAS DE LOS SERVICIOS DEL BIOS

Los servicios de la ROM-BIOS utilizan algunos convenios en las llamadas, que proporcionan consistencia en la utilización de los registros, los indicadores, la pila y la memoria. Perfilaremos las

Interrupción		Uso
Dec	Hex	
<i>Servicios de dispositivos periféricos</i>		
16	10	Servicios de visualización de video (véase capítulo 9)
19	13	Servicios de <i>diskette</i> (véase capítulo 10)
20	14	Servicio de comunicaciones (véase capítulo 12)
21	15	Servicio de cassette (véase capítulo 12)
22	16	Servicios del teclado estándar (véase capítulo 11)
23	17	Servicios de impresora (véase capítulo 12)
<i>Servicios de estado del equipo</i>		
17	11	Servicio de listado del equipo (véase capítulo 12)
18	12	Servicio de tamaño de la memoria (véase capítulo 12)
<i>Servicio de hora/fecha</i>		
26	1A	Servicio de hora y fecha (véase capítulo 12)
<i>Servicio de escritura en pantalla</i>		
5	5	Servicio de escritura en pantalla (véase capítulo 12)
<i>Servicios especiales</i>		
24	18	Activación del lenguaje ROM-BASIC (véase capítulo 12)
25	19	Activación de la rutina de puesta en marcha (véase capítulo 12)

Figura 8-1. Los doce servicios de la ROM-BIOS.

características de estos convenios operativos, comenzando con los registros de segmento.

El registro del segmento del código (CS) es reservado, cargado y restablecido automáticamente como parte del proceso de interrupción. Consecuentemente no tenemos que preocuparnos del CS de nuestro programa. Los registros DS y ES son preservados por las rutinas de servicio de la ROM-BIOS, excepto en los pocos casos en que sean utilizados explícitamente. El registro de segmento de pila (SS) se deja sin modificar, y los servicios de la ROM-BIOS dependen de que nosotros les proporcionemos una pila de trabajo. (Todo depende de una pila de trabajo.)

Las exigencias de la pila de la ROM-BIOS no están indicadas y pueden variar considerablemente, puesto que algunos servicios invocan otros servicios. (Uno de los defectos de los ordenadores personales de IBM, y de todos los ordenadores basados en el 8088 de Intel, es la especificación muy confusa sobre el uso de la pila y las limitaciones de la pila.) Generalmente, la mayoría de los programas debe trabajar con pilas mucho más grandes que las que necesitan los servicios de la ROM-BIOS.

En conexión con los registros de segmento, el contador de programa (PC o IP) está preservado por el mismo mecanismo que preserva el segmento de código. De hecho, el puntero de pila (SP) está preservado porque todos los servicios de la ROM-BIOS dejan la pila intacta, desapilando (POP) todo lo que fue apilado (PUSH) durante la ejecución de la rutina de servicio.

Como de costumbre, se puede trabajar a voluntad con los registros de propósito general, AX al DX. La regla general es no esperar que el contenido de estos registros se mantenga cuando pase el control a otra rutina, y se aplica también a los servicios de la ROM-BIOS. Si se inspecciona estrechamente la codificación de los servicios en el «Manual de Referencia Técnica de IBM», se encontrará que en un servicio u otro se dejan sin modificar uno o más registros, pero sería tonto tratar de obtener ventajas de esto. Como regla general, cuando se devuelve un resultado simple de una subrutina, se deja en el registro AX; esto se aplica a la ROM-BIOS y a todos los lenguajes de programación. Veremos lo a menudo que realmente sucede esto cuando tratemos en detalle los servicios de la ROM-BIOS.

Los registros de índice (SI y DI) pueden ser modificados, igual que los registros AX al DX.

El registro de indicadores se cambia asiduamente como producto de la ejecución de instrucciones en las rutinas de la ROM-BIOS. No se debe esperar que permanezca inalterado cualquiera de ellos. En

unos pocos casos, el indicador de acarreo (CF) o el de cero (ZF) son utilizados para señalar el resultado de una operación solicitada.

Los detalles que hemos estado inspeccionando son importantes, pero bastante tediosos, y hay pocas razones para que se preste mucha atención en ellos. Si los programas siguen las reglas generales de interfaz dadas en la próxima sección, y si siguen los requerimientos específicos del lenguaje de programación (● explicados en los capítulos 19 y 20), puede no ser necesario interesarse en ellos en absoluto.

□ *NOTA: Si extiende la utilización de los servicios de la ROM-BIOS en sus programas, estará interesado naturalmente sobre los posibles conflictos entre los servicios y los convenios operativos que sigue su lenguaje. Quédese tranquilo. Encontrará que no tiene que tomar precauciones extraordinarias para proteger su lenguaje de programación de la ROM-BIOS, o viceversa.*

CREACION DE UN INTERFAZ EN LENGUAJE ENSAMBLADOR

Para hacer uso directo de los servicios de la ROM-BIOS desde nuestros programas, es necesario crear una rutina de interfaz en lenguaje ensamblador para enlazar nuestro lenguaje de programación con la ROM-BIOS. Cuando decimos «rutina de interfaz», nos estamos refiriendo a rutinas de desarrollo de programas (subrutinas que están ensambladas en módulos objeto (ficheros OBJ) y enlazadas en los programas de trabajo (ficheros .EXE o .COM en el DOS)). (● Para más información sobre este aspecto, véase capítulo 19.)

Trabajar en lenguaje ensamblador puede parecer una tarea espantosa si uno no está acostumbrado. Mientras hay muchas buenas razones para estar intimidado por el lenguaje ensamblador —después de todo, es el tipo de programación más difícil—, no es difícil realmente crear una rutina de interfaz en lenguaje ensamblador. Como he contado a menudo, la primera vez que necesité crear una rutina de interfaz para los programas de las Norton Utilities que estaba escribiendo en Pascal, no tenía absolutamente ninguna experiencia previa con el ensamblador de IBM, o el lenguaje de máquina y ensamblador del 8088. Aunque empecé con dificultad, tuve mi primera rutina de interfaz completamente hecha y comprobada en 45 minutos. Menciono esto no para jactarme, sino para remarcar que esto no es tan duro como la gente piensa.

Para crear sus propios interfaces, necesitará tener un ensamblador que sea compatible con los estándar del DOS para ficheros objeto. El

que yo utilizo es el macro ensamblador de IBM, pero hay otros disponibles. Sin embargo, no piense en utilizar el famoso «cheap assembler» CHASM, de David Whitman. El CHASM y algunos otros ensambladores producen sólo programas completos en lenguaje ensamblador, en lugar de módulos que puedan ser enlazados en otros programas, tales como las rutinas de interfaz en las que estamos interesados. Todos los ejemplos que damos aquí son para el macro ensamblador de IBM.

□ NOTA: *El BASIC puede trabajar con subrutinas en lenguaje máquina puestas directamente en la memoria. La preparación de las diversas subrutinas en ensamblador que trabajaran con BASIC se puede hacer tan fácilmente con el comando de ensamblador A del DEBUG, como utilizando un ensamblador ordinario.*

Forma básica de una rutina de interfaz

La forma exacta de una rutina de interfaz varía según su uso. Un interfaz en lenguaje ensamblador es un enlace entre nuestro lenguaje de programación y un servicio de la ROM-BIOS; y tiene que ser diseñado para que satisfaga los requerimientos de ambos extremos. Es importante el lenguaje de programación que se esté utilizando y qué servicio de la ROM-BIOS se invoque; también importa si se está pasando algún dato en una dirección u otra.

Sin embargo, las líneas generales de un interfaz en lenguaje ensamblador es básicamente la misma, no importa lo que se esté haciendo.

Una de las mejores formas de comprender cómo se codifica un interface en lenguaje ensamblador es verlo como cinco partes anidadas, que se esquematizan aquí:

- Nivel 1: Encabezamiento general de ensamblado
- Nivel 2: Encabezamiento de ensamblado de la subrutina
- Nivel 3: Código de entrada
- Nivel 4: Paso de los datos
- Nivel 5: Llamada al servicio de la ROM-BIOS
- Nivel 4: Devolución de datos
- Nivel 3: Código de salida
- Nivel 2: Final de subrutina
- Nivel 1: Final del proceso

En este esquema, los niveles 1 y 2 son necesarios para decirle al ensamblador lo que está pasando, pero no producen ninguna instrucción de trabajo. Los niveles 3 al 5 producen las verdaderas instrucciones en lenguaje máquina.

Profundizaremos en todos estos niveles para mostrarle las reglas y explicarle lo que pasa. No olvide que los requerimientos específicos de una rutina de interfaz cambian según sean las circunstancias. Le indicaremos los pocos elementos de diseño que son universales para todas las rutinas

Nivel 1: encabezamiento general de ensamblado

A continuación se da la descripción de una típica sección de nivel 1 de una rutina de interfaz, las líneas están numeradas para referencias posteriores.

```
1-1 INTERFAZ SEGMENT 'CODE'
1-2         ASSUME  CS:INTERFAZ
           ; aquí aparecen los niveles desde el 2 hasta el 5
1-3 INTERFAZ ENDS
1-4         END
```

En la línea 1-1, INTERFAZ es un nombre arbitrario que hemos dado a esta rutina de ensamblador; SEGMENT es esencial y se utiliza para definir una rutina de ensamblador; CODE es una categoría que puede variar según el lenguaje (veremos otro ejemplo brevemente).

La línea 1-2 no es siempre necesaria. Simplemente la instrucción ASSUME nos permite hacer programación del tipo *si-entonces* sin ningún tipo de problemas; la utilizaremos en capítulos posteriores.

La línea 1-3 termina el segmento empezado en la línea 1-1, y la línea 1-4 acaba la rutina en ensamblador.

Las convenciones de formato que hemos mostrado aquí están tomadas del IBM/Microsoft Pascal (■ que veremos en el capítulo 20). Para un ejemplo de algo diferente, el C necesita estas dos líneas en lugar de la línea 1-1:

```
PGROUP      GROUP      PROG
           INTERFAZ SEGMENT  BYTE PUBLIC 'PROG'
```

Nivel 2: encabezamiento de ensamblado de la subrutina

Seguidamente, veremos una descripción de un típico nivel 2, el encabezamiento de ensamblador para una subrutina (denominada *procedimiento* en el lenguaje ensamblador). He aquí una codificación típica del nivel 2:

```
2-1         PUBLIC  MEMTAM
2-2 MEMTAM   PROC    FAR
           ; aquí aparecen los niveles desde el 3 hasta el 5
2-3 MEMTAM   ENDP
```

La línea 2-1 le indica al ensamblador que ponga el nombre de nuestro procedimiento. MEMTAM, como información pública, lo que significa que el programa de montaje (*linker*) puede entonces conectarlo a los usuarios.

Las líneas 2-2 y 2-3, nuestro procedimiento, que ha sido llamado arbitrariamente MEMTAM PROC y ENDP, son obligatorias y están ligadas a cualquier procedimiento, con PROC que define el comienzo del procedimiento y ENDP señala su final. FAR dice al ensamblador que el procedimiento está localizado fuera del segmento en curso. Se podía haber utilizado FAR o NEAR. Si se hubiera utilizado NEAR, se habría indicado que el procedimiento estaba localizado dentro del segmento en curso, en lugar de otro de fuera. Las llamadas FAR son las más comunes, pero algunos lenguajes utilizan (C) o pueden usar (Pascal) llamadas NEAR. Excepto FAR, o NEAR, lo visto aquí es universal para todos los lenguajes y propósitos.

Nivel 3: código de entrada

En el nivel 3 comienzan las instrucciones reales de trabajo. Realiza el conjunto de operaciones de gestión interna necesarias para que una rutina trabaje cooperativamente con el lenguaje que la llamó. He aquí un ejemplo:

```
3-1  PUSH    BP
3-2  MOV     BP,SP
      ; aquí aparecen los niveles 4 y 5
3-3  POP     BP
3-4  RET     0
```

Las líneas 3-1 y 3-2 son utilizadas para entrar y preservar los parámetros que ha pasado el programa llamador. Estos aparecerán de una u otra forma en la pila. El registro de puntero de base (BP) se utiliza universalmente para tomar nota de la localización del punto de entrada en la pila. Nuestro programa llamador tendrá su propio BP, que preservamos en la línea 3-1, apilándolo (PUSH) en la pila, y que recuperamos en la línea 3-3, desapilándolo (POP).

En la línea 3-2 obtenemos nuestra propia referencia de pila, cogiendo el puntero de pila (SP) y moviéndolo al BP. A partir de ahí, no importa lo que se agregue a la pila, habremos tomado nota de dónde están en la pila los parámetros del programa llamador. Si se necesitan preservar otros registros, deberán de cargarse en la pila inmediatamente, después de la línea 3-2, y desapilándolos, en orden inverso, justo antes de la línea 3-3. Normalmente, esto no será necesario.

La línea 3-4 se utiliza para devolver el control; el ensamblador convierte nuestro RET en un retorno NEAR, o FAR, según si nuestra PROC fue declarado NEAR, o FAR. El 0 de la línea 3-4 es un trabajo de limpieza que normalmente tenemos que hacer para quitar de la pila los parámetros del llamador. Si no había parámetros o si las convenciones del lenguaje de programación hacen que el llamador suprima los parámetros de la pila, como hace el C (véanse páginas 22 y 31), entonces este valor será cero. Si había parámetros o el lenguaje de programación no deja intacta la pila, tenemos que conocer el valor necesario para que supriman todos los parámetros. El valor debe ser incrementarse por dos para cada parámetro de 1 ó 2 bytes (byte, palabra, o dirección relativa), y por 4 para cada parámetro de 4 bytes (dirección segmentada). Mientras que podamos identificar la naturaleza de nuestros parámetros (que serán de alguno de estos cuatro tipos), todo irá bien.

Nivel 4: paso de datos desde el programa llamador

El nivel 4 trata los parámetros pasándolos del programa llamador a la ROM-BIOS y los resultados, de la ROM-BIOS al llamador. Los parámetros están en la pila, en forma de datos o direcciones (véase capítulo 20 para una ayuda sobre este tema). Los registros, sobre todo del AX al DX, son utilizados por la ROM-BIOS para entradas y salidas. La treta aquí —y puede ser difícil— es utilizar los offsets de pila correctos para encontrar los parámetros. Se abordará este problema por partes.

Primero se obtienen los parámetros de la pila mediante direccionamiento relativo respecto a la base BP. He aquí un ejemplo típico:

Localización	Contenidos
BP	BP del llamador, salvado
BP + 2	Dirección de retorno, offset y segmento
BP + 6	Un parámetro
BP + 8	Otro parámetro
BP + 10	Otro parámetro más

La dirección de retorno en BP+2 es de cuatro bytes para un procedimiento FAR, pero sólo de dos bytes para un procedimiento NEAR. Si es un procedimiento NEAR, todos los subsiguientes offsets serían 2 unidades menos que los que se muestran aquí. La mayoría de los lenguajes guarda sus parámetros en la pila en el orden en que se escriben. Esto significa que el último parámetro es el que se escribe en la parte superior de la pila, en BP+6. Sin embargo, los C de Lattice y

Microsot utilizan el orden inverso, de forma que el parámetro que cierra es el primero escrito en el programa de llamada.

Normalmente, los parámetros ocupan dos o cuatro bytes de la pila, aunque lo más común es dos bytes. Nuestro ejemplo tiene las localizaciones $BP+6$, $+8$ y $+10$ separadas por dos bytes. Si alguno de estos parámetros tuviera un tamaño de cuatro bytes, ajustaríamos las referencias subsiguientes teniéndolo en cuenta.

Si se situó un *dato* en la pila, podemos obtenerlo inmediatamente, direccionándolo de forma: $[BP+6]$ Si lo que se situó fue una *dirección*, son necesarios dos pasos: primero, obtener la dirección, y segundo, obtener el dato. He aquí un ejemplo del nivel 4 que muestra la recuperación de un dato ($[BP+6]$) y de una dirección ($[BP+8]$).

```

4-1 MOV  AX,[BP+6]           ; valor del parámetro 1
4-2 MOV  DX,[BP+8]           ; dirección del parámetro 2
4-3 MOV  BX,[DX]             ; valor del parámetro 2
    ; aquí aparece el nivel 5
4-4 MOV  DX,[BP+8]           ; dirección del parámetro 2 (de nuevo)
4-5 MOV  [DX],BX             ; devuelve el nuevo valor

```

Todas estas instrucciones MOV mueven datos del segundo operando al primer operando. La línea 4-1 coge directamente un dato de la pila y lo introduce en el registro AX. Las líneas 4-2 y 4-3 toman un dato utilizando una dirección de la pila; la línea 4-2 obtiene primero la dirección (dejándola en DX), y entonces la línea 4-3 utiliza esa dirección para conseguir el dato real que sitúa en BX. Las líneas 4-4 y 4-5 realizan el proceso inverso; la línea 4-4 obtiene la dirección otra vez, y entonces la línea 4-5 mueve el contenido de BX a la posición de memoria.

□ **NOTA:** *Se muestra aquí un punto crucial de la notación de ensamblador: DX se refiere a lo que está en DX, y [DX] se refiere a una posición de memoria cuya dirección está en DX.*

Aunque parezca que esta nomenclatura es un poco confusa, si pensamos atentamente, funciona bien.

Nivel 5: invocación del servicio de la ROM-BIOS

El nivel 5 es nuestro paso final: sencillamente invoca el servicio de la ROM-BIOS. Típicamente, este paso incluye dos instrucciones simples, como éstas:

```

5-1 MOV  AH,1               ; función 1
5-2 INT  16                 ; llama la rutina BIOS

```

La línea 5-1 selecciona un subservicio de la interrupción. Típicamente hay varios subservicios numerados de 0 en adelante. Siempre se seleccionan con un código en el registro AH.

La línea 5-2 genera la interrupción que requiere al servicio; en este ejemplo, es la interrupción 16 (10 hex), es la interrupción de servicios del video.

Este proceso de cinco pasos describe los principios básicos de casi todos los aspectos de una interfaz en lenguaje ensamblador. En los siguientes capítulos veremos cómo se utiliza este diseño en ejemplos específicos.

Servicios de video de la ROM-BIOS

Acceso a los servicios de video del BIOS	192
Servicio 0: Ajuste del modo de video	193
Servicio 1: Ajuste del tamaño del cursor	194
Servicio 2: Posicionamiento del cursor	195
Servicio 3: Lectura de la posición del cursor	196
Servicio 4: Lectura de la posición del lápiz óptico	196
Servicio 5: Elección de la página de visualización activa	197
Servicio 6: Desplazamiento vertical ascendente (<i>scroll up</i>)	198
Servicio 7: Desplazamiento vertical descendente (<i>scroll down</i>)	199
Servicio 8: Lectura del carácter y del atributo	199
Servicio 9: Escritura de un carácter y del atributo	200
Servicio 10 (A hex): Escritura de un carácter	201
Servicio 11 (B hex): Elección de la paleta de color	202
Servicio 12 (C hex): Escritura de un <i>pixel</i>	203
Servicio 13 (D hex): Lectura de un <i>pixel</i>	204
Servicio 14 (E hex): Escritura de un carácter como TTY	204
Servicio 15 (F hex): Obtención del modo de video actual	205
Servicio 19 (13 hex): Escritura de una cadena de caracteres	206
Comentarios y ejemplo	206

En este capítulo se discutirán cada uno de los servicios de video, o de control de pantalla proporcionados por la ROM-BIOS. Se ha dedicado la mayor parte del capítulo a descripciones detalladas de cada servicio de video. Empezando por el capítulo 9, hemos incluido algunas indicaciones de programación y una rutina de lenguaje ensamblador que hace uso de alguno de los servicios de video. Para un estudio más general de las características de los visualizadores de video en la familia PC, véase capítulo 4. Para información sobre las posiciones bajas de la memoria utilizadas por la ROM-BIOS para guardar la información del estado del video, véase página 62.

ACCESO A LOS SERVICIOS DE VIDEO DEL BIOS

Todos los servicios de video de la ROM-BIOS se solicitan mediante la interrupción 16 (10 hex). Hay 16 servicios principales y un servicio de AT disponibles bajo esta interrupción (véase la figura 9-1). Al igual que los otros servicios de la ROM-BIOS, los servicios de video están numerados a partir del 0, y se seleccionan situando el número del servicio en el registro AH. Los servicios necesitan, a menudo, parámetros adicionales, que se sitúan en BX,

Servicio		Descripción
Dec	Hex	
0	0	Ajuste de modo de video
1	1	Ajuste del tamaño del cursor
2	2	Posicionamiento del cursor
3	3	Lectura de la posición del cursor
4	4	Lectura de la posición del lápiz óptico
5	5	Elección de la página de visualización activa
6	6	Desplazamiento vertical ascendente (<i>Scroll up</i>)
7	7	Desplazamiento vertical descendente (<i>Scroll down</i>)
8	8	Lectura del carácter y del atributo
9	9	Escritura del carácter y del atributo
10	A	Escritura de un carácter
11	B	Elección de la paleta de color
12	C	Escritura de un <i>pixel</i>
13	D	Lectura de un <i>pixel</i>
14	E	Escritura de un carácter como TTY
15	F	Obtención del modo de video actual
19	13	Escritura de una cadena de caracteres

Figura 9-1. Los diecisiete servicios de video

CX o DX, en función de las especificaciones de la rutina de servicio. Trataremos sobre el propósito y situación de los parámetros en cada descripción de servicio.

Servicio 0: Ajuste del modo de video

El servicio 0 se utiliza para seleccionar entre los catorce modos de video mostrados en la figura 9-2. Para más detalles, los modos de video, véase página 82.

Se puede recordar del capítulo 4 que los modos 0 al 6 se aplican al adaptador color/gráficos (CGA); el modo 7 se aplica al adaptador monocromo; los modos 8 al 10 se introdujeron con el PCjr, y los modos 13 al 15 se añadieron para el adaptador gráfico mejorado (EGA), que también mantiene el resto de modos, excepto los modos 8 y 9.

Algo más que se puede tener en cuenta si se está trabajando con los modos blanco y negro, o con la supresión del color (modos 0, 2 y 5), es que sólo suprimen el color en la salida compuesta y no en la salida RGB.

Normalmente, la ROM-BIOS borra el *buffer* de pantalla cuando se activa un modo, incluso aunque se active el mismo modo una y otra vez. Por tanto, la activación de mismo modo repetidamente puede ser una forma fácil y efectiva de borrar la pantalla. Sin

Modo	Tipo	Colores	Resolución	Dimensiones	Adaptador
0	Texto	b/n	Media	40 × 25	CGA
1	Texto	16	Media	40 × 25	CGA
2	Texto	b/n	Alta	80 × 25	CGA
3	Texto	16	Alta	80 × 25	CGA
4	Gráficos	4	Media	320 × 200	CGA
5	Gráficos	4 gris	Media	320 × 200	CGA
6	Gráficos	b/n	Alta	640 × 200	CGA
7	Texto	b/n	Alta	80 × 25	Monocr.
8	Gráficos	16	Baja	160 × 200	PCjr
9	Gráficos	16	Media	320 × 200	PCjr
10	Gráficos	4,64	Alta	640 × 200	PCjr, EGA
13	Gráficos	16	Media	320 × 200	EGA
14	Gráficos	16	Alta	640 × 200	EGA
15	Gráficos	4	Muy alta	640 × 350	EGA

Figura 9-2. Valores de los modos de video para el servicio 0

embargo, no es la operación ideal para el borrado de la pantalla en los Compaq compatibles-PC, ya que presentan una perceptible espera cuando se utiliza esta técnica.

☛ Véase capítulo 4, página 82, para más información sobre los modos de video. Véase página 62, posición de memoria 449 hex, para más información sobre cómo se almacena en la memoria el registro de modo. Véase el servicio 15 (F. hex) para encontrar la manera de leer el modo de video en curso.

Servicio 1: Ajuste del tamaño del cursor

El servicio 1 controla la forma y tamaño del cursor que aparece en los modos de texto. El cursor IBM estándar aparece normalmente como una o dos líneas de barrido parpadeantes en la parte inferior de la posición de visualización de un carácter. Se puede cambiar el tamaño que tiene por defecto el cursor redefiniendo el número de líneas de barrido que son visualizadas.

El adaptador color/gráficos puede visualizar un cursor que tiene ocho líneas de barrido, numeradas del 0, en la parte superior, al 7, en la inferior. El adaptador monocromo y el EGA pueden visualizar un cursor que tiene catorce líneas de barrido, también numeradas desde la parte superior, del 0 al 13. Se ajusta el tamaño del cursor especificando las líneas de barrido inicial y final. (Estos son los mismos que los parámetros de comienzo y fin de la instrucción LOCATE del BASIC.) El número de línea de comienzo se carga en el registro CH, y el número de línea de fin en el registro CL. Los valores por defecto son CH = 6, CL = 7 para el adaptador de color/gráficos, y CH = 12, CL = 13 para el adaptador monocromo.

Si la línea de comienzo es menor que la línea de fin, aparece el cursor normal de una pieza. Si la línea de comienzo es mayor que la línea de parada, se producirá un cursor de dos partes.

Se observa que los números de línea válidos ocupan sólo tres de

Número de servicio	Parámetros
AH = 1	CH = línea de comienzo del cursor CL = línea de fin del cursor

Figura 9-3. Registros utilizados para ajustar el tamaño del cursor utilizando el servicio 1

los bits (bits 0 al 2) situados en estos registros. Si el bit 5 de CH se activa con un valor de 32 (20 hex), el cursor desaparecerá. Cuando un modo de gráficos está activado, el bit 5 se activa automáticamente para resguardar el mecanismo del cursor de interferir con el visualizador de gráficos. Esta es una de las dos técnicas que podemos utilizar para eliminar el cursor en los modos de texto. La otra técnica es sacarlo de la pantalla, mandarlo a la fila 26, columna 1. Puesto que no hay un cursor verdadero en los modos gráficos, cualquier cursor que se vea está simulado con el carácter de bloque sólido, CHR\$(223), o con un cambio de los atributos de fondo.

• Para más información sobre cursores, véase página 104. Véase el servicio 3 para la operación inversa: lectura de la posición del cursor.

Servicio 2: Posicionamiento del cursor

El servicio 2 fija la posición del cursor utilizando las coordenadas de la fila y la columna. En los modos de texto puede haber múltiples páginas de visualización, cada una teniendo la posición de cursor registrada independientemente. Incluso aunque los modos gráficos no tienen un cursor visible, pueden seguir la pista de la posición lógica del cursor, de la misma forma que los modos de texto. Esta posición lógica del cursor se utiliza para supervisar la E/S de caracteres.

La posición del cursor se especifica situando el número de fila en el registro DH, el número de columna en DL y el número de página en BH. La numeración de filas y columnas empieza con las coordenadas 0,0 en la esquina superior izquierda. Los modos gráficos utilizan también las coordenadas de fila y columna del carácter para identificar la localización del cursor, en lugar de las coordenadas del *pixel*. El número de página es el mismo número de página que utiliza

Número de servicio	Parámetros
AH = 2	DH = número de fila DL = número de columna BH = número de página (puesto a 0 en los modos gráficos)

Figura 9-4. Registros utilizados para fijar la posición del cursor utilizando el servicio 2

el BASIC: páginas 0 a 7 en los modos de 40 columnas, y páginas 0 a 3 en los modos de 80 columnas. El número de página debe de ponerse a 0 en los modos gráficos.

☛ Véase figura 9-4 para más información sobre las páginas de visualización. Véase página 98 para más información sobre los formatos de visualización del texto. Véase el servicio 3 para la operación inversa: lectura de la posición del cursor.

Servicio 3: Lectura de la posición del cursor

El servicio 3 es el opuesto de los servicios 1 y 2. Cuando especificamos el número de página en BH, la ROM-BIOS informa sobre el tamaño del cursor devolviendo la línea de comienzo en CH y la línea de fin en CL. Informa sobre la posición del cursor devolviendo la fila en DH y a la columna en DL. Como con el servicio 2, la página debe fijarse a 0 en los modos gráficos.

☛ Véase figura 9-5 para más información sobre las páginas de visualización. Véase página 98 para más información sobre los formatos de visualización del texto.

Número de servicio	Parámetros
AH = 3	BH = número de página (puesto a 0 en los modos de gráficos) DH = número de fila DL = número de columna CH = línea de comienzo del cursor CL = línea de fin del cursor

Figura 9-5. Registros utilizados para leer la posición del cursor utilizando el servicio 3

Servicio 4: Lectura de la posición del lápiz óptico

El servicio 4 informa sobre el estado del lápiz óptico, especificando si ha sido disparado o no, y dónde está en la pantalla en el caso de que así haya sido.

El registro AH se utiliza para indicar el disparo: si AH es 1, el lápiz óptico ha sido disparado; si AH es 0, no ha sido disparado. La localización del *pixel* del lápiz óptico en la pantalla se detecta mediante el *hardware*, y la ROM-BIOS nos lo informa traducido de

Número de servicio	Parámetros
AH = 4	DH = número de fila del carácter DL = número de columna del carácter CH = número de línea de <i>pixel</i> (0 al 199) BX = número de columna del <i>pixel</i>

Figura 9-6. Registros utilizados para leer la posición del lápiz óptico utilizando el servicio 4

dos formas: la posición del carácter (fila en DH, columna en DL) y la localización del *pixel* (línea de barrido en CH, columna/punto en BX). Puesto que la localización de la columna del *pixel* puede ser mayor que 255, se manda a un registro de una palabra entera. Todos los otros valores se manejan como bytes.

Servicio 5: Elección de la página de visualización activa

El servicio 5 elige la página de visualización activa para los modos texto 0 al 3. Se especifica el número de página en el registro AL. En los modos de 40 columnas podemos elegir de las páginas 0 a la 7, y en los modos de 80 columnas, de las páginas 0 a la 3. La página 0 se utiliza por defecto. La página 0 está localizada al principio de la memoria de visualización, con cada página subsiguiente empezando cada 2Kbytes (en los modos de 40 columnas) o 4Kbytes (en los modos de 80 columnas). Los números de páginas más elevados están en las posiciones de memoria más altas.

☛ Véase página 97 para más información sobre las páginas de visualización.

Número de servicio	Parámetros
AH = 5	AL = nuevo número de páginas de visualización (0–3 para los modos 2 y 3, 0–7 para los modos 0 y 1)

Figura 9-7. Los registros utilizados para elegir la página de visualización activa utilizando el servicio 5

Servicio 6: Desplazamiento vertical ascendente (Scroll up)

El servicio 6 y su compañero el servicio 7 se utilizan para definir un área para textos, como una ventana rectangular en la pantalla y desplazar hacia arriba o hacia abajo los contenidos de una o más líneas. Para conseguir el efecto de desplazamiento, se insertan líneas en blanco en la parte inferior del área de la ventana con el servicio 6 (en la parte superior con el servicio 7) y las líneas superiores de la ventana (las líneas inferiores con el servicio 7) se desplazan fuera y desaparecen.

El número de líneas a desplazar se especifica en AL. Si AL es 0, se borra la ventana entera (lo mismo que ocurriría si desplazáramos más líneas de las que permita el tamaño de la pantalla). La localización, o el tamaño, de la ventana se especifica en los registros CX y DX: CH es la fila superior, y DH es la fila inferior; CL es la columna izquierda, y DL es la columna derecha. El atributo de visualización de las nuevas líneas en blanco insertadas por los dos servicios se toman de BH. La figura 9-8 muestra un sumario de la utilización de los registros para ambos servicios 6 y 7.

El desplazamiento de la ventana es normalmente un proceso de dos fases. Cuando una nueva línea está lista para escribirse en la ventana, se utiliza el servicio 6 (o servicio 7) para desplazar el contenido de la ventana. Así, la nueva información se escribe en la nueva línea, utilizando los servicios de posicionamiento del cursor y de escritura de carácter. El siguiente ejemplo muestra esta acción de ventana.

Número de servicio	Parámetros
AH = 6	AL = número de líneas a desplazar
AH = 7	CH = fila superior
	CL = columna izquierda
	DH = fila inferior
	DL = columna derecha
	BH = atributo de visualización para las líneas nuevas

Figura 9-8. Registros utilizados para ajustar el tamaño de la ventana para el desplazamiento utilizando los servicios 6 y 7

DEBUG	; llama al DEBUG desde el DOS
A	; pide ensamblar las instrucciones
INT 10	; interrupción de creación instrucción hex 10
[Return]	; termina de ensamblar
R AX	; pide ver y cambiar los contenidos de AX
0603	; especifica el servicio 6 (scroll up), 3 líneas
R CX	; pide ver y cambiar los contenidos de CX
050A	; especifica la esquina superior izquierda: fila 5, columna 10
R DX	; pide ver y cambiar los contenidos de DX
1020	; especifica la esquina inferior derecha: fila 16, columna 32
D 0 L 180	; llena la pantalla con cosas sin sentido
G = 100 102	; ejecuta INT 10, después para

☛ Véase capítulo 8 para más información sobre rutinas en lenguaje ensamblador. Véase el «Manual de Referencia del DOS» para más información sobre el DEBUG.

Servicio 7: Desplazamiento vertical descendente (Scroll down)

El servicio 7 es, como ya hemos mencionado, equivalente al servicio 6. La principal diferencia entre los dos servicios es la acción de desplazamiento. En el servicio 7 las nuevas líneas en blanco aparecen en la parte superior de la ventana, y las viejas líneas desaparecen por la parte inferior. La acción de desplazamiento es opuesta a la que tiene lugar en el servicio 6. ☛ Véase la descripción de los parámetros en el servicio 6.

Servicio 8: Lectura del carácter y del atributo

El servicio 8 se utiliza para leer caracteres «dibujados en la pantalla»; esto es, directamente de la memoria de visualización. Este servicio es inusualmente elegante, porque trabaja en ambos modos, el de texto y el de gráficos.

En los modos de gráficos, las mismas tablas que se utilizan para dibujar los caracteres se utilizan también para reconocerlos con una operación de verificación. Incluso si creamos nuestro propio conjunto de caracteres en el modo gráfico, este servicio podrá reconocerlos. En modo texto, por supuesto, los códigos de los caracteres ASCII están directamente disponibles en la memoria de visualización.

El servicio 8 devuelve el código ASCII del carácter leído de la pantalla en AL. En el modo gráfico, si el carácter no corresponde con ningún código ASCII estándar, se informa con 00 hex. En los

modos de texto, el servicio devuelve también los atributos del color del texto en AH. En modo texto, la página de visualización debe ser especificada en BH. No es necesario en los modos gráficos indicar el número de página.

Véase página 89 para más información sobre los caracteres del texto y los atributos. Véase página 98 para saber más sobre los caracteres en modo texto y en modo gráfico. Véase apéndice C para más información sobre los caracteres ASCII.

Número de servicio	Parámetros
AH = 8	BH = página de visualización activa (no necesario en los modos de gráficos) AL = carácter ASCII leído en la localización del cursor AH = atributo del carácter

Figura 9-9. Registros para leer un carácter y el atributo utilizando el servicio 8

Servicio 9: Escritura de un carácter y del atributo

El servicio 9 escribe una o más copias de un único carácter y su atributo de color. El carácter se especifica en AL, y el atributo en modo de texto, o el color en modo de gráfico, se especifican en BL. El número de veces que va a ser escrito el carácter (uno o más veces) se sitúa en CX.

Para los modos de texto, la página de visualización debe ser especificada en BH; no necesita especificarse para los modos gráficos.

El carácter con sus atributos de color se escribe tantas veces como se requiera, comenzando en la localización en curso del cursor. Aunque el cursor no se mueve, los caracteres duplicados se escriben en las subsiguientes localizaciones de pantalla. En modo de texto, los caracteres duplicados saltarán de línea cuando sea necesario, lo que incrementa la utilidad de este servicio. En modo gráfico, esto no ocurre.

El servicio 9 es bastante útil para la escritura de caracteres individuales y para la reproducción de un carácter. La operación de repetición se utiliza la mayor parte de las veces para desplegar rápidamente espacios en blanco, u otros caracteres repetidos, tal como las líneas horizontales que forman parte de los dibujos de cajas (véase apéndice C). Cuando se desee hacer una única copia de un

Número de servicio	Parámetros
AH = 9	AL = carácter ASCII para escribir en la pantalla BL = atributo del carácter para escribir en la pantalla BH = página de visualización activa (no es necesario en los modos de gráficos) CX = número de veces que hay que escribir el carácter y el atributo

Figura 9-10. Registros utilizados para escribir un carácter y el atributo utilizando el servicio 9

carácter, hay que asegurarse de poner CX a 1. Si está puesto a 0, el número de repeticiones se disparará.

El servicio 9 tiene una ventaja sobre su similar, el servicio 14, y es que podemos controlar los atributos del color. Sin embargo, su única desventaja es que el cursor no avanza automáticamente.

En el modo de gráficos, el color especificado en BL es el color del primer plano; es decir, el color de los *pixels* que componen el dibujo del carácter. Si el bit 7 es 1 (valor de 128 ó 80 hex), entonces los bits de color en BL se combinan con los bits de color de los *pixel* que hay en pantalla con una operación O exclusiva (XOR). Esta es una forma conveniente de asegurar que el color resultante es diferentes del que había antes, una garantía de legibilidad. Si el bit 7 de BL es 0, el color en BL reemplaza simplemente los colores de *pixel* existentes. La misma característica se aplica también a los servicios de escritura del carácter y de *pixel*, servicios 10 y 12.

☛ Véase página 89 para más información sobre los atributos en los modos texto. Véase página 93 para más información sobre los atributos de color en los modos gráficos.

Servicio 10 (A hex): Escritura de un carácter

El servicio 10 es el mismo que el servicio 9, con una excepción: el servicio 9 nos permite cambiar el atributo del color existente en el modo de texto, mientras que el servicio 10 no. Sin embargo, en el modo de gráficos, el color necesita ser especificado en BL, siendo este servicio únicamente un servicio de escritura de caracteres parcialmente incorrecto. Se aplican las mismas reglas de color de gráficos que en los servicios 9 y 12: el color puede ser utilizado directamente, o hacer el XOR con el color existente. (☛ Véase el servicio 9 para una explicación.)

Número de servicio	Parámetros
AH = 10	AL = carácter ASCII que se va a escribir en la pantalla BL = atributo de color para los modos gráficos BH = página de visualización activa CX = número de veces que se va a escribir el carácter

Figura 9-11. Registros utilizados para escribir un carácter utilizando el servicio 10

☛ Véase página 89 para más información sobre los atributos de visualización en los modos de texto. Véase página 93 para más información sobre atributos de color en los modos de gráficos.

Servicio 11 (B hex): Elección de la paleta de color

El servicio 11 se utiliza para seleccionar una de las dos paletas gráficas de resolución media. Para utilizar este servicio, cargamos BH con el identificador de la paleta de color y BL con un valor cualquiera de color. (☛ Véase capítulo 4 para más información sobre las paletas color.)

Se aplica una variación de este servicio a los modos de texto; todas las demás se aplican sólo a los modos de gráficos. En los modos de texto, si BH está a 0, entonces BL especifica el color del marco del área de texto (un color escogido entre los 16 colores de la paleta). En cualquier modo de gráficos, si BH es 0, entonces BL especifica el color por defecto del fondo, así como el marco límite. El área límite se fusiona con cualquier parte del área de la pantalla de trabajo que tenga el mismo color que el fondo. El valor BL puede seleccionarse entre los 16 colores de la pantalla completa.

Por otra parte, si BH está a 1, entonces BL selecciona la paleta que se utiliza. Para el adaptador color/gráficos (CGA), esto se aplica sólo al modo 4 (resolución media, para gráficos de cuatro colores). Para adaptadores de visualización más avanzados, incluyendo el del PCjr, esto se puede aplicar también a los otros modos. Aquí nos ocuparemos sólo de las paletas estándar de cuatro colores que nos proporciona el modo 4: las paletas 0 y 1. El número de paleta se selecciona con BL. La paleta 0 tiene estos cuatro colores:

- 0: Color de fondo en curso
- 1: Verde (2)
- 2: Rojo (4)
- 3: Castaño (6)

Número de servicio	Parámetros
AH = 11	BH = identificador de paleta (0 ó 1 en gráficos de 320 × 200) BL = valor del color o de la paleta que se va a utilizar

Figura 9-12. Registros utilizados para elegir la paleta utilizando el servicio 11

La paleta 1 tiene estos cuatro colores:

- 0: Color de fondo en curso
- 1: Cian (3)
- 2: Magenta (5)
- 3: Blanco (7)

Servicio 12 (C hex): Escritura de un pixel

El servicio 12 escribe un *pixel* individual. Puesto que el posicionamiento del cursor utilizado en los servicios 9, 10 y 14 sirve sólo para caracteres, este servicio requiere la especificación de la línea y de la columna. Como de costumbre, las situaciones respectivas se numeran desde el 0, comenzando en la parte superior izquierda de la pantalla.

El número de fila, que requiere sólo un byte, está especificado en DL. El número de columna, que requiere más de un byte, está especificado en CX. El color se da en AL, con la opción del color directo o bien con la del color operado con XOR (O-exclusivo). (☛ Véase el servicio 9 para una explicación.)

Número de servicio	Parámetros
AH = 12	AL = código de color del <i>pixel</i> (0-15) DL = número de fila del <i>pixel</i> CX = número de columna del <i>pixel</i>

Figura 9-13. Registros utilizados para escribir un pixel utilizando el servicio 12

☛ Véase página 100 para más información sobre los *pixels* en los modos de gráficos.

Servicio 13: (D hex): Lectura de un pixel

El servicio 13 es el inverso del servicio 12: lee el contenido del *pixel* en vez de escribirlo. Un *pixel* tiene un único atributo de color, que es exactamente la información que se devuelve por medio del servicio 13. (El servicio 8 de lectura de caracteres devuelve un color y un código de carácter ASCII.) La fila se especifica en DL, no en DX (véase la nota en el servicio 12), y la columna CX. El código del color del *pixel* se devuelve a AL. Todos los bits de orden alto están puestos a 0, como cabría esperar.

Número de servicio	Parámetros
AH = 13	AL = código de color del <i>pixel</i> (0-15) DL = número de fila del <i>pixel</i> CX = número de columna del <i>pixel</i>

Figura 9-14. Registros utilizados para leer un pixel utilizando el servicio 13

Servicio 14 (E hex): Escritura de un carácter como TTY

El servicio 14 es el servicio básico de salida de caracteres. Escribe caracteres individuales en la pantalla en el modo que se conoce como teletipo (TTY). Este hace que la pantalla actúe como la forma de impresión más simple y elemental (únicamente lo necesario para la salida de texto). Como tal, no contempla sutilezas tales como el color, parpadeo de caracteres, o control sobre la localización del cursor.

Cuando se utiliza este servicio, el carácter se escribe en la localización del cursor, y el cursor avanza una posición cubriendo nuevas líneas, o desplazando la pantalla cuando sea necesario. El carácter a escribir se especifica en AL.

En el modo texto, los atributos en curso de la pantalla se mantienen de un carácter al siguiente. En el modo de gráficos, el valor del primer plano se debe especificar cada vez en el registro BL.

Hay cuatro caracteres para los que el servicio 14 reacciona de acuerdo a su significado ASCII: CHR\$(7): señal sonora, CHR\$(8): retroceso (borra el último carácter), CHR\$(10): salta una línea y CHR\$(13): retorno del carro. El resto de caracteres simplemente se visualiza normalmente.

Número de servicio	Parámetros
AH = 14	AL = carácter ASCII que se va a escribir BL = color de primer plano del carácter (sólo en el modo de gráficos) BH = página de visualización activa (no es necesario en los modos de gráficos)

Figura 9-15. Registros utilizados para escribir un carácter como TTY utilizando el servicio 14

La principal ventaja de este servicio sobre el servicio 9 es que el cursor se mueve automáticamente; la ventaja del servicio 9 es que podemos controlar el atributo de color. Si pudiéramos combinar los dos...

Servicio 15 (F hex): Obtención del modo de video actual

El servicio 15 devuelve el modo de video en curso y junto con otras dos informaciones útiles: la anchura de la pantalla en caracteres (80, 40 ó 20) y el número de página de visualización.

El modo de video, como se explicó en el servicio 0, se devuelve en AL. La anchura de la pantalla se devuelve en AH, en número de caracteres por línea (el modo de gráficos de baja resolución se indicará como una anchura de 20 caracteres). La página de visualización se devolverá en BH. El número de página se pondrá a 0 en el modo de gráficos.

☛ Véase página 83 para más información sobre los modos de video. Véase página 62, la localización de memoria 449 hex, para más información sobre cómo se guarda registro del modo.

Número de servicio	Parámetros
AH = 15	AL = modo de visualización en curso AH = número de caracteres por línea BH = página de visualización activa (0 en los modos gráficos)

Figura 9-16. Registros utilizados para leer el modo de video utilizando el servicio 15

Servicio 19 (13 hex): Escritura de una cadena de caracteres



El servicio 19, disponible sólo en el AT, nos permite escribir una cadena de caracteres en la pantalla. A través de los cuatro subservicios que componen este servicio, se pueden especificar los atributos del carácter, individualmente o como un grupo. También se puede mover el cursor hasta el final de la cadena o dejarlo en el sitio, según el subservicio que se elija.

El número de subservicio se sitúa en AL, el puntero de la cadena en CX, la posición donde va a ser escrita la cadena en DX y el número de página de visualización en BH.

Los subservicios 0 y 1 escriben una cadena de caracteres en la pantalla, utilizando el atributo especificado en el registro BL. Con el subservicio 0, el cursor no se mueve de la localización especificada en el registro DX; con el subservicio 1, el cursor se mueve a la localización siguiente al último carácter de la cadena.

Los subservicios 2 y 3 escriben una cadena de caracteres y atributos en la pantalla, tomando primero el carácter y luego su atributo. Con el subservicio 2, el cursor no se mueve de la localización especificada en el registro DX; con el subservicio 3, el cursor no se mueve de la localización siguiente al último carácter de la cadena.

COMENTARIOS Y EJEMPLO

Repasando cómo trabajan individualmente, hemos visto todos los servicios de video de la ROM-BIOS. Teniendo presente dicha información, la siguiente pregunta usualmente es: teniendo la posibilidad de utilizar directamente los servicios de la ROM-BIOS o utilizar los servicios de más alto nivel, tales como los servicios del DOS o los servicios implementados en su lenguaje de programación, ¿qué es lo mejor? El consejo general que siempre damos es utilizar los servicios de más alto nivel que realicen lo que se quiera hacer. En este caso no hay una razón específica para que evite utilizar los servicios de video de la ROM-BIOS; no se puede hacer gran daño si se utilizan. Pero en el próximo capítulo sobre los servicios del *diskette* trataremos el caso de otra forma, aconsejándole que evite utilizar los servicios del *diskette* de la ROM-BIOS, puesto que hay mayor riesgo asociado con ello.

Las prestaciones de video de los modelos del PC son notables, y los servicios de la ROM-BIOS nos proporcionan un uso completo de ellas. Los servicios del DOS, como veremos en los capítulos 14 al 18, son más pobres y proporcionan sólo la clase más simple de servicios

de tratamiento de caracteres. Del mismo modo, algunos lenguajes de programación (por ejemplo, Pascal y C) sólo proporcionan una versión disfrazada de los servicios del DOS, nada más. Así, si necesita utilizar todas las prestaciones de la pantalla del PC y no está utilizando un lenguaje tal como el BASIC que proporciona los servicios que necesita, deberá de utilizar los servicios de la ROM-BIOS. El control de la pantalla de visualización es una de las mejores razones para utilizar los servicios de la ROM-BIOS.

El uso de los servicios de la ROM-BIOS directamente suele necesitar un interfaz en lenguaje ensamblador; así pues, le daremos un ejemplo de cómo se puede hacer. Para nuestro ejemplo, pondremos en marcha un módulo en un formato que pueda ser llamado por el Pascal. Haremos que el módulo cambie a modo 1 (texto de 40 columnas en color) y ponga el color del fondo en azul.

Aquí está nuestro módulo en ensamblador (véase capítulo 8, página 186 para las notas generales sobre el formato):

```

MODULO    SEGMENT 'CODE'
          PUBLIC  AZUL40
AZUL40    PROC    FAR
          PUSH    BP                ; salva el puntero de base antiguo
          MOV     BP,SP            ; establece nuestro puntero de base
; fija el modo de video
          MOV     AH,0              ; servicio 0: ajuste de modo
          MOV     AL,1              ; modo 1: texto de 40 columnas en color
          INT     16                ; petición del servicio de video
; fija el color de fondo
          MOV     AH,11             ; servicio 11: fija color
          MOV     BH,0              ; fija fondo
          MOV     BL,1              ; color 1 = azul
          INT     16                ; petición del servicio de video
          POP     BP                ; realmacenar el antiguo puntero de base
          RET     0                 ; vuelta al usuario
AZUL40    ENDP
MODULO    ENDS
          END

```


Servicios para disco de la ROM-BIOS

Servicios estándar de la ROM-BIOS para *diskette* 210

Servicio 0: Reinicialización del sistema de *diskette* 211

Servicio 1: Obtención del estado del *diskette* 211

Servicio 2: Lectura de sectores del *diskette* 212

Servicio 3: Escritura de sectores del *diskette* 214

Servicio 4: Verificación de sectores del *diskette* 214

Servicio 5: Formateo de pistas del *diskette* 215

Servicios para disco duro del AT 217

Servicio 8: Obtención de los parámetros de la unidad activa 217

Servicio 9: Inicialización de las tablas de parámetros del disco duro 218

Servicios 10 y 11 (A y B hex): Lectura y escritura de sectores largos 218

Servicio 12 (C hex): Busca de cilindro 218

Servicio 13 (D hex): Reinicialización alternativa del disco 218

Servicio 16 (10 hex): Comprueba si la unidad está lista 218

Servicio 17 (11 hex): Recalibra la unidad 218

Servicio 20 (14 hex): Diagnóstico del controlador 219

Servicio 21 (15 hex): Obtención del tipo de disco 219

Servicio 22 (16 hex): Cambio del estado del disco 219

Servicio 23 (17 hex): Indica el tipo de disco 220

Tabla base del disco 220

Comentarios y ejemplos 222

Vamos a cubrir ahora los servicios de *diskette* proporcionados por el ROM-BIOS. Para comprender la estructura lógica de los contenidos de un *diskette*, véase capítulo 5, y para encontrar información sobre los servicios de *diskette* de más alto nivel proporcionados por el DOS, véanse capítulos 15 al 18.

Hablando de forma general, es mejor dejar las operaciones del *diskette* al sistema operativo. Si decide utilizar cualquiera de los servicios de *diskette* de la ROM-BIOS, le recomiendo que lea primero la sección titulada «Comentarios y ejemplos» de este capítulo, página 222.

SERVICIOS ESTANDAR DE LA ROM-BIOS PARA DISKETTE

Puesto que una unidad de disco desarrolla sólo unas pocas tareas simples, no se necesitan más que seis servicios que son comunes a todos los modelos PC **AT**. El AT, debido a la introducción de una unidad de disco más complicada, ha añadido a la ROM-BIOS otros nuevos servicios. Discutiremos por separado estas ampliaciones más adelante.

Todos los servicios de *diskette* de la ROM-BIOS se invocan con la interrupción 19 (13 hex) y se seleccionan cargando el número de servicio deseado en el registro AH. Los seis servicios estándar, mostrados en la figura 10-1, se numeran del 0 al 5, como es costumbre.

Los servicios del *diskette* operan bajo la supervisión de la tabla base del disco, que es un conjunto de alrededor de una docena de parámetros de control del *diskette*, que están almacenados en ROM y que especifican cosas tales como el tamaño de sector, velocidad de

Servicio	Descripción
0	Reinicialización del sistema de <i>diskette</i>
1	Obtención del estado del <i>diskette</i>
2	Lectura de sectores del <i>diskette</i>
3	Escritura de sectores del <i>diskette</i>
4	Verificación de sectores del <i>diskette</i>
5	Formateo de pistas del <i>diskette</i>

Figura 10-1. Los seis servicios estándar del *diskette* proporcionados por la ROM-BIOS

giro y otros parámetros mecánicos. Para la mayoría de los programadores, la tabla base del disco es una parte invisible de los servicios del *diskette*. Sin embargo, ocasionalmente algunos de sus parámetros se alteran por motivos especiales. Por esta razón se incluyen una breve descripción hacia el final de este capítulo.



□ **NOTA:** *Mientras que los servicios del diskette de la ROM-BIOS para el PCjr son idénticos a los de los otros modelos de ordenadores personales de IBM, las prestaciones de las unidades de diskette del PCjr son radicalmente diferentes. Esto se debe principalmente a que los controladores de diskette del Junior no utilizan acceso directo a memoria (DMA), que permite que los datos sean transferidos directamente entre el diskette y la memoria. En su lugar, el software de la ROM-BIOS hace el trabajo de transferencia, lo que interrumpe la posibilidad de cálculo del PCjr durante las operaciones del diskette, haciendo que los tiempos y prestaciones de los servicios del diskette sean diferentes de los de los otros modelos. Entre otras cosas, esto significa que algunos esquemas de protección de copia diseñados en los otros modelos no operan satisfactoriamente en el Junior.*

Servicio 0: Reinicialización del sistema de diskette

El servicio 0 se utiliza para reinicializar el controlador y la unidad del *diskette*. Este servicio no afecta al *diskette* cargado en las unidades de discos del ordenador. En cambio, esta operación a través del servicio 0 fuerza a las rutinas de soporte del *diskette* de la ROM-BIOS a comenzar recalibrando la cabeza de lectura/escritura de la unidad del *diskette* (operación que posiciona la cabeza en una cierta pista). En nuestros programas, este servicio se utiliza normalmente después de que se haya producido un error en cualquier otra operación de la unidad de disco.

Servicio 1: Obtención del estado del diskette

El servicio 1 devuelve el estado del *diskette* en los ocho bits del registro AL. El estado se almacena después de cada operación del *diskette* incluyendo la lectura, escritura, verificación y las operaciones de formateo descritas más adelante. Este almacenamiento del estado del *diskette* permite que una rutina de control del error dé información de ese error y sea completamente independiente de las rutinas que controlan el *diskette*. Esto puede ser muy útil, ya que bajo ciertas circunstancias se puede confiar en el DOS, o en nuestro lenguaje de programación, para gobernar el *diskette* (resulta una elección muy

acertada; ➤ Véase el párrafo «Comentarios y ejemplos» de este mismo capítulo, página 222), y al mismo tiempo facilita que nuestros programas encuentren e informen en detalle de los errores. ➤ Véase la figura 10-2 para encontrar más información sobre el byte de estado.

Servicio 2: Lectura de sectores del diskette

El servicio 2 lee uno o más sectores del *diskette* y los pasa a la memoria. Si queremos leer más de un sector, todos los sectores deben estar en la misma pista y en la misma cara. Esto es necesario porque la ROM-BIOS no sabe identificar cuántos sectores hay en la pista, y por tanto no sabe cuándo cambiar de una cara o pista a otra. Normalmente, este servicio se utiliza para leer sectores individuales, o una cara entera en el caso de operaciones en masa, tales como el DISKCOPY del DOS. Se utilizan varios registros para controlar la información durante una operación de lectura, ➤ están expuestos en la figura 10-3.

DL contiene el número de la unidad.

DH contiene la cara del *diskette* o el número de cabeza de lectura/escritura (0 ó 1).

Bit								Valor	Significado
7	6	5	4	3	2	1	0		
1	.	.	.	.	.	.	.	120	Fuera de tiempo: no responde el disco
.	1	.	.	.	.	.	.	64	Búsqueda incorrecta: no ha sido posible desplazarse a la pista requerida
.	.	1	.	.	.	.	.	32	Fallo del controlador del <i>diskette</i>
.	.	.	1	.	.	.	.	6	Mal CRC: datos leídos sin sentido lógico o dañados
.	.	.	.	1	.	.	.	8	Fallo del DMA (no se aplica al PCjr)
.	.	.	.	.	1	.	.	4	Mal sector: el sector requerido no se encuentra en el <i>diskette</i>
.	.	.	.	.	.	1	.	2	Mala marca de dirección: marca identificadora del sector invalidada o no encontrada
.	.	.	.	.	.	.	1	1	Mal comando: solicitud no válida al controlador del <i>diskette</i>
.	.	.	.	1	.	.	1	9	Error de límites en DMA: intento de DMA, que cubre más de 64K (no se aplica al PCjr)
.	.	.	.	.	.	1	1	3	Error de protección de escritura: intento de escribir en un disco protegido

Figura 10-2. Codificación del byte de estado devuelto en el registro AL por el servicio 1

CH contiene el número de pista. El número de pista tiene normalmente un valor de 0 al 39 (00 al 27 hex), pero puede ser más alto, y es más alto en algunos sistemas de protección contra copia. Algunas unidades de *diskette* trabajan satisfactoriamente con más de 42 pistas.

CL contiene el número de sector. Normalmente, el número de sector está comprendido en un rango de 0 a 8 ó 9, aunque se utilizan a veces números de sector mayores de 9 en sistemas de protección contra copia. Note que los sectores se numeran a partir de 1, a diferencia de las unidades de disco, pistas o cabezas (caras).

AL contiene el número de sectores que se van a leer. Esto es normalmente desde 1 hasta 8 ó 9. IBM advierte de no solicitar 0 sectores.

ES:BX contienen la localización del *buffer*. La localización del área de memoria donde serán situados los datos está proporcionada por una dirección segmentada dada en este par de registros. El par de registros ES:BX se utiliza normalmente para todas las direcciones segmentadas de los servicios de la ROM-BIOS.

El área de datos debería ser lo suficientemente grande como para acomodar tantos datos como sean leídos; tenga en cuenta que mientras los sectores normales del DOS son de 512 bytes, los sectores pueden ser de hasta 1024 bytes (☛ véase el servicio de formateo que sigue). Cuando este servicio lee más de un sector, dispone los sectores en la memoria uno tras del otro.

CF contiene el informe de error de la operación. El resultado de la operación se devuelve a través de una combinación del indicador de arrastre (CF) y del registro AH. Si CF es 0, significa que no había error y AH también será 0. Si CF es 1, significa que había un error y AH contiene los bits de estado detallados por el servicio 1, que es el servicio de estado.

Cuando utilice el servicio 2, o cualquier otro servicio activo del *diskette*, recuerde que el motor de la unidad de *diskette* tarda algún tiempo en alcanzar la velocidad de trabajo y que ninguno de estos servicios espera a que esto ocurra. Aunque mi propia experiencia con los servicios de *diskette* de la ROM-BIOS sugiere que esto es un problema raro, IBM recomienda que cualquier programa que utilice estos servicios lo intente durante tres veces antes de suponer que un error es real y que utilice el servicio de reinicialización entre los intentos. La lógica de la operación sugerida es como sigue:

```
10 NUMERO.ERRORES = 0
20 WHILE NUMERO.ERRORES < 3
30 ' realice una operación de lectura/escritura/verificación/formateo
40 ' aquí compruebe el error: si no hay error goto 90
```

```

50  NUMERO.ERRORES = NUMERO.ERRORES + 1
60  ' haga una operación de reinicialización
70  WEND
80  ' actúe sobre el error
90  ' continúe después del éxito

```

Servicio 3: Escritura de sectores del diskette

El servicio 3 escribe uno o más sectores en un *diskette*; es el inverso de servicio 2. Todos los registros, detalles y comentarios dados para el servicio 2 se aplican al servicio 3. (● Véase también la figura 10-3.) Los sectores del *diskette* deben de estar formateados antes de que se pueda escribir en ellos.

Servicio 4: Verificación de sectores del diskette

El servicio 4 «verifica» el contenido de uno o más sectores del *diskette*. Esta operación no es lo que alguna gente piensa que es: no se hace la comparación entre los datos del *diskette* y los de la memoria. La verificación que ejecuta este servicio comprueba simplemente que los sectores pueden ser encontrados y leídos, y que el chequeo de redundancia cíclica (CRC) es correcto.

El CRC actúa como un test de paridad sofisticado de los datos de cada sector, y detecta de forma muy fiable los errores, tal como bits perdidos o indescifrables.

Se utiliza más a menudo el servicio de verificación para comprobar los resultados de una operación de escritura, después de la utilización del servicio 3, pero se puede verificar cualquier parte del *diskette* en cualquier momento. Sin embargo, mucha gente contempla la verificación como una operación innecesaria, porque las unidades

Parámetros	Estado que resulta
DL = número de unidad	Si CF = 0 entonces no hay error y AH = 0
DH = número de cara o de cabeza	Si CF = 1 entonces hay error y AH
CH = número de pista	contiene los bits de estado del servicio 1
CL = número de sector	
AL = número de sectores a leer	
ES:BX = dirección del <i>buffer</i>	

Figura 10-3. Registros utilizados para la información de control de los servicios de lectura, escritura, verificación y formateo

de *diskette* son fiables y porque los mensajes de error también lo son. El DOS no verifica la operación de escritura, a menos que se lo indiquemos mediante el comando VERIFY ON.

El servicio de verificación opera con los servicios de lectura y escritura, y utiliza los mismos registros. La única diferencia entre ellos es que la operación de verificación no utiliza el área de memoria y, por tanto, no necesita el par de registros ES:BX.

Servicio 5: Formateo de pistas del diskette

El servicio 5 formatea una pista en una cara del *diskette*. El servicio de formateo opera de forma muy parecida al de los servicios de lectura y escritura, excepto en que el número de sector almacenado en el registro CL no se utiliza. El resto de los parámetros, mostrados en la figura 10-3, pasa y se devuelve a través de los registros.

Puesto que el formateo se hace cada vez en una pista completa, no se pueden formatear sectores individuales. Sin embargo, podemos especificar características individuales para cada sector de una pista.

Cada sector de una pista tiene cuatro bytes descriptivos asociados con él, que están localizados en el área de datos apuntada por el par de registros ES:BX. Estos bytes son las marcas de dirección que se utilizan después para identificar sectores individuales durante las operaciones de lectura, escritura y verificación. Estos cuatro bytes de dirección están referenciados como C para el cilindro (que es el término más general para pista), H para cabeza (o cara de disco), R para registro (o número de sector) y N para número de bytes por sector (también llamado código de tamaño). Debe haber un campo de 4 bytes por cada sector especificado en AL.

Cuando se está leyendo o escribiendo un sector, la ROM-BIOS busca en la pista el ID del sector, cuya parte fundamental es R, el número de registro o sector. Los parámetros del cilindro y cabeza no son en realidad necesarios, puesto que la cabeza de lectura/escritura se sitúa mecánicamente en la pista, y la cara se selecciona electrónicamente, pero son registrados y comprobados por seguridad.

El código de tamaño (N) puede tomar uno de los cuatro valores estándar que aparecen en la figura 10-4. Lo normal es el código 2, correspondiente a 512 bytes.

Los sectores se escriben en el *diskette* en el orden especificado por los bytes de dirección, y no secuencialmente. En efecto, el orden de los sectores puede alterarse para lograr mejores prestaciones, o crear diferencias de temporización con propósitos de protección contra copia. El disco duro del XT tiene sus sectores intercalados, de forma

N	Tamaño del sector (bytes)	Tamaño del sector (K)
0	128	$\frac{1}{8}$
1	256	$\frac{1}{4}$
2	512	$\frac{1}{2}$
3	1024	1

Figura 10-4. Los cuatro tamaños estándar del código de tamaño N

que los sectores consecutivos estén físicamente separados 6 sectores. Los *diskettes* del DOS tienen sus sectores registrados en orden secuencial: 1, 2, 3, etc.

Para un *diskette* del DOS convencional de nueve sectores, las direcciones de formateo para la pista 0, cara 1, serían de la forma siguiente:

C H R N	C H R N	C H R N	...	C H R N
0 1 1 2	0 1 2 2	0 1 3 2	...	0 1 9 2

Cuando se formatea una pista, la unidad del *diskette* pone atención en el agujero de índice del *diskette* y lo utiliza como una marca de comienzo para formatear la pista. El agujero de índice se ignora en todas las otras operaciones (lectura, escritura o verificación), y las pistas se buscan simplemente mediante sus marcas de dirección.

Advierta que nada en este servicio de formateo especifica el valor del dato inicial que se escribe en cada sector formateado. Eso está controlado por la tabla base del disco. (■ Véase página 220).

Utilización del servicio 5 para la protección contra copia

Las pistas pueden formatearse de mil formas diferentes, pero la mayoría de los sistemas operativos sólo pueden leer ciertos formatos. Consecuentemente, la mayoría de los esquemas de protección contra copia está basada en un formato no convencional, que impide al sistema operativo leer y escribir datos satisfactoriamente. Podemos elegir entre varios métodos diferentes de protección de copia:

- Podemos reorganizar el orden de los sectores, de tal manera que se altere el tiempo de acceso en una forma que puedan detectar los esquemas de protección contra copia.

- Podemos comprimir más sectores en una pista (diez es prácticamente el límite superior para sectores de 512 bytes).
- Podemos simplemente omitir el número de sector.
- Podemos añadir un sector con un número extraño (por ejemplo, podemos marcar $R = 22$).
- Podemos especificar que uno o más sectores tengan un tamaño no convencional.
- Podemos registrar valores erróneos de C y H.

Puede utilizarse cualquiera de estas técnicas para protección de copia o para cambiar las características operativas del *diskette*. Dependiendo de qué opciones se utilicen, un *diskette* formateado convencionalmente puede tener sus características no usuales completamente ocultas al DOS, de forma que el mecanismo de protección de copia sea transparente a la detección ordinaria.

SERVICIOS PARA DISCO DURO DEL AT

AT



El AT utiliza unidades de disco tan diferentes de las unidades en los otros modelos que ha sido necesario añadir varios servicios nuevos de la BIOS. Están diseñados para soportar los *diskettes* de alta capacidad y la gran variedad de discos duros que se pueden utilizar con el AT. Daremos indicaciones sobre los nuevos servicios, pero no entraremos en detalle, por la misma razón por la que hemos pasado someramente sobre otras características dependientes del modelo. Nuestro principal objetivo en este libro es explorar los principios generales y prácticas de programación que se aplican a la familia entera del PC, y no las peculiaridades de un modelo u otro.

Servicio 8: Obtención de los parámetros de la unidad activa

El servicio 8 devuelve los parámetros de la unidad del disco: DL informa sobre el número de unidades de disco (de 0 a 2), DH informa sobre el número máximo de cara-cabeza, CH devuelve el número máximo de cilindro-pista, y el CL devuelve el número de sector más alto.

Servicio 9: Inicialización de las tablas de parámetros del disco duro

El servicio 9 se utiliza para crear las tablas base del disco para las dos unidades de disco duro. Los vectores de interrupción de las interrupciones 65 (41 hex) y 70 (46 hex) se utilizan para proporcionar las direcciones de la tabla. Este servicio se utilizaría sólo para instalar una unidad de disco «extraña».

Servicios 10 y 11 (A y B hex): Lectura y escritura de sectores largos

El servicio 10 lee y el servicio 11 escribe sectores largos en discos duros de 20 megabytes. Un sector largo incluye un ECC, que es un código de error de 4 bytes que proporciona verificación de errores de alto nivel y corrección de errores para los datos del sector.

Servicio 12 (C hex): Busca de cilindro

El servicio 12 realiza una operación de búsqueda que posiciona las cabezas de lectura/escritura del disco sobre un cilindro particular del disco duro. El registro DL proporciona el identificador de la unidad, DH el número de cabeza y CH el número de cilindro.

Servicio 13 (D hex): Reinicialización alternativa del disco

El servicio 13 realiza una operación de reinicialización del disco alternativa, utilizable con los discos duros. La unidad se especifica en el registro DL. El servicio funciona igual que el servicio 0 para *diskettes*.

Servicio 16 (10 hex): Comprueba si la unidad está lista

El servicio 16 comprueba si la unidad del disco duro está lista. La unidad se especifica en el registro DL y el estado se devuelve en el registro AH.

Servicio 17 (11 hex): Recalibra la unidad

El servicio 17 recalibra las unidades de disco duro. La unidad se especifica en el registro DL y el estado se devuelve en el registro AH.

Servicio 20 (14 hex): Diagnóstico del controlador

El servicio 20 invoca una rutina de diagnóstico interno del controlador de disco del AT. El estado del controlador se devuelve en el registro AH.

Servicio 21 (15 hex): Obtención del tipo de disco

El servicio 21 se utiliza para conocer el tipo de unidad de disco instalada. Dado el identificador de la unidad de disco en el registro DL, devuelve en el registro AH uno de estos cuatro indicadores de tipo de disco: si AH es 0, significa que no hay ninguna unidad presente; si AH es 1, indica la presencia de una unidad de *diskette* que no puede detectar cuando ha sido cambiado el disco (típica de la mayoría de las unidades de disco); si AH es 2, indica la presencia de una unidad de disco que puede detectar un cambio de disco (unidades como las de *diskette* de alta capacidad del AT), finalmente, si AH es 3, significa que está instalada una unidad de disco duro. Cuando el tipo de unidad es 3, el par de registros CX:DX actúa como un entero de 4 bytes, que da el número total de sectores de disco en la unidad.

Servicio 22 (16 hex): Cambio del estado del disco

El servicio 22 se utiliza para preguntar sobre un cambio de discos para las unidades que pueden detectarlo, como es el caso de las unidades de disco de alta capacidad del AT. El registro AH se pone a 0 para indicar que no ha habido cambio de disco y a 6 para indicar un cambio de disco. El registro DL devuelve el número de la unidad que ha cambiado el disco.

Los servicios 21 y 22 son muy útiles para programas que necesitan saber si ha sido cambiado un disco. Para ciertas operaciones del disco críticas, tales como la lectura de la tabla de localización de ficheros (FAT), ayuda conocer si el disco ha sido cambiado o no. Si se ha cambiado, los datos del disco almacenados en la memoria deben de desecharse y volverse a leer. Cuando una unidad de disco no puede informar de un cambio de *diskette*, el programa tiene que suponer que ha sido cambiado y reaccionar de forma lógica, a costa de la eficiencia del programa. Si estamos diseñando programas que controlen una unidad de disco, es claramente útil y más eficiente poder chequear esta clase de información.

Servicio 23 (17 hex): Indica el tipo de disco

El servicio 23 se utiliza para indicar la combinación del *diskette* y la unidad para el AT. Si AL es 0, no hay unidad de disco; si AL es 1, indica un *diskette* normal en una unidad normal; si AL es 3, indica un *diskette* de alta capacidad en una unidad de alta capacidad. Este servicio se utiliza con el servicio de formateo (servicio 5) para indicar el tipo de disco a ser formateado.

TABLA BASE DEL DISCO

Las operaciones de la unidad de *diskette* están controladas por el conjunto de parámetros llamados tabla base del disco. Aunque una versión por defecto de la tabla base del disco está almacenada en la ROM, en la dirección prácticamente estándar F000:EFC7 podemos crear una nueva tabla. La nueva tabla puede utilizarse situándola en la memoria ordinaria y cambiando el vector de interrupción de la tabla base del disco. Cada versión del DOS desde la primera versión 1.00 ha creado su propia tabla base del disco en lugar de utilizar la que se encuentra en la ROM.

La tabla base del disco está compuesta por los once bytes de la figura 10-5. La examinaremos byte por byte, y compararemos los valores utilizados por el DOS versión 2.10 con las tablas por defecto en la versión original del PC. La mayoría de la información almacenada en la tabla base del disco es muy poco útil para nosotros,

Dirección relativa	Uso
0	Byte de especificación 1: datos de tiempo SRT y de carga de cabeza
1	Byte de especificación 2: tiempo de carga de cabeza, modo DMA
2	Tiempo de espera hasta que se apaga el motor
3	Bytes por sector: 0 = 128; 1 = 256; 2 = 512; 3 = 1024
4	Número del último sector
5	Longitud del intervalo entre sectores para operaciones de lectura/escritura
6	Longitud de datos cuando no está especificada la longitud del sector
7	Longitud del intervalo entre sectores para operaciones de formateo
8	Valor del dato almacenado en sectores formateados
9	Tiempo de estabilización de la cabeza
A	Tiempo de puesta en marcha del motor

Figura 10-5. Uso de los once bytes de la tabla base del disco

a menos que planeemos escribir una nueva tabla base del disco para sustituir a la utilizada por el DOS.

Los bytes 0 y 1 están referenciados como bytes de especificación. Son parte de las cadenas de comandos enviadas al controlador del disco flexible (FDC), que es también conocido como controlador NEC (Nippon Electric Company). Los primeros cuatro bits del primer byte son el tiempo SRT (Step-rate time), que es el tiempo que la ROM-BIOS indica para que la unidad de *diskette* se mueva de una a otra pista. El valor por defecto es de 8 milisegundos para cada pista. El DOS 2.10 reduce esto a 6 milisegundos, lo que aumenta las prestaciones en velocidad de la unidad de disco. Hay también un ajuste para el DMA en el byte uno, porque el PCjr no tiene DMA para transferir los datos del disco. Como resultado, el bit de modo del DMA no está puesto a uno en la tabla base del disco por defecto del PCjr. (Es un misterio para mí el porqué esto no produce problemas en el Junior.)

El byte 2, dirección relativa 2, especifica cuánto tiempo debe dejar correr el motor del *diskette* después de cada operación. El valor está en unidades de tics de reloj (alrededor de 18 tics son un segundo). Todas las versiones de la tabla tienen este byte puesto a 37 (25 hex), lo que significa que el motor espera alrededor de 2 segundos.

El byte 3, dirección relativa 3, da el código de longitud del sector —el mismo código N que se utiliza en la operación de formateo (véase el servicio 5, página 215)—. Este está puesto normalmente a 2, representando la longitud de sector acostumbrada de 512 bytes. En cualquier operación de lectura, escritura, o verificación, el código de longitud en la tabla base del disco debe ponerse con el valor adecuado, especialmente cuando se trabaja con sectores de longitud no convencional.

El byte 4, dirección relativa 4, da el número del último sector de la pista. Este valor es 8 en la tabla por defecto, de la ROM, y 9 en la tabla del DOS 2.10.

El byte 5, dirección relativa 5, especifica el tamaño del intervalo entre sectores, que se utiliza en la lectura y escritura de datos. En efecto, le dice a la ROM-BIOS cuánto tiene que esperar antes de buscar la siguiente marca de dirección del sector, así puede evitar mirar zonas sin sentido en el *diskette*. En cada base de disco estándar el tamaño de intervalo está a 42 (2A hex).

El byte 6, dirección relativa 6, se llama longitud de transferencia de datos (DTL), y está a 255 (FF hex). Este byte contiene la longitud de datos máxima cuando no está especificada la longitud del sector.

El byte 7, dirección relativa 7, da el tamaño del intervalo entre sectores cuando se formatea una pista. Naturalmente es mayor que el

intervalo de búsqueda, byte 5. El valor normal de este byte es 80 (50 hex).

El byte 8, dirección relativa 8, proporciona el dato que será almacenado en cada byte de los sectores cuando se formatee una pista. El valor estándar es F6 (el símbolo de división). Podemos cambiarlo al valor que queramos si creemos tener una buena razón para hacerlo.

El byte 9, dirección relativa 9, indica el tiempo de estabilización de la cabeza, que es el tiempo que debe de esperar el sistema para que termine la vibración, después de buscar una nueva pista. El tiempo por defecto para la estabilización de la cabeza es 25 (19 hex) milisegundos, pero el DOS 2.10 lo reduce a 15 (F hex) milisegundos.

El byte 10 es el byte final de la base del disco, y tiene dirección relativa 10, indica el tiempo que tarda el motor de la unidad del *diskette* en tomar velocidad, y está medida en octavos de segundo. El valor por defecto es 4, que significa $\frac{1}{2}$ segundo; el DOS 2.10 cambia esto a 2; es decir, $\frac{1}{4}$ segundo.

Es divertido jugar con los valores de la base del disco, hay suficientes de ellos para darnos una oportunidad de sentir toda clase de excitación. El siguiente programa ilustra cómo cambiar el valor de los datos que se almacenan en un sector cuando es formateado. Lo cambiaremos de F6 hex a AA hex para mostrar cómo se hace.

```

10 DEF SEG
20 DIRELATIVA = PEEK(120+0)+256*PEEK(120+1) ' dirección relativa de la base
                                           del disco
30 SEGMENTO = PEEK(120+2)+256*PEEK(120+3) ' segmento de la base
                                           del disco
40 DEF SEG = SEGMENTO
50 ' si el segmento es muy alto, la base se encuentra en ROM
60 IF SEGMENTO >= &HF000 THEN PRINT "base de disco en ROM"
70 DATO.FORMATEO = PEEK(DIRELATIVA+8) ' obtener el dato antiguo
80 PRINT "El valor de formateo era " HEX$(DATO.FORMATEO)
90 POKE DIRELATIVA+8, &HAA ' cambia el valor de los datos
                             a hex AA

```

COMENTARIOS Y EJEMPLOS

En el capítulo anterior, donde tratamos de los servicios de video de la ROM-BIOS, recomendé que hiciera uso directo de los servicios de la ROM-BIOS siempre que lo deseara. Pero en el caso de los servicios del *diskette*, las cosas son diferentes.

Para las operaciones del *diskette* que normalmente quiera realizar, le recomiendo que deje al DOS la manipulación y supervisión de los *diskettes* y que la realice a través de las sentencias de fichero convencionales de los lenguajes de programación, o a través de los

servicios del DOS (véanse los capítulos 14 al 18). Hay algunas razones para esto. La razón principal es que es más rápido dejar que el DOS haga el trabajo. Los recursos del DOS cuidan de casi todas las funciones básicas del *diskette*, desde la interpretación del formato del *diskette*, hasta el redireccionamiento de los datos con el comando ASSIGN. La mayoría de las veces no es necesario profundizar en el *software* del sistema. Sin embargo, hay veces en que queremos trabajar con los contenidos del *diskette* de forma absoluta y precisa, normalmente para la protección contra copia. En estas ocasiones es cuando se utilizarían los servicios de la ROM-BIOS.

En nuestro ejemplo utilizaremos Pascal para escribir una pareja de subrutinas, que leerán y escribirán sectores de *diskette*. Si no está familiarizado con el Pascal y no quiere descifrar esta rutina, puede obviarla, y aún así podrá entender el ejemplo de interfaz en lenguaje ensamblador que le sigue:

```
PROGRAM INTERFAZ_DISKETTE;
{definición del área de lectura/escritura de segmento: }
TYPE
  TIPO_SEGMENTO = ARRAY [0..511] OF BYTE;
VAR
  DATOS_SEGMENTO : TIPO_SEGMENTO;
{definición de las rutinas de lectura/escritura en ensamblador: }
FUNCTION LEESEG (
  VAR S : TIPO_SEGMENTO; {área de datos}
  D : INTEGER;           {número de unidad}
  C : INTEGER;           {número de pista}
  H : INTEGER;           {número de cabeza}
  R : INTEGER )          {número de segmento}
  : BYTE;                {código de estado devuelto como un byte}
EXTERNAL;
FUNCTION ESCRIBESEG (
  VAR S : TIPO_SEGMENTO; {área de datos}
  D : INTEGER;           {número de unidad}
  C : INTEGER;           {número de pista}
  H : INTEGER;           {número de cabeza}
  R : INTEGER )          {número de segmento}
  : BYTE;                {código de estado devuelto como un byte}
EXTERNAL;
{este breve programa lee el segmento del registro de arranque del
disco: }
BEGIN
  IF LEESEG ( DATOS_SEGMENTO, 0, 0, 0, 1 ) = 0
  THEN
    {no error}
  ELSE ;
    {error}
  END.
```

Con estas dos rutinas está acabada la primera fase y podemos ir a la rutina de interfaz en lenguaje ensamblador. La forma de la

rutina de interfaz debe ser familiar a alguien que haya leído las observaciones generales del capítulo 8, página 184, o haya estudiado el ejemplo del capítulo 9, página 206.

En cada uno de estos ejemplos tratamos de mostrar cosas nuevas. Aquí tenemos dos rutinas en lenguaje ensamblador distintas combinadas en una. En este caso, las dos rutinas son idénticas, excepto por sus nombres y el código del servicio de la ROM-BIOS que utilizan. Advertirá también que los métodos utilizados para obtener los parámetros de la pila utilizan direccionamiento de la forma $[BP] + x$. En el caso de este interfaz, el primer parámetro de la pila es una dirección relativa, mientras que los otros son valores (número de unidad, etc.). Incluso aunque el primer parámetro es una dirección y no un valor, lo manejamos de la misma forma, porque estamos interesados en la dirección y no en su contenido. Una vez tenemos la dirección, simplemente la entregamos al servicio de la ROM-BIOS.

Esta es la rutina interfaz. Tiene dos procedimientos prácticamente idénticos para la lectura y excritura.

```
INTERFAZ    SEGMENT 'CODE'
PUBLIC      LEESEG
PUBLIC      ESCRIBESEG
; éste es el servicio de lectura:
LEESEG      PROC      FAR
            PUSH      BP
            MOV        BP,SP
            PUSH      DS                ; mueve DS...
            POP       ES                ; ...a ES
            MOV        BX,[BP+14]      ; obtener la dirección relativa de los datos
            MOV        DL,[BP+12]      ; obtener el número de unidad
            MOV        CH,[BP+10]      ; obtener el número de pista
            MOV        DH,[BP+08]      ; obtener el número de cara
            MOV        CL,[BP+06]      ; obtener el número de sector
            MOV        AL,1            ; pide un sector
            MOV        AH,2            ; pide el servicio de lectura
            INT        19              ; solicita el servicio de diskette
            MOV        AL,AH          ; pone el byte de estado donde se espera
            POP        BP
            RET        10              ; 10 es el tamaño de los parámetros en el stack
LEESEG      ENDP
; éste es el servicio de escritura (sólo difiere el número de servicio)
ESCRIBESEG  PROC      FAR
            PUSH      BP
            MOV        BP,SP
            PUSH      DS                ; mueve DS...
            POP       ES                ; ...a ES
            MOV        BX,[BP+14]      ; obtiene la dirección relativa de los datos
            MOV        DL,[BP+12]      ; obtener el número de unidad
            MOV        CH,[BP+10]      ; obtener el número de pista
```

```
MOV     DH,[BP+08]    ; obtener el número de cara
MOV     CL,[BP+06]    ; obtener el número de sector
MOV     AL,1          ; pedir un sector
MOV     AH,3          ; pedir el servicio de escritura
INT     19            ; solicitar el servicio de diskette
MOV     AL,AH         ; poner el byte de estado donde se espera
POP     BP
RET     10            ; 10 es el tamaño de los parámetros en el stack

LEESEG  ENDP
INTERFAZ ENDS
END
```


Servicios para el teclado de la ROM-BIOS

Acceso a los servicios de teclado 228

Servicio 0: Lectura del siguiente carácter del teclado 228

Servicio 1: Informa si hay caracteres en el *buffer* 229

Servicio 2: Lectura del estado de *shift* 229

Comentarios y ejemplos 230

Aunque los servicios de la ROM-BIOS para el teclado no son tan numerosos ni tan complicados como los de pantalla (capítulo 9) o de la unidad de *diskette* (capítulo 10), los servicios de teclado de la ROM-BIOS son lo suficientemente importantes para justificar el incluirlos en un capítulo propio. Los demás servicios de la ROM-BIOS se recogen en el capítulo 12.

ACCESO A LOS SERVICIOS DE TECLADO

A los servicios de teclado se accede con la interrupción 22 (16 hex). Hay tres servicios numerados del 0 al 2. Como todos los demás servicios de la ROM-BIOS, los servicios de teclado se seleccionan en el registro AH.

Servicio 0: Lectura del siguiente carácter del teclado

El servicio 0 devuelve el siguiente carácter de la entrada del teclado. Si hay un carácter en el *buffer* de teclado, se pasa inmediatamente. Si no, el servicio espera hasta que lo haya. Como se describe en el capítulo 6, cada carácter del teclado se entiende como un par de bytes, que llamamos byte principal y auxiliar. El byte principal se devuelve en AL, es o bien 0 para caracteres especiales (así como las teclas de función) o bien, en otros casos, su código para caracteres ASCII ordinarios. El byte auxiliar se devuelve en AH, es o bien el carácter identificador de caracteres especiales o bien código estándar de exploración del teclado del PC para caracteres ASCII.

Si no hay ningún carácter en el *buffer* de teclado cuando el servicio 0 es llamado, el servicio espera hasta que lo haya, lo que esencialmente congela el programa. El siguiente servicio que discutiremos permite que un programa lea el teclado sin el riesgo de detener la ejecución del programa.

Servicio	Descripción
0	Lectura de un carácter del <i>buffer</i> de teclado
1	Información de si hay algún carácter en el <i>buffer</i>
2	Consecución del estado shift

Figura 11-1. Los tres servicios del teclado de la ROM-BIOS

De manera contraria a lo que algunas versiones del «Manual de Referencia Técnica del IBM» sugieren, los servicios 0 y 1 se aplican tanto a los caracteres ASCII ordinarios como a los caracteres especiales, tales como las teclas de función.

Servicio 1: Informa si hay caracteres en el buffer

El servicio 1 informa si un carácter de entrada de teclado está listo. Esta es una operación de comprobación. Se informa de que existe un carácter en el *buffer*, pero éste permanece hasta que sea retirado por el servicio 0. El indicador de cero (ZF) se utiliza como señal. 1 indica que la entrada no está preparada, 0 indica hay un carácter preparado. Hay que tener cuidado en no confundirse con la inversión aparente del significado del indicador, 1 significa No, y 0 en este caso significa Sí. Cuando hay un carácter (ZF=0), se indica cuál en AL y AH, al igual que con el servicio 0.

Este servicio es particularmente útil para dos operaciones de programación realizadas comúnmente. Una es prueba-y-sigue, donde un programa comprueba el teclado y sigue la ejecución si no se ha pulsado ninguna tecla. Normalmente, esto se realiza para interrumpir un proceso por una pulsación de tecla. La otra operación común es el borrado del *buffer* del teclado. Puede ser interesante que los programas permitan teclear instrucciones por adelantado, aunque en determinadas operaciones (comprobaciones de seguridad) no resulte muy aconsejable. En estas circunstancias, nuestros programas necesitan vaciar el *buffer* del teclado, limpiándolo de cualquier entrada. El *buffer* se descarga por el uso de los servicios 0 y 1, como este programa muestra:

```
10  ' llamar al servicio 1, para probar si el carácter está listo
20  WHILE ZF = 0
30  ' llamar al servicio 0, para borrar el carácter
40  ' llamar al servicio 1, para probar si hay otro carácter
50  WEND
```

Contrariamente a lo que sugieren algunos manuales de referencias técnicas, los servicios 0 y 1 se aplican tanto a caracteres ASCII ordinarios, como a caracteres especiales, así como a teclas de función.

Servicio 2: Lectura del estado de shift

El servicio 2 informa del estado shift en el registro AL. El estado shift se toma bit a bit del primer byte de estado del teclado, el cual se

guarda en la localización de memoria hex 417. La figura 11-2 describe cada bit. (Véanse páginas 154 y 160 para más información acerca de otros bytes de estado del teclado, en hex 418 y hex 488.)

Generalmente, el servicio 2 y la información del bit de estado no son particularmente útiles, pero si se está planeando hacer alguna programación especial para teclado puede venir bien. Frecuentemente los verá utilizar en programas que realicen cosas no convencionales, tales como diferenciar entre las teclas shift derecha e izquierda.

Bit								Significado
7	6	5	4	3	2	1	0	
X	.	.	.	.	.	.	.	Estado de inserción: 1 = activo
.	X	.	.	.	.	.	.	CapsLock: 1 = activo
.	.	X	.	.	.	.	.	NumLock: 1 = activo
.	.	.	X	.	.	.	.	Desplazamiento de la pantalla: 1 = activo
.	.	.	.	X	.	.	.	Alt shift: 1 = activo (Alt pulsado)
.	.	.	.	.	X	.	.	Ctrl shift: 1 = activo (Ctrl pulsado)
.	.	.	.	.	.	X	.	Shift normal: 1 = activo (shift de la izquierda pulsado)
.	.	.	.	.	.	.	X	Shift normal: 1 = activo (shift de la derecha pulsado)

Figura 11-2. Los bits de estado del teclado se devuelven en el registro AL cuando se utiliza el servicio 2 del teclado

COMENTARIOS Y EJEMPLOS

Si se encuentra en condiciones de elegir entre los servicios de teclado de su lenguaje de programación, o los servicios de teclado de la ROM-BIOS, pienso que podría utilizar cualquiera de ellos con seguridad y acierto. Mientras que en algunos casos existen argumentos contrarios al uso de los servicios de la ROM-BIOS, como ocurre con los servicios de *diskette*, dichos argumentos no se aplican tan rigurosamente a los servicios de teclado. Sin embargo, como siempre, recomiendo que examine el potencial de los servicios del DOS antes de acudir a los servicios del BIOS; puede encontrar allí todo lo que necesite, y los servicios del DOS son más duraderos y compatibles en el ambiente de «cambio continuo» de los ordenadores personales.

La mayoría de los lenguajes de programación dependen de los servicios del DOS para sus operaciones de teclado, un factor que tiene algunas ventajas claras. La primera ventaja es que los servicios del DOS permiten la utilización de las operaciones estándar de edición del DOS en la entrada de textos (la entrada que no se realiza

hasta que se pulsa la tecla Intro). Si no necesitamos nuestro propio control en la entrada, nos puede ahorrar considerable esfuerzo de programación (y «educación de usuario») utilizar el DOS para la entrada de textos, tanto directamente a través de los servicios del DOS o indirectamente a través de los servicios de su lenguaje. Pero si se necesita un control completo de la entrada al teclado, probablemente acabará usando las rutinas de la ROM-BIOS. De cualquier forma, la elección corresponde a cada cual.

En nuestro ejemplo, en lenguaje ensamblador, de la utilización de los servicios de teclado mostraremos un programa borrador del *buffer* de teclado. Esta rutina realizará la acción detallada en el servicio 1 del teclado: informa si hay algún carácter preparado. Entre las cosas nuevas que ilustrará esta rutina de borrado de *buffers*, la utilización de etiquetas y ramificaciones. Cuando discutimos las generalidades de las rutinas de interfaz en lenguaje ensamblador en el capítulo 8, mencionamos que una sentencia ASSUME CS es necesaria en algunas circunstancias, y se verá aquí uno de ellas.

Siempre que se utilice ASSUME CS, realmente estamos engañando al ensamblador sobre el contenido del registro CS, ya que CS no estará apuntando necesariamente donde nosotros decimos. Este pequeño engaño es completamente inofensivo mientras que todas las instrucciones de la ramificación (tales como JZ, en nuestro ejemplo) sean saltos cortos (NEAR). Esto es debido a que un salto corto se realiza de una forma relativa a la posición del programa que se está ejecutando, el cual se indica por la combinación CS:IP. Para realizar un salto corto, el ensamblador genera una dirección relativa a IP, no a CS, y, por esto, no es realmente necesario conocer CS. De hecho, lo que estamos haciendo es decir al ensamblador que ASUMA un valor concreto en CS, sólo porque el ensamblador es demasiado tonto para darse cuenta que no le hace falta conocerlo. He aquí nuestro ejemplo:

```
; BORRATCL, una rutina para limpiar el buffer del teclado
INTERFAZ  SEGMENT 'CODE'
          PUBLIC  BORRATCL
          ASSUME  CS:INTERFAZ
BORRATCL  PROC    FAR
          PUSH    BP
          MOV     BP,SP
          MOV     AH,1                ; primera prueba de datos
          INT     22
WHILE:
          JZ      SHORT RETURN        ; mientras que ZF = 0
          MOV     AH,0                ; descarte los datos
          INT     22
          MOV     AH,1                ; prueba repetida de datos
```

```
                INT      22
                JMP      SHORT WHILE      ; wend
RETURN:         POP      BP              ; vuelta al programa llamador
                RET
BORRATCL      ENDP
INTERFAZ      ENDS
                END
```

Miscelánea de servicios

Servicios de comunicaciones en serie RS-232 234

Servicio 0: Inicializa los parámetros del puerto serie 235

Servicio 1: Envía un carácter 236

Servicio 2: Recibe un carácter 237

Servicio 3: Obtención del estado del puerto serie 237

Servicios de la cinta de cassette 238

Servicio 0: Pone en marcha el motor del cassette 239

Servicio 1: Para el motor del cassette 239

Servicio 2: Lee bloques de datos 239

Servicio 3: Escribe bloques de datos 240

Ampliación de servicios para el AT 240

Servicios de impresora 241

Servicio 0: Envía un byte a la impresora 242

Servicio 1: Inicializa la impresora 242

Servicio 2: Obtención del estado de la impresora 242

Otros servicios 243

Interrupción 5: Servicio de impresión de la pantalla 243

Interrupción 17 (11 hex): Servicio de listado del equipo 244

Interrupción 18 (12 hex): Servicio de tamaño de memoria 245

Interrupción 24 (18 hex): Servicio cargador del BASIC 246

Interrupción 25 (19 hex): Servicio de arranque (*bootstrap*) 246

Interrupción 26 (1A hex): Servicios horarios 247

En este capítulo trataremos todos los servicios de la ROM-BIOS que no son suficientemente importantes o complejos como para ser tratados en capítulos específicos: en particular, los servicios que se describen son el de comunicaciones en serie RS-232, el de cinta de cassette, las ampliaciones del AT y los de impresora. También describiremos algunos servicios que son lo suficientemente raros como para calificarlos con el nombre de «servicios diversos».

SERVICIOS DE COMUNICACIONES EN SERIE RS-232

Esta sección habla de los servicios de los puertos de comunicaciones en serie asíncronas RS-232 de la ROM-BIOS. Antes de empezar a describirlos con detalle, se deben conocer unas cuantas cosas importantes sobre los puertos de comunicaciones en serie, particularmente en el aspecto terminológico. Suponemos que ya tiene unas nociones básicas sobre transmisión de datos, pero si descubre que no comprende la siguiente información, le recomendamos que busque los conceptos básicos suplementarios en alguno de los muchos libros especializados que hay sobre comunicaciones.

Se emplean muchas palabras para describir el «camino» de entrada y salida de los datos de un ordenador. Una de las más corrientes es «puerto». De cualquier forma, la utilización de esta palabra es completamente diferente de la que le hemos dado previamente. A lo largo del libro, se ha usado el término puerto para referirnos a los caminos direccionables usados por el microprocesador 8088 para comunicarse con otras partes del sistema dentro de los confines de la circuitería del ordenador. Todas las referencias a números de puerto, las sentencias de BASIC INP y OUT, y las operaciones en lenguaje ensamblador IN y OUT hacían referencia a estos puertos direccionables. El puerto de comunicaciones en serie asíncronas RS-232 es diferente, porque es un camino de ENTRADA/SALIDA de propósito general, que se puede utilizar para interconectar muchos tipos de equipos *fuera del ordenador*. Normalmente, los puertos en serie en el PC se utilizan principalmente para telecomunicaciones (es decir, para realizar conexiones telefónicas a través de un *modem*) y también para enviar datos a una impresora tipo serie.

Los servicios de comunicaciones en serie se llaman mediante la interrupción 20 (14 hex). Hay cuatro servicios comunes a todos los modelos IBM. Se numeran de 0 a 5 y se seleccionan a través del registro AH (● Véase figura 12.1).

El diseño original de los ordenadores personales IBM permite añadir hasta siete puertos en serie, aunque es raro emplear más de

Servicio	Descripción
0	Inicializa los parámetros del puerto serie
1	Envía un carácter
2	Recibe un carácter
3	Obtiene el estado del puerto serie

Figura 12-1. Los cuatro servicios de los puertos en serie RS-232 accesibles mediante la interrupción 20 (hex 14)

uno o dos. No importan cuántos puertos serie tenga. El número del puerto serie se especifica en el registro DX. Cuando sólo hay una única puerta en serie, se indica mediante un cero en DX.

Servicio 0: Inicializa los parámetros del puerto serie

El servicio 0 fija los diferentes parámetros RS-232, e inicializa el puerto. Da valores a cuatro parámetros: la velocidad de transmisión (en baudios), la paridad, el número de bits de parada y el tamaño del carácter (también llamado longitud). Los parámetros están combinados en un código de 8 bits, que se sitúa en el registro AL, siguiendo el esquema mostrado en la figura 12-2. La codificación de los bits para indicar las diferentes posibilidades se muestra en la figura 12-3. Cuando ha terminado el servicio, el estado de las comunicaciones queda reflejado en AX, del mismo modo que en el caso del servicio 3 (☛ véase el servicio 3 para obtener más detalles).

☐ NOTA: Aunque es terriblemente lento, 300 baudios es la velocidad más usada por los ordenadores personales que emplean modems para aplicaciones no profesionales. La velocidad de 1200

Bit								Uso
7	6	5	4	3	2	1	0	
X	X	X	.	.	.	.	.	Código de velocidad
.	.	.	X	X	.	.	.	Código de paridad
.	.	.	.	.	X	.	.	Código de bit de parada
.	.	.	.	.	.	X	X	Código de longitud de carácter

Figura 12-2. Orden de los bit de los parámetros del puerto serie, que deben situarse en el registro AL para el servicio 0

la naturaleza del error que se ha producido es usar el informe completo de estado dado por el servicio 3, en vez de fiarse del código de estado devuelto junto con el error en los servicios 1 y 2, ya que este código no es completo.

Servicio 2: Recibe un carácter

El servicio 2 recibe un carácter del canal de comunicaciones especificada en DX y lo coloca en el registro AL. El servicio espera un carácter, o alguna señal, que indique la conclusión del servicio, como, por ejemplo, un exceso en el tiempo de espera (time-out) AH refleja el éxito, o fracaso, del servicio en su bit 7, como ya se ha explicado al hablar del servicio 1. Tenga en cuenta el consejo sobre la manipulación de los errores dado en el párrafo referido al servicio 1 y lea el servicio 3 para analizar los códigos de error.

Servicio 3: Obtención del estado del puerto serie

El servicio 3 devuelve el informe completo del estado del puerto serie en el registro AX. Cada uno de los 16 bits informan individualmente de un posible problema. Los bits de estado se dividen en dos grupos: AH informa del estado de la línea (de lo cual también se informa cuando se producen errores con los servicios 1 y 2) y AL informa del estado del *modem*, cuando resulta pertinente. La figura 12-4 contiene la codificación de los bits de estado. Advertirá que algunos códigos informan sobre errores, mientras que otros simplemente informan sobre una condición.

Bit	Significado	Bit	Significado
7 6 5 4 3 2 1 0	(cuando se pone a 1)	7 6 5 4 3 2 1 0	(cuando se pone a 1)
Registro AH (estado de la línea)		Registro AL (estado del modem)	
1	Error de temporización	1	Detección de portadora
. 1	Registro de desplazamiento vacío	. 1	Indicador de llamada
. . 1	Buffer de transmisión	. . 1	Datos preparados (DSR)
. . . 1	Error en la detección de portadora	. . . 1	Preparado para el envío (CTS)
. . . . 1 . . .	Error de tramas	 1 . . .	Detección de portadora secundaria
. 1 . .	Error de paridad	 1 . .	Detección secundaria de llamada
. 1 .	Error de alcance	 1 .	Datos preparados (DSR). Secundario
. 1	Datos preparados	 1	Preparado para el envío (CTS). Secundario

Figura 12-4. Codificación de los bits para los bytes de estado, colocados en el registro AX por el servicio 3

□ **NOTA:** Hay un asunto especialmente digno de atención sobre el error de temporización (time-out) (AH, bit 7). La primera versión de la ROM-BIOS para el PC original tenía un error de programación que causaba que se produjese informe de un error de temporización (time-out) como si fuera un error conjunto de registro de desplazamiento vacío/interrupción de la detección de portadora cuya combinación era (bits 01010000 en vez de 10000000). Esto se ha corregido en todas las versiones siguientes de la ROM-BIOS, pero ha causado que muchos programas de comunicaciones trataran con escepticismo estos errores. Es posible que usted desee tenerlo en cuenta. ■ Véase página 67 para encontrar más detalles sobre la identificación de la versión de la ROM-BIOS y de los códigos de identificación de máquina.

SERVICIOS DE LA CINTA DE CASSETTE

Los servicios de cinta de cassette se usan cuando se trabaja con la conexión de cassette, que es una posibilidad de algunos modelos PC, como el PC original y el PCjr, pero no el XT, ni el PC portátil, ni el AT, ni el 3270-PC ni otros miembros de la familia PC. Así pues, el puerto del cassette es, en su mayor parte, una creación huérfana—algo que se diseñó para el PC original, suponiendo que sería algo muy solicitado—. Esto no ha sido así y ha permanecido casi totalmente en desuso. El propósito de este puerto de cassette era permitir que se grabaran en cintas de cassette estándar los datos y los programas, especialmente programas en BASIC, como se hace en muchos ordenadores personales baratos que carecen de disco. Nunca he encontrado a la venta un solo programa en cinta de cassette para el PC. De hecho, el único uso del puerto de cassette del que tengo conocimiento es la utilización casera e improvisada como un pobre puerto serie. De todos modos, IBM sigue manteniendo su uso, tanto a través de las rutinas de servicios de la ROM-BIOS, de las que se habla

Servicio	Descripción
0	Pone en marcha el motor del cassette
1	Para el motor del cassette
2	Lee bloques de datos
3	Escribe bloques de datos

Figura 12-5. Los cuatro servicios de la ROM-BIOS para el cassette, activados mediante la interrupción 21 (hex 15)

aquí, como a través del BASIC, que nos ofrece la posibilidad de leer y escribir datos y programas en cinta.

Recuerde que el uso del puerto de cassette conlleva ciertos problemas. El primero de todos es que no todos los modelos PC la tienen. En segundo lugar, pocos PC están equipados con los cables necesarios para conectar un cassette. Y en tercer lugar, el uso de un cassette lleva consigo más intervención manual de la que cabría esperar. Por ejemplo, no puede rebobinarse una cinta bajo el control de un programa.

Los servicios de cassette se llaman con la interrupción 21 (hex 15). Hay cuatro servicios, numerados del 0 al 3. Como siempre, el servicio se especifica en el registro AH. (☛ Véase la figura 12-5.)

Servicio 0: Pone en marcha el motor del cassette

El servicio 0 activa el motor del cassette, lo que no es una operación inmediata de los servicios de la ROM-BIOS, a diferencia de los servicios de disco. Cualquier programa que use este servicio debe realizar un retardo temporal, mientras se espera a que arranque el motor.

Servicio 1: Para el motor del cassette

El servicio 1 desactiva el motor del cassette, y tampoco es una operación inmediata de los servicios de la ROM-BIOS, a diferencia de los servicios de disco.

Servicio 2: Lee bloques de datos

El servicio 2 lee uno o más bloques de datos del cassette. Los datos del cassette se transfieren en bloques de un tamaño estándar de 256 bytes, de forma análoga a los datos de disco que transmiten normalmente sectores con un tamaño estándar de 512 bytes. El número de bytes que se van a leer se sitúa en el registro CX. Aunque los datos se sitúan en la cinta en bloques de 256 bytes, se puede leer o escribir cualquier número de bytes. Por tanto, el número de bytes que se colocan en el registro CX no necesita ser un múltiplo de 256. El par de registros ES:BX se utiliza como un puntero del área de memoria donde se van a situar los datos.

Después de que el servicio se completa, DX contiene el número de bytes leídos, ES:BX apunta al byte que está inmediatamente después del último byte transferido, y el indicador de acarreo (CF) se pone a cero o a uno para anunciar el éxito o el fracaso de la operación. En caso de fallo, AH señala la naturaleza del error usando el código mostrado en la figura 12-6.

Código	Significado
1	Error en el CRC (código de redundancia cíclica)
2	Pérdida de las transiciones de los datos: las señales de los bits son confusos
3	No se encuentran datos en la cinta

Figura 12-6. Códigos de error colocados por el servicio 2 en el registro AH, cuando CF anuncia un fallo producido durante la lectura de los bloques de datos

Servicio 3: Escribe bloques de datos

El servicio 3 escribe uno o más bloques de datos de 256 bytes cada uno. (■ Véase el servicio 2). Como en el caso del servicio 2, el registro CX da el número de bytes a tratar, y ES:BX señala el área de datos en la memoria. Si la cantidad de datos a escribir no es un múltiplo de 256 bytes, el último bloque de datos se utiliza hasta completar el tamaño.

Después de que el servicio se termina, CX se habrá decrementado hasta cero, y ES:BX debe señalar la posición siguiente al último byte de memoria que se escribió.

Curiosamente no hay señales de error para este servicio, esencialmente porque un cassette no es capaz de informar a nuestro ordenador de ninguna dificultad que se presente. Esto obliga a la ROM-BIOS a escribir los datos a ciegas, suponiendo que todo está bien. No es necesario decir que sería una buena idea volver a leer cualquier dato escrito para verificarlo.

AMPLIACION DE SERVICIOS PARA EL AT



Varios de los nuevos servicios de la BIOS, que vamos a indicar ahora, fueron introducidos con el AT para soportar su mayor memoria y algunas de sus características más avanzadas. Se llaman con la interrupción 21 (15 hex), igual que los servicios de entrada/salida (I/O) del cassette, junto con un número de servicio (que varía entre 80 hex y 91 hex) situado en el registro AH. No entraremos en detalles sobre estos servicios en este capítulo, ■ pero le sugerimos que lea el capítulo 13 y el listado del BIOS en el «Manual Técnico de Referencia del AT» para obtener más información.

Servicio (hex)	Descripción
80	Dispositivo abierto
81	Dispositivo cerrado
82	Terminación del programa
83	Espera de suceso
84	Joystick
85	Pulsación de la tecla SysReq
86	Espera
87	Movimiento de bloque
88	Obtención del tamaño de la ampliación de memoria
89	Conmutador a memoria virtual (PRECAUCION: Vea el listado del BIOS antes de usarlo)
90	Bucle del dispositivo ocupado
91	Activación de indicador y realización de interrupción

Figura 12-7. Los doce servicios extendidos para el AT disponibles con la interrupción 21 (15 hex)

SERVICIOS DE IMPRESORA

En los PC estándar, los servicios de impresora sirven sólo para generar salida paralelo. Aun así, en algunos modelos PC la salida de impresora puede ser reconducida a un puerto serie, entre ellos el PCjr.

Los servicios de impresora ROM-BIOS se llaman con la interrupción 23 (17 hex). Hay tres servicios, numerados del 0 al 2, solicitados a través del registro AH. (● Véase la figura 12-8.) El diseño general de la familia PC permite que se instalen más de una impresora, así que su número se debe especificar en el registro DX para todos estos servicios. La impresora número 0 se usa de forma automática por el servicio de impresión de la pantalla. (● Véase página 243.)

Servicio	Descripción
0	Envía un byte a la impresora
1	Inicializa la impresora
2	Obtiene el estado de impresora

Figura 12-8. Los tres servicios de impresora de la ROM-BIOS llamados mediante la interrupción 23 (17 hex)

Servicio 0: Envía un byte a la impresora

El servicio 0 envía un byte a la impresora, situando el byte que va a ser impreso en AL. Cuando se completa el servicio, AH informa del estado de la impresora (véase el servicio 2), que se puede utilizar para determinar el éxito o fracaso de la operación. Véanse las anotaciones especiales sobre el retardo de la impresora dadas en el servicio 2.

Servicio 1: Inicializa la impresora

El servicio 1 inicializa la impresora. Para hacerlo, simplemente envía dos códigos de control (08 y 0C hex) al puerto de control de la impresora (normalmente la 762, 2FA hex). Como con los otros dos servicios, al final se informa del estado de la impresora en AH.

Servicio 2: Obtención del estado de impresora

El servicio 2 informa del estado de la impresora, colocándolo en el registro AH. Los bits, que actúan como códigos individuales, aparecen en la figura 12-9.

El tiempo de retardo de la impresora ha causado algunos problemas en los ordenadores personales IBM. Cualquier controlador de E/S necesita fijar un tiempo límite para esperar una respuesta del mecanismo que se controla. Idealmente, este tiempo no debe ser excesivamente largo, de tal forma que un dispositivo que no responda pueda ser detectado de manera oportuna. Desafortunadamente hay alguna operación normal de la impresora que puede

Bit								Significado (cuando se pone a 1)
7	6	5	4	3	2	1	0	
1	.	.	.	.	.	.	.	Impresora no ocupada (0 = ocupada)
.	1	.	.	.	.	.	.	Señal de reconocimiento producida por la impresora
.	.	1	.	.	.	.	.	Señal de falta de papel
.	.	.	1	.	.	.	.	Impresora seleccionada
.	.	.	.	1	.	.	.	Error de entrada/salida
.	.	.	.	.	1	.	.	No usado
.	.	.	.	.	.	1	.	No usado
.	.	.	.	.	.	.	1	Temporización (time-out)

Figura 12-9. Los bits de estado de la impresora situados en el registro AH por el servicio 2

tardar un tiempo sorprendentemente largo: la expulsión de una página (o un salto al principio de la página siguiente cuando se está al principio de una página). El tiempo permitido varía de versión en versión de la ROM-BIOS. Por tanto, trate con cuidado esta señal.

OTROS SERVICIOS

Vamos a tratar ahora el resto de los servicios ofrecidos por la ROM-BIOS, entre los cuales se encuentran algunos servicios que IBM recomienda su uso, entre ellos el servicio que envía una combinación de caracteres de retorno de carro/salto de línea a la pantalla, y otros que IBM no recomienda (y que yo tampoco lo recomiendo). En esta sección trataremos las seis interrupciones mostradas en la figura 12-10, una por una.

Interrupción 5: Servicio de impresión de la pantalla

La interrupción 5 activa el servicio de impresión de la pantalla. Se utiliza por las rutinas de apoyo al teclado en respuesta a la combinación Shift-PrtSc. Cualquier otro programa que desee realizar una operación de impresión de la pantalla puede hacerlo con seguridad y cómodamente generando la interrupción 5. La subrutina de impresión de la pantalla se desarrolló específicamente para ser controlada por interrupción, así que podemos incorporar este servicio en nuestros propios programas.

El servicio de impresión de la pantalla mantiene la posición actual del cursor de la pantalla y sucesivamente imprimirá cualquier carácter que aparezca en la pantalla, ya sea en el modo de texto o en el modo gráfico. Hace uso de los servicios estándar de video (aquellos que desplazan al cursor alrededor de la pantalla y leen caracteres del

Interrupción		Descripción
Dec	Hex	
5	5	Rutina de impresión de pantalla
17	11	Listado de componentes del equipo
18	12	Tamaño de memoria
24	18	Activación del lenguaje ROM-BASIC
25	19	Activación de la rutina de arranque del sistema
26	1A	Fecha y hora

Figura 12-10. Seis servicios de la ROM-BIOS ofrecidos por IBM, con sus interrupciones asociadas

buffer de pantalla), y también hace uso de los servicios estándar de impresora.

Este servicio dirige todas sus salidas a la impresora número 0, que es la impresora por defecto. No hay registros de entrada ni de salida para este servicio. No obstante, es posible obtener un código de estado localizado en la posición de memoria 500 hex (☛ véase capítulo 3). Si el byte en esa posición tiene un valor de 255 (FF hex), entonces la operación previa de impresión no se ha completado con éxito. Un valor de 0 indica que no hubo error y la operación de impresión está lista para continuar. Un valor de 1 indica que se está realizando la operación de impresión en ese momento; cualquier petición de una segunda operación es ignorada.

Interrupción 17 (11 hex): Servicio de listado del equipo

La interrupción 17 da una información básica del equipo instalado en el ordenador. Es exactamente la misma información que la almacenada en la posición de memoria 410 hex (☛ véase capítulo 3, página 59). La información está codificada tal y como se muestra en la figura 12-11, en los bits de una palabra de 16 bits que se sitúa en el registro AX. ☛ Véase la interrupción 18 que ofrece un servicio complementario.

La información del equipo se recoge de la manera más precisa posible, pero puede que no sea totalmente correcta. Se emplean diferentes métodos para adquirir la información en los diversos modelos.

La lista del equipo se recopila sólo una vez, en el momento de dar corriente, y se guarda en la memoria. Esto quiere decir que podemos cambiar la lista de equipo bajo control del *software*. Por ejemplo, podemos dejar parte del equipo fuera de línea, de manera que no pueda ser utilizado. No obstante, modificar la lista de equipo es una tarea arriesgada (no espere que tenga éxito). ☛ Véase la interrupción 25 en las páginas siguientes, donde podrá encontrar comentarios sobre cómo descomponer la lista del equipo y obtener resultados exactos.

El formato de la lista del equipo está definida para el PC original. Como resultado de esto, algunas partes de la lista no se ajustan a otros modelos. El bit 3 (el bit de impresora en serie) ni se usa ni está definido en todos los modelos IBM anteriores al PCjr. **JR** En el Junior, este bit se activa siempre que las rutinas de puesta en marcha de la ROM-BIOS no encuentran instalada ninguna opción de impresora en paralelo y encuentran conectado al puerto en serie algo razonable.

Bit										Significado
F	E	D	C	B	A	9	8	7	6	
X	X	.	.	.	.	.	.	.	.	Número de impresoras instaladas
.	.	X	.	.	.	.	.	.	.	Impresora en serie: 1 = instalado (sólo PCjr; ni PC ni XT)
.	.	.	X	.	.	.	.	.	.	Adaptador de juego: 1 = instalado (siempre es cierto en el PCjr)
.	.	.	.	X	X	X	.	.	.	Número de puertos serie RS-232
.	.	.	.	.	.	.	X	.	.	Chip DMA: 0 = instalado (no instalado en los PCjr ordinarios)
.	.	.	.	.	.	.	.	X	X	Número de unidades de disco, menos una: 0 = una unidad
.	.	.	.	.	.	.	.	.	X	Modo inicial de video: 10 = color 80 columnas, 11 = monocromo (PC o XT); 01 = color 40 columnas (PCjr)
.	.	.	.	.	.	.	.	.	X	RAM en la placa del sistema: 11 = 64K (normal para todos los modelos)
.	.	.	.	.	.	.	.	.	X	No usado (ajustada a 0)
.	.	.	.	.	.	.	.	.	X	1 si hay discos (en tal caso, véanse los bits 6 y 7)

Figura 12-11. Codificación de los bits para la lista del equipo de la que se informa en el registro AX y que se llama mediante la interrupción 17 (11 hex)

Interrupción 18 (12 hex): Servicio del tamaño de memoria

La interrupción 18 llama a un servicio que informa del tamaño de la memoria libre en Kbytes. Es exactamente la misma información almacenada en la posición de memoria 413 hex (● véase página 246). De este valor se informa en AX. (● Véase la interrupción 17 para encontrar un servicio complementario.)

En los modelos estándar del PC este valor se toma de los ajustes de los conmutadores físicos que hay dentro del sistema. Se supone que estos conmutadores reflejan la memoria instalada realmente, aunque bajo ciertas circunstancias están configurados indicando menos memoria de la que está realmente presente.

En el PCjr, el tamaño de la memoria se determina mediante una exploración *software* durante la puesta en marcha. El Junior ajusta la cantidad de memoria a 16K para poder encontrar aparte la memoria para la pantalla. Cuando un programa usa los modos de video 9 y 10, la pantalla emplea 32K, lo que reduce la memoria útil. No obstante, esto no se refleja en la anotación ROM-BIOS de memoria útil de la que se informa con este servicio. El uso de los modos de video 9 y 10 es un fenómeno transitorio que sólo tiene lugar dentro de la



operación de un programa, y no es un estado permanente. Así pues, hay muy pocas razones para que la ROM-BIOS cambie su registro del tamaño de la memoria cuando se usan estos modos.

Interrupción 24 (18 hex): Servicio cargador del BASIC

La interrupción 24 se usa normalmente para activar la ROM-BASIC. Se ha dispuesto su activación mediante una interrupción, para permitir que el BASIC por defecto sea ignorado. **JR** Esta es la técnica empleada por el PCjr para cargar el cartucho del BASIC, como si fuese el BASIC por defecto, cuando se pone en marcha el equipo, en vez de cargar el BASIC de «cassette» que está incorporado al ordenador.

Cualquier programa, que desee hacerlo, puede activar el BASIC (o lo que lo reemplace) generando la interrupción 24. Esto se puede realizar para cargar el BASIC o, alternativamente, finalizar un programa repentinamente. No obstante, véa la siguiente interrupción, la número 25, para encontrar una forma mejor de abandonar un programa.

Interrupción 25 (19 hex): Servicio de arranque (bootstrap)

La interrupción 25 activa la rutina estándar *bootstrap* del ordenador, que produce un resultado similar a la puesta en marcha, y casi el mismo resultado de la combinación de teclas Ctrl-Alt-Del. No obstante, esta interrupción de *bootstrap* evita la larga rutina de comprobación de la memoria y las operaciones de *reset* de Ctrl-Alt-Del.

Hay dos usos que conozco para este servicio de interrupción. Uno es el inmediato cierre o abandono de las operaciones del ordenador. Esto se puede realizar mediante programa, cuando se encuentra una situación que considera intolerable, como, por ejemplo, una aparente violación de la protección de copia. Muchos programas protegidos para evitar su copia terminan el programa de esta forma cuando detectan algún truco.

El otro uso para esta operación es reactivar el ordenador sin pasar por las operaciones de *reset* y *restart*, que recalcularían el tamaño de la memoria de la lista de equipo que se utiliza en las interrupciones 17 y 18. Esta interrupción es especialmente útil para cualquier programa que pretenda tener en cuenta cualquiera de estos dos detalles. La razón es simple: si deseamos cambiar la lista de equipo o el tamaño de la memoria (por ejemplo, para dejar aparte algo de memoria para un disco RAM), no podemos informar, con seguridad,

todos los programas (incluyendo al DOS) del tamaño de la memoria y el equipo que están usando en cada momento. Pero un programa puede apartar algo de memoria, cambiar las especificaciones de equipo y entonces usar esta interrupción para relanzar el sistema. Cuando se ha hecho esto, y el DOS se activa de nuevo, se tomará el registro de memoria útil del valor ajustado por nosotros con nuestro programa. Ni el DOS ni ningún programa civilizado interferirá con el área de memoria que se ha dejado aparte.

Para dar un breve ejemplo, vea el siguiente fragmento del código ensamblador para cambiar el registro del BIOS del tamaño de la memoria y a continuación emplee la interrupción 25 para relanzar el ordenador.

MOV	AX,40H	; obtener el segmento de datos del BIOS de hex 40...
MOV	ES,AX	; ...y pasado al registro de segmento ES
MOV	WORD PTR ES:19,256	; ajustar la memoria a 256K
INT	25	; relanzar el sistema

Interrupción 26 (1A hex): Servicios horarios

La interrupción 26 suministra los servicios de hora del día. A diferencia de las otras interrupciones que cubren esta sección, pero al igual que todos los demás servicios de la ROM-BIOS, se pueden activar más de un servicio con esta interrupción. Los dos servicios normales se numeran con el 0 y el 1, y se especifican, como es usual, en el registro AH. (● Véase la figura 12-12.)

La ROM-BIOS mantiene un reloj horario que se basa en la cuenta de los *ticks* generados por el reloj del sistema desde la medianoche. El reloj del sistema genera impulsos mediante la interrupción 8 a intervalos específicos. En cada impulso de reloj o *ticks*, la rutina de servicio de interrupción 8 incrementa la cuenta del reloj en 1. Cuando esta cuenta sobrepasa el equivalente en impulsos a las 24 horas (12 de la noche), se vuelve a poner a 0 (*reset*) y se hace una marca para indicar que ha pasado la medianoche. Esta marca no tiene forma numérica y, por tanto, no hay manera de detectar si ha pasado dos o más veces por la medianoche.

El reloj da impulsos a una velocidad que es casi con exactitud 1193180/64K, o aproximadamente 18,2 veces por segundo. La cuenta se guarda como un entero de 4 bytes en la posición de memoria 46C hex. El valor de medianoche usado para comparar con la cuenta ascendente del reloj es 1573040 (1800B0 hex) y se almacena en la posición 470 hex (● véase página 64). Cuando el DOS necesita saber la hora, lee el conteo del reloj mediante el servicio de hora del

Servicio	Descripción	Ajuste de registros
0	Lee el conteo del reloj	CX = parte de mayor peso de la cuenta del reloj DX = parte de menor peso de la cuenta del reloj AL = 0 si el reloj no ha sobrepasado el periodo de 24 horas AL <> 0 si el reloj cuenta un nuevo día
1	Ajusta el conteo del reloj	CX = parte de mayor peso de la cuenta del reloj DX = parte de menor peso de la cuenta del reloj

Figura 12-12. Los dos servicios de hora del día de la ROM-BIOS, llamados mediante la interrupción 26, y su ajuste de registros

día y calcula la hora a partir de sencillas operaciones. Si ve que el reloj ha pasado por cero (*reset*), incrementa la fecha.

Podemos calcular la hora actual a partir del conteo del reloj usando las siguientes fórmulas BASIC:

```
HORA = RELOJ \ 65543 (hex 10007)
RESTO = RELOJ MOD 65543
MINUTOS = RESTO \ 1092 (hex 444)
RESTO = RESTO MOD 1092
SEGUNDOS = RESTO \ MOD 18.21 ; para precisión; de otro modo use 18
RESTO = RESTO MOD 18.21
CENTESIMAS = CINT( RESTO * 100 )
```

A la inversa, podemos calcular la cuenta del reloj de forma casi correcta a partir de la hora, utilizando esta fórmula:

```
RELOJ = (HORA * 65543.33) + (MINUTOS * 1092.38)
+ (SEGUNDOS * 18.21) + (CENTESIMAS * .182)
```

AT Como veremos brevemente, las mejoras del BIOS que vienen con el AT incluyen servicios de hora que realizan algunas de estas tareas.

Servicio 0: Lee la cuenta actual del reloj

El servicio 0 vuelca la cuenta actual del reloj en dos registros: la parte de mayor peso en CX y la de menor peso en DX, AL es cero si no ha pasado la medianoche desde que fue leído o ajustado el último

valor del reloj. La señal de medianoche siempre se reajusta cuando el reloj es leído. Es responsabilidad de cualquier programa la utilización de la señal de medianoche para tener en cuenta los cambios de fecha. Los programas DOS normalmente no usan este servicio directamente. Si lo hacen, deben encargarse de la tediosa tarea de calcular y ajustar una nueva fecha.

☐ *NOTA: Encuentro curioso que la versión 2.00 del DOS no actualice convenientemente la fecha con la señal de medianoche. La siguiente versión del DOS, 2.10, y todas las demás versiones del DOS, lo hacen.*

Servicio 1: Da valor a la cuenta del reloj

El servicio 1 da valor a la cuenta del reloj situada en la posición 46 C hex. El valor se toma del par de registros CS:DX. La señal de medianoche se reajusta siempre que la cuenta alcanza la marca de 24 horas.

Los servicios horarios del AT

Los servicios que van del 2 al 6 también se llaman mediante la interrupción 26 y se introdujeron en la versión del BIOS del AT. Los servicios 2, 3 y 4 leen y ajustan la hora del reloj de tiempo real, proporcionando tanto la hora del día como la información de la fecha, y los servicios 5 y 6 ajustan una alarma para que suene pasado un tiempo de hasta 24 horas. ➤ Para obtener más información sobre estos servicios, véase página 267, o la lista del BIOS en el «Manual Técnico de Consulta del AT».

AT

Resumen de servicios de la ROM-BIOS

Resumen breve 252

Resumen amplio 254

Este capítulo presenta un resumen de las rutinas de servicio de la ROM-BIOS que se han tratado desde el capítulo 8 hasta el 12, y tiene por objeto proporcionarle una guía de referencia rápida. Una vez que comprenda los servicios de la ROM-BIOS, estas tablas le proporcionarán toda la información que necesite para la programación.

RESUMEN BREVE

En esta sección listamos brevemente todos los servicios de la ROM-BIOS, de tal forma que se puedan consultar de forma rápida.

Tema	Interrupción		Servicio (hex)	Descripción	Modo específico
	Dec	Hex			
Impresión de pantalla	5	5	n/a	Envía los contenidos de la pantalla a la impresora	
Video	16	10	0	Fija el modo de video	
Video	16	10	1	Ajusta el tamaño del cursor	
Video	16	10	2	Posiciona el cursor	
Video	16	10	3	Lee la posición del cursor	
Video	16	10	4	Lee la posición del lápiz óptico	
Video	16	10	5	Elección de la página activa	
Video	16	10	5 (AL:128)	Obtención de los registros de la página de visualización	
Video	16	10	5 (AL:129)	Da valores al registro de página de visualización de la CPU	
Video	16	10	5 (AL:130)	Da valores al registro de página de visualización del CRT	
Video	16	10	5 (AL:131)	Da valores a ambos registros de página de visualización	
Video	16	10	6	Desplazamiento vertical ascendente (<i>Scroll up</i>)	
Video	16	10	7	Desplazamiento vertical descendente (<i>Scroll down</i>)	
Video	16	10	8	Lee carácter y atributo	
Video	16	10	9	Escribe carácter y atributo	
Video	16	10	A	Escribe carácter	
Video	16	10	B	Elección de la paleta de color	
Video	16	10	C	Escribe un <i>pixel</i>	
Video	16	10	D	Lee un <i>pixel</i>	
Video	16	10	E	Escribir un carácter como TTY	
Video	16	10	F	Obtiene el modo actual del video	
Video	16	10	10 (AL:0)	Da valores a un registro de paleta	
Video	16	10	10 (AL:1)	Da valores al registro del margen	
Video	16	10	10 (AL:2)	Da valores a todos los registros de paleta	
Video	16	10	13	Escribir una cadena de caracteres	
Equipo	17	11	n/a	Obtiene la lista de los equipos periféricos	
Memoria	18	12	n/a	Obtiene el tamaño de la memoria útil (en Kbytes)	

JR
JR
JR
AT

Figura 13-1. Un breve resumen de los servicios de la ROM-BIOS

Tema	Interrupción		Servicio (hex)	Descripción	Modo específico
	Dec	Hex			
Disco	19	13	0	Reinicializa el sistema del disco	
Disco	19	13	1	Obtención del estado del disco	
Disco	19	13	2	Lee sectores del disco	
Disco	19	13	3	Escribe sectores en el disco	
Disco	19	13	4	Verifica los sectores del disco	
Disco	19	13	5	Formatea las pistas del disco	
Disco	19	13	8	Obtiene los actuales parámetros del controlador	AT
Disco	19	13	9	Inicializa las tablas de parámetros del disco duro	AT
Disco	19	13	A	Lectura de sectores largos	AT
Disco	19	13	B	Escritura de sectores largos	AT
Disco	19	13	C	Busca un cilindro	AT
Disco	19	13	D	Reinicialización alternativa del disco	AT
Disco	19	13	10	Comprueba si la unidad está lista	AT
Disco	19	13	11	Recalibrado de la unidad	AT
Disco	19	13	14	Diagnóstico del controlador	AT
Disco	19	13	15	Obtención del tipo de disco	AT
Disco	19	13	16	Cambio el estado del disco	AT
Disco	19	13	17	Indica el tipo de disco	AT
Puerto: serie	20	14	0	Inicializa los parámetros del puerto serie	
Puerto: serie	20	14	1	Envía un carácter	
Puerto: serie	20	14	2	Recibe un carácter	
Puerto: serie	20	14	3	Obtiene el estado del puerto serie	
Cassette	21	15	0	Activa el motor del cassette	
Cassette	21	15	1	Desactiva el motor del cassette	
Cassette	21	15	2	Lee bloques de datos	
Cassette	21	15	3	Escribe bloques de datos	
Dispositivos	21	15	80	Dispositivo abierto	AT
Dispositivos	21	15	81	Dispositivo cerrado	AT
Dispositivos	21	15	82	Finalización del programa del dispositivo	AT
Dispositivos	21	15	83	Espera de suceso	AT
Palanca de mando					
(Joystick)	21	15	84	Joystick	AT
Petición del sistema	21	15	85	Pulsación de la tecla SysReq	AT
Dispositivos	21	15	86	Espera	AT
Dispositivos	21	15	87	Mueve bloque	AT
Memoria	21	15	88	Obtención del tamaño de la memoria ampliada	AT
Memoria	21	15	89	Conmuta a memoria virtual	AT
Dispositivos	21	15	90	Bucle de dispositivo ocupado	AT
Dispositivos	21	15	91	Activación del indicador y realización de interrupción	AT
Teclado	22	16	0	Lee el siguiente carácter del teclado	
Teclado	22	16	1	Informa si hay algún carácter preparado en el <i>buffer</i>	
Teclado	22	16	2	Obtiene el estado de shift	

Figura 13-1. Un breve resumen de los servicios de la ROM-BIOS

Tema	Interrupción		Servicio (hex)	Descripción	Modo específico
	Dec	Hex			
Teclado	22	16	3 (AL:0)	Reinicializa el sistema de repetición de tecla	JR
Teclado	22	16	3 (AL:1)	Incrementa el retardo inicial	JR
Teclado	22	16	3 (AL: 2)	Incrementa el retardo de continuación	JR
Teclado	22	16	2 (AL:3)	Incrementa ambos retardos	JR
Teclado	22	16	3 (AL:4)	Desactiva el sistema de repetición de tecla	JR
Teclado	22	16	4 (AL:0)	Desconecta el <i>click</i>	JR
Teclado	22	16	4 (AL:1)	Conecta el <i>click</i>	JR
Impresora	23	17	0	Envía un byte a la impresora	
Impresora	23	17	1	Inicializa la impresora	
Impresora	23	17	2	Obtiene el estado de la impresora	
BASIC	24	18	n/a	Pasa el control al BASIC	
Bootstrap	25	19	n/a	Relanza el ordenador	
Hora	26	1A	0	Lee la cuenta del reloj	
Hora	26	1A	1	Da valor a la cuenta del reloj	
Hora	26	1A	2	Lee la hora del reloj de tiempo real	AT
Hora	26	1A	3	Fija la hora del reloj de tiempo real	AT
Hora	26	1A	4	Lee la fecha del reloj de tiempo real	AT
Hora	26	1A	5	Fija la fecha del reloj de tiempo real	AT
Hora	26	1A	6	Pone la alarma	AT
Hora	26	1A	7	Quita la alarma	AT

Figura 13-1. Un breve resumen de los servicios de la ROM-BIOS (continuación)

RESUMEN AMPLIO

En esta sección ampliaremos la tabla-resumen anterior para mostrar la utilización de los registros para pasar parámetros. La sección anterior se usa más para encontrar rápidamente qué servicio se necesita; esta sección se usa más para encontrar rápidamente cómo usar cada servicio.

Servicio	Interrupción (hex)	Entrada	Registro	Salida	Descripción
Copia impresa de la pantalla	05	AH = 05	n/a		Envía a la impresora el contenido de la pantalla. El byte de estado y resultado se encuentra en la dirección baja de memoria hex 500 (0050: 0000)

Servicio	Interrupción (hex)	Entrada	Registro	Salida	Descripción
<i>Servicios de video</i>					
Fija el modo de video		AH = 06 AL = modo de video	ninguna		Modos de video en AL: 00: Texto 40 × 25, 16 grises 01: Texto 40 × 25, 16/8 colores 02: Texto 80 × 25, 16 grises 03: Texto 80 × 25, 16/8 colores 04: Gráficos 320 × 200, 4 colores 05: Gráficos 320 × 200, 4 grises 06: Gráficos 640 × 200, blanco y negro (b/w) 07: Texto 80 × 25, blanco y negro (b/w) 08: Gráficos 160 × 200, 16 colores 09: Gráficos 320 × 200, 16 colores 0A: Gráficos 640 × 200, 4 colores
Ajusta el tamaño del cursor	10	AH = 01 CH = línea de comienzo CL = línea de final	ninguna		El adaptador color/gráficos usa las líneas 0-7 El adaptador monocromo usa las líneas 0-13
Fija la posición del cursor	10	AH = 02 NH = número de página DH = fila DL = columna	ninguna		
Lee la posición del cursor	10	AH = 03 BH = número de página del display	CH = línea de comienzo CL = línea de final DH = fila DL = columna		
Lee la posición del lápiz óptico	10	AH = 04	AH = señal de disparo del lápiz		
			BX = columna del <i>pixel</i> CH = fila del <i>pixel</i> DH = fila del carácter DL = columna del carácter		
Fija la página activa	10	AH = 05 AL = número de página			

Figura 13-2. Un resumen completo de los servicios de la ROM-BIOS

Servicio	Interrupción (hex)	Entrada	Registro	Salida	Descripción
Obtiene el registro de página	10	AH = 05 AL = 80		BH = registro de página del CTR BL = registro de página de la CPU	
Inicializa el registro de la página de la CPU	10	AH = 05 AL = 81 BL = registro de página de la CPU		BH = registro de página del CTR BL = registro de página de la CPU	
Inicializa el registro de página del CRT	10	AH = 05 AL = 82 BH = registro de página del CRT		BH = registro de página del CRT BL = registro de página de la CPU	
Inicializa ambos registros de página	10	AH = 05 AL = 83 BH = registro de página del CRT BL = registro de página de la CPU		BH = registro de página del CRT BL = registro de página de la CPU	
Desplazamiento vertical ascendente (<i>scroll up</i>)	10	AH = 06 AL = líneas a subir BH = atributo de las líneas añadidas CH = fila superior CL = columna de la izquierda DH = fila inferior DL = columna de la derecha		ninguna	
Desplazamiento vertical descendente (<i>scroll down</i>)	10	AH = 07 AL = líneas a bajar BH = atributo de las líneas añadidas CH = fila superior CL = columna de la izquierda DH = fila inferior DL = columna de la derecha			
Lee carácter y atributo	10	AH = 08 BH = número de página del display		AH = carácter AL = atributo	

Figura 13-2. Un resumen completo de los servicios de la ROM-BIOS (continuación)

Servicio	Interrupción (hex)	Entrada	Registro	Salida	Descripción
Escribe carácter y atributo	10	AH = 09 AL = carácter BH = número de página BL = atributo CX = número de caracteres a repetir		ninguna	
Escribe carácter	10	AH = 0A AL = carácter BH = número de página BL = color en modo gráfico CX = número total de caracteres		ninguna	
Elección de la paleta de color	10	AH = 0B BH = identificador de paleta BL = color que se va a usar		ninguna	
Escribe un <i>pixel</i>	10	AH = 0C AL = color CX = columna del <i>pixel</i> DL = fila del <i>pixel</i>		ninguna	
Lee un <i>pixel</i>	10	AH = 0D CX = columna del <i>pixel</i> DL = fila del <i>pixel</i>		AL = color leído	
Escribe un carácter como TTY	10	AH = 0E AL = carácter BL = color para el modo gráfico		ninguna	
Obtiene el modo actual del video	10	AH = 0F		AH = anchura en caracteres AL = modo de video BH = número de página	
JR Da valores al registro de la paleta	10	AH = 10 AL = 00 BH = valor de la paleta BL = registro de la paleta		ninguna	
JR Da valores al registro del margen	10	AH = 10 AL = 01 BL = color de margen		ninguna	

Figura 13-2. Un resumen completo de los servicios de la ROM-BIOS (continuación)

Servicio	Interrupción (hex)	Entrada	Registro	Salida	Descripción
JR Da valores a todos los registros de paleta	10	AH = 10 AL = 02 ES:DX = puntero de los valores de paleta		ninguna	
AT Escribe cadena sin mover el cursor	10	AH = 13 AL = 00 BL = atributo BH = número de página DX = posición inicial del cursor CX = longitud de la cadena ES:BP = puntero del principio de la cadena		ninguna	
AT Escribe cadena moviendo el cursor al final de la cadena		AH = 13 AL = 01 BL = atributo BH = número de página DX = Posición inicial del cursor CX = longitud de la cadena ES:BP = puntero del principio de la cadena		ninguna	
AT Escribe cadena de caracteres y atributos alternos: no mueve el cursor	10	AH = 13 AL = 02 BH = número de página DX = posición inicial del cursor CX = longitud de la cadena ES:BP = puntero del principio de la cadena		ninguna	
AT Escribe cadena de caracteres y atributos alternos: mueve el cursor	10	AH = 13 AL = 03 BH = número de página DX = posición inicial del cursor CX = longitud de la cadena ES:BP = puntero del principio de la cadena		ninguna	

*Figura 13-2. Un resumen completo de los
servicios de la ROM-BIOS (continuación)*

Servicio	Interrupción (hex)	Entrada	Registro	Salida	Descripción
<i>Servicio de lista de equipo</i>					
Obtiene la lista del equipo periférico	11	ninguno		AX = código de bits de la lista de equipo	Valores de AX: 00 = unidad de disco 01 = coprocesador matemático 02, 03 = RAM en bloques de 16K 04, 05 = modo de video inicial 00 = sin usar 01 = color 40 × 25; 10 = color 80 × 25; 11 = blanco y negro 80 × 25 06, 07 = número de unidades de disco 08 = DMA 00 = Sí; 01 = no 09, 10, 11 = número de tarjetas RS-232 en el sistema 12 = E/S para juegos conectada (no usado en AT) 13 = impresora serie conectada 14, 15 = número de impresoras conectadas
<i>Servicio de memoria</i>					
Obtiene el tamaño de la memoria útil (en Kbytes)	12	ninguna		AX = tamaño de la memoria	
<i>Servicios de disco</i>					
Reinicializa el sistema del disco	13	AH = 00		ninguna	

Figura 13-2. Un resumen completo de los servicios de la ROM-BIOS (continuación)

Servicio	Interrupción (hex)	Entrada	Registro	Salida	Descripción
Obtiene el estado del disco	13	AH = 01	AL = código de estado		Valores de estado: AL = 1: comando incorrecto AL = 2: marca de dirección no encontrada AL = 3: intento de escritura en disco protegido AL = 4: sector no encontrado AL = 6: disco retirado AL = 8: rebasar el límite de DMA AL = 9: DMA más allá del límite de 64K AL = 10: mal CRC AL = 20: fallo del controlador NEC AL = 40: fallo del búsqueda AL = 80: time-out
Lee sectores del disco	13	AH = 02 AL = número de sectores CH = número de pista CL = número de sector DH = número de cabeza DL = número de unidad ES:BX = puntero de <i>buffer</i>	CF = Señal de éxito/fracaso & = código de estado AL = número de sectores leídos		Códigos de estado en AH. Vea el servicio del disco 01
Escribe sectores del disco	13	AH = 03 AL = número de sectores CH = número de pista CL = número de sector DH = número de cabeza DL = número de unidad ES:BX = puntero de <i>buffer</i>	CF = Señal de éxito/fracaso AH = código de estado AL = número de sectores escritos		Códigos de estado en AH: Vea el servicio del disco 01
Verifica sectores del disco	13	AH = 04 AL = número de sectores CH = número de pista CL = número de sector DH = número de cabeza DL = número de unidad	CF = Señal de éxito/fracaso AH = código de estado AL = número de sectores escritos		Códigos de estado en AH. Vea el servicio del disco 01

Figura 13-2. Un resumen completo de los servicios de la ROM-BIOS (continuación)

Servicio	Interrupción (hex)	Entrada	Registro	Salida	Descripción
Formateo de pistas	13	AH = 05 AL = número de sectores CH = número de pista CL = número de sector DH = número de cabeza DL = número de unidad ES:BX = puntero de la lista de 4 bytes Byte 1 = pista Byte 2 = cabeza Byte 3 = sector Byte 4 = bytes/sector	CF = señal de éxito/fracaso AH = código de estado		Códigos de estado en AH: vea el servicio del disco 01
AT Obtiene los actuales parámetros de la unidad	13	AH = 08	DL = número de unidades DH = máximo número de caras CH = máximo número de rectores CH = máximo número de pistas CF = indicador de éxito/fracaso AH = código de estado		Códigos de estado en AH: Vea el servicio del disco 01
AT Inicializa las tablas base del disco duro	13	AH = 09	CF = indicador de éxito/fracaso AH = código de estado		La interrupción 41 apunta a la tabla de la unidad 0 La interrupción 46 apunta a la tabla de la unidad 1 Códigos de estado en AH: vea el servicio del disco 01
AT Lectura de sectores largos	13	AH = 0A DL = identificador de la unidad DH = número de cabeza CH = número de cilindro CL = número de sector ES:BX = puntero del <i>buffer</i>	CF = indicador del éxito/fracaso AH = código de estado		Códigos de estado en AH: vea el servicio de disco 01
AT Escritura de sectores largos	13	AH = 0B DL = identificador de la unidad DH = número de cabeza CH = número de cilindro CL = número de sector ES:BX: puntero del <i>buffer</i>	CF = indicador de éxito/fracaso AH = código de estado		Códigos de estado en AH: vea el servicio de disco 01

Figura 13-2. Un resumen completo de los servicios de la ROM-BIOS (continuación)

Servicio	Interrupción (hex)	Entrada	Registro	Salida	Descripción
AT Busca un cilindro	13	AH = 0C DL = identificador de unidad CH = número de sector	CF = indicador de éxito/fracaso AH = código de estado		Códigos de estado en AH: vea el servicio de disco 01
AT Reinicialización alternativa	13	AH = 0D DL = identificador de unidad	CF = indicador de éxito/fracaso AH = código de estado		Códigos de estado en AH: vea el servicio de disco 01
AT Comprueba si la unidad está lista		AH = 10 DL = identificador de unidad	CF = indicador de éxito/fracaso AH = código de estado		Códigos de estado en AH: vea el servicio de disco 01
AT Recalibrado de la unidad	13	AH = 11 DL = identificador de unidad	CF = indicador de éxito/fracaso AH = código de estado		Códigos de estado en AH: vea el servicio de disco 01
AT Diagnóstico del controlador	13	AH = 14	CF = indicador de éxito/fracaso AH = código de estado		Códigos de estado en AH: vea el servicio de disco 01
AT Obtención del tipo de disco	13	AH = 15 DL = identificador de unidad	AH = tipo de disco CX, DX = número de sectores de 512 bytes cuando AH = 3		Tipos de disco: AH = 0: el disco no está AH = 1: el <i>diskette</i> no ha cambiado AH = 2: el <i>diskette</i> ha cambiado AH = 3: disco duro
AT Cambio del estado del disco	13	AH = 16	DL = unidad en la que se ha cambiado el disco AH = estado de cambio del disco 00 = disco no cambiado 01 = disco cambiado		
AT Fija el tipo de disco	13	AH = 17 AL = tipo de disco			Tipo de disco en AL: AL = 00: no hay disco AL = 01: disco normal en unidad normal AL = 03: disco de gran capacidad (1.2 megabytes) en controlador de gran capacidad

Figura 13-2. Un resumen completo de los servicios de la ROM-BIOS (continuación)

Servicio	Interrupción (hex)	Entrada	Registro	Salida	Descripción
<i>Servicios del puerto serie</i>					
Inicializar los parámetros del puerto serie	14	AH = 00 DX = número del puerto serie	AX = estado del puerto serie		Bits de estado: 00,01 = longitud de la palabra 10 = 7 bits; 11 = 8 bits 02 = bits de parada; 0 = 1; 1 = 2 03,04 = paridad 00,01 = ninguna; 01 = impar; 11 = par 05, 06, 07 = velocidad en baudios: 000 = 110 001 = 150 010 = 360 011 = 600 100 = 1200 101 = 2400 110 = 4800 111 = 9600 (4800 en el PCjr)
Envía un carácter	14	AH = 01 AL = carácter DX = número del puerto serie	AH = código de éxito/fracaso AL = estado del <i>modem</i>		Bits en AH: 00 = dato preparado 01 = error de alcance 02 = error de paridad 03 = error de tramas 04 = interrupción de portadora 05 = <i>buffer</i> de transmisión vacío 06 = registro de desplazamiento vacío 07 = error de tempori- zación Bits en AL: 00 = preparado para el envío (CTS). Secundario 01 = datos listos (DSR). Secundario 02 = detección secundaria de llamada 03 = detección de porta- dora secundaria 04 = preparado para el envío (CTS) 05 = datos listos (DSR) 06 = detección de llamada 07 = detección de portadora

Figura 13-2. Un resumen completo de los
servicios de la ROM-BIOS (continuación)

Servicio	Interrupción (hex)	Entrada	Registro	Salida	Descripción
Recibe un carácter	14	AH = 02 DX = número del puerto serie		AH = código de estado de éxito/fracaso AL = carácter	Bits de estado: vea el servicio de la puerta serie 01
Obtención del estado del puerto serie	14	AH = 03		AX = código de estado	Código de los bits de código: vea el servicio de la puerta serie 00
<i>Servicios de cinta de cassette</i>					
Activa el motor del cassette	15	AH = 00		ninguna	
Desactiva el motor del cassette	15	AH = 01		ninguna	
Lee bloques de datos	15	AH = 02 CX = cuenta de los bytes ES:BX = puntero del área de datos		CF = señal de error DX = cuenta la lectura de los bytes ES:BX = puntero del último byte leído	
Escribe bloques de datos	15	AH = 03 CX = cuenta de los bytes a escribir ES:BX puntero del área de datos		ES:BX = puntero del último byte escrito	
<i>Servicios extendidos para el AT</i>					
AT Dispositivo abierto	15	AH = 80 BX = identificador de periférico CX = tipo de proceso		ninguna	
AT Dispositivo cerrado	15	AH = 81 BX = identificador de periférico CX = tipo de proceso		ninguna	
AT Finalización del programa del dispositivo	15	AH = 82 BX = identificador de periférico		ninguna	
AT Espera de suceso	15	AH = 83 AL = subservicio 0 = ajuste del intervalo 1 = cancelar ES:BX = puntero a la memoria del dispositivo CX, DX = número de microsegundos de espera		ninguna	

Figura 13-2. Un resumen completo de los servicios de la ROM-BIOS (continuación)

Servicio	Interrupción (hex)	Entrada	Registro	Salida	Descripción
AT Joystick	15	AH = 84 DX = 0 obtiene los valores actuales de los conmutadores	AL = valores de los conmutadores		
AT Joystick	15	AH = 84 DX = 1 lee las entradas	AX = valor de A(x) BX = valor de A(y) CX = valor de B(x) DX = valor de B(y)		
AT Pulsación de la tecla SysReq	15	AH = 85 AL = 00 pulsación, AL = 01 ruptura suelta	ninguna		
AT Espera	15	AH = 86 CX, DX = número de microsegundos de espera	ninguna		
AT Mueve bloque	15	AH = 87 CX = número de palabras a mover ES:SI = puntero de la tabla	ninguna		
AT Obtención del tamaño de la ampliación de memoria	15	AH = 88	ninguna		
AT Conmuta el modo virtual	15	AH = 89	ninguna		Precaución: comprueba un listado de BIOS antes de usarlo
AT Bucle del dispositivo ocupado	15	AH = 90 AL = código de tipo	ninguna		Comprueba un listado del BIOS
AT Activación del indicador y realización de interrupción	15	AH = 91 AL = código de tipo	ninguna		Comprueba un listado del BIOS
<i>Servicios de teclado</i>					
Lee el siguiente carácter del teclado	16	AH = 00	AL = código de exploración (byte auxiliar) AL = código de carácter (byte principal)		

Figura 13-2. Un resumen completo de los servicios de la ROM-BIOS (continuación)

Servicio	Interrupción (hex)	Entrada	Registro	Salida	Descripción
Informa hay carácter preparado en el <i>buffer</i>	16	AH = 01		ZF = señal de "preparado" AH = código de rastreo (byte auxiliar) AL = código de carácter (byte principal)	
Obtener el estado de la tecla (shift)	16	AH = 02		AL = bits de estado de shift	Bits de estado: Bit 0 = 1: Shift derecho presionado Bit 1 = 1: Shift izquierdo presionado Bit 2 = 1: Ctrl presionado Bit 3 = 1: Atl presionado Bit 4 = 1: Scroll Lock activo Bit 5 = 1: Num Lock activo Bit 6 = 1: Caps Lock (Mayúscula) activo Bit 7 = 1: Estado de inserción activo
JR Reinicializa el sistema de repetición de tecla	16	AH = 03 AL = 00		ninguna	
JR Incrementa el retardo inicial	16	AH = 03 AL = 01		ninguna	
JR Incrementa el retardo de continuación	16	AH = 03 AL = 02		ninguna	
JR Incrementa ambos retardos	16	AH = 03 AL = 03		ninguna	
JR Desactiva el sistema de repetición de tecla	16	AH = 03 AL = 04		ninguna	
JR Desconecta el <i>click</i>	16	AH = 04 AL = 00		ninguna	

Figura 13-2. Un resumen completo de los servicios de la ROM-BIOS (continuación)

Servicio	Interrupción (hex)	Entrada	Registro	Salida	Descripción
JR Conecta el click	16	AH = 04 AL = 01		ninguna	
<i>Servicios de impresora</i>					
Envía un byte a la impresora	17	AH = 00 AL = carácter		AH = código éxito/fracaso	Bits de estado: 0 = time-out 1 = no usado 2 = no usado 3 = 1: error de entrada/salida (I/O) 4 = 1: seleccionado 5 = 1: sin papel 6 = 1: reconocimiento 7 = 1: no ocupado
Inicializa la impresora	17	AH = 01		AH = código de estado	Bits del código de estado. Vea el servicio de impresora 00
Obtiene el estado de la impresora	17	AH = 02		AH = código de estado	Bits de código de estado. Vea el servicio de impresora 00
<i>Servicios varios</i>					
Pasa el control al BASIC	18	ninguna		n/a	Sin retorno; así que no hay salida posible
Relanza el ordenador	19	ninguno		n/a	Sin retorno; así que no hay salida posible
<i>Servicios horarios</i>					
Lee la cuenta del reloj	1A	AH = 00		AL = señal de medianoche CX = parte alta de la cuenta DX = parte baja de la cuenta	
Da valores a la cuenta del reloj	1A	AH = 01 CX = parte alta de la cuenta DX = parte baja de la cuenta		ninguno	
AT Lee la hora del reloj de tiempo real	1A	AH = 02		CH = horas CL = minutos DH = segundos	

Figura 13-2. Un resumen completo de los servicios de la ROM-BIOS (continuación)

Servicio	Interrupción (hex)	Entrada	Registro	Salida	Descripción
Fija la hora del reloj de tiempo real	1A	AH = 03 CH = horas CL = minutos DH = segundos DL = 1 si es la hora normal: 0 si es la hora estándar			
Lee la fecha del reloj de tiempo real	1A	AH = 04		DL = día DH = mes CL = Dos últimas cifras del año CH = Dos primeras cifras del año	
Fija la fecha del reloj de tiempo real	1A	AH = 05 DL = día DH = mes CL = Dos últimas cifras del año CH = Dos primeras cifras del año			
Pone la alarma	1A	AH = 06 CH = horas CL = minutos DH = segundos			Sitúe la dirección de la rutina de alarma en la posición de la interrupción 4A
Quita la alarma	1A	AH = 07			

Figura 13-2. Un resumen completo de los servicios de la ROM-BIOS (continuación)

El DOS, conceptos básicos

Los pros y los contras de usar los servicios del DOS 270

El DOS: la mejor herramienta para manejar discos 271

El DOS y el video, una pareja difícil 272

Diferencias entre las versiones del DOS 273

Consideraciones sobre el formato del disco 274

Comentarios 275

Desde el capítulo 15 hasta el 18 se va a tratar sobre los servicios de apoyo a los programas que proporciona el DOS. El último capítulo de esta serie, el 18, es un resumen de los detalles técnicos de cada servicio. En este capítulo presentaré algunos de los principales asuntos con los que se encuentra un programador cuando trabaja con los servicios del DOS.

Utilizo el término *servicios* del DOS para definir el juego completo de operaciones que el DOS proporciona a los programas, aunque no encontrará este término en los manuales del DOS. Utilizando la propia terminología del DOS, estos servicios están divididos en dos categorías: interrupciones DOS y llamadas a las funciones del DOS. Todo lo que yo sé es que esta distinción no está basada en ninguna decisión de diseño, sino que surge del deseo de alcanzar un razonable grado de compatibilidad con el predecesor del DOS, el sistema operativo CP/M.

Las interrupciones del DOS se llaman mediante códigos de interrupción individuales con la instrucción INT. Y las llamadas a las funciones del DOS se realizan del mismo modo que los servicios de la ROM-BIOS; es decir, mediante una interrupción común, la interrupción 33 (21 hex). Como en el caso de los servicios de la ROM-BIOS, las funciones individuales se seleccionan a través del registro AH.

Desde el punto de vista tanto de la programación como del diseño, el mecanismo de llamadas a una función, utilizando una sola interrupción, realmente es más eficiente que utilizar un grupo de interrupciones individuales. Esto es fundamentalmente beneficioso, ya que ha permitido añadir un número ilimitado de nuevos servicios. Todos los servicios introducidos con las versiones del DOS-2 son adiciones que se han hecho a las funciones y no a las interrupciones. La mayoría de los servicios que fueron introducidos con las versiones del DOS-3 son también llamadas a función, aunque hay una nueva interrupción.

LOS PROS Y LOS CONTRAS DE USAR LOS SERVICIOS DEL DOS

La cuestión de si usar o no usar los servicios del DOS surge naturalmente durante el diseño y desarrollo de programas sofisticados. Mi consejo general, repetido a lo largo de este libro, es emplear los servicios de mayor nivel disponibles que lleven a cabo lo que se necesite. Esto quiere decir que, siempre que sea posible, debe usar las prestaciones que tenga incorporadas el lenguaje de programación de alto nivel, que se estén utilizando, recurriendo sólo cuando sea necesario al uso directo de los servicios del DOS, o de los servicios de

la ROM-BIOS, y sólo en circunstancias extremas se debe recurrir a la programación directa del *hardware* del ordenador.

Desde un punto de vista práctico, un programa puede estar escrito totalmente utilizando las prestaciones del lenguaje de programación, o bien casi todo el trabajo de entrada/salida debe hacerse fuera del lenguaje de programación, trabajando en un nivel más bajo. Cuando se necesita un nivel inferior de programación, creo que, con muy pocas excepciones, los servicios del DOS se adaptan mejor a las operaciones con disco, y cuando se trabaja con el teclado u otros dispositivos de entrada/salida, son adecuadas tanto las rutinas del DOS como las rutinas de la ROM-BIOS, dependiendo la elección de la aplicación. En el caso de necesitarse una programación gráfica de bajo nivel, la situación es más compleja. La obtención de una salida por pantalla satisfactoria casi siempre requiere los servicios de la ROM-BIOS y una programación directa del *hardware*, aunque en algunos casos puede ser mejor dejarlo en manos del DOS. Veremos por qué dentro de un momento.

El DOS: La mejor herramienta para manejar discos

Cuando se examina el alcance total de las herramientas y servicios que están en nuestras manos a través de los lenguajes de programación, el DOS, la ROM-BIOS y el *hardware* del ordenador, queda bastante claro que la concentración más rica de servicios orientados al disco se encuentra a nivel del DOS. No hace falta decir que el DOS, como sistema operativo de disco, es intrínsecamente el más potente apoyo a operaciones de disco.

Como se detallará en los capítulos 16 y 17, la mayoría de los servicios que el DOS realiza está directamente relacionada con la manipulación de ficheros en disco. Incluso algunos servicios nominalmente controlados por programa, como cargar y ejecutar otro programa (función 75 (4B hex)), incluyen operaciones con ficheros en disco. Desde esta perspectiva, el DOS no es un sistema operativo de disco, sino un sistema de servicios de disco, diseñado para que lo puedan utilizar los programas. Cuando se desarrollan programas para la familia de ordenadores personales IBM, es una buena idea considerar al DOS exactamente desde el siguiente punto de vista: el DOS es la mejor herramienta puesta a nuestro servicio para realizar las operaciones con disco.

El DOS y el video, una pareja difícil

El realizar la salida por pantalla a bajo nivel ha llegado a ser algo común en la mayoría de los programas sofisticados. Normalmente, toda salida por pantalla se realiza al más bajo nivel, actuando directamente en el área de memoria de la pantalla. Otras operaciones, como el movimiento del cursor, se hacen normalmente pasando a un nivel superior; es decir, a través de los servicios de la ROM-BIOS.

Al principio, resultaba necesario porque el DOS no proporcionaba los servicios de video adecuados. Pero a partir de la versión 2.00 es posible realizar la mayor parte del trabajo pantalla, utilizando los servicios del DOS ampliados por el programa controlador ANSI, también conocido como ANSI.SYS (● véase apéndice A para encontrar más detalles). Este programa emplea un juego de comandos que, cuando se interpretan, realizan cualquier acción que se pueda realizar en pantalla. De cualquier modo, los servicios del controlador ANSI pueden ser algo engorrosos de utilizar, porque no sólo requieren que los programas funcionen bajo la versión del DOS-2, sino que también el DOS debe estar configurado para incluir el controlador. Mi experiencia es que muchos usuarios no tienen muy claros los procesos necesarios para incorporar el controlador ANSI, y este único factor es un gran argumento en contra de usar cualquier facilidad del DOS que lo requiera.

Si nos enfrentamos sólo con este factor, se podría llegar fácilmente a la conclusión de que se debe evitar usar los servicios de video DOS en conjunto, pero no es todo tan sencillo. Muchos de los entornos operativos más sofisticados que están apareciendo, particularmente los sistemas de ventanas (*windowing systems*) requieren que los programas que actúen bajo su control usen los servicios de sistema operativo disponibles oficialmente, y que no trabajen directamente con el *hardware*. Estos sistemas son un gran argumento a favor del uso estricto de los servicios del DOS, siempre que sea posible.

Al intentar decidir lo que es más prudente hacer, se ve que depende mucho de la vida media probable de los programas, y del rango de las máquinas en las que van a ser usados. Para un programa de juego específico del PC, con una vida media de unos pocos meses (normal para los juegos), no hay razones para preocuparse de estas cosas. La situación es completamente diferente para una aplicación generalizada de negocios o profesional, que será útil durante muchos años y en muchos ambientes. Haga su elección lo mejor posible.

DIFERENCIAS ENTRE LAS VERSIONES DEL DOS

El DOS 3.10 representa la sexta versión oficial del DOS. Aunque ha habido tanto mejoras del *software* como perpetuación de errores en cada versión, la razón para cada cambio ha sido el *hardware*, y cada cambio en el *hardware* normalmente ha supuesto un cambio en el tipo de unidad de disco.

En todas las versiones, excepto la 2.10 y la 3.10, los cambios en el DOS implicaron modificaciones importantes en los discos (incluyendo nuevos formatos de almacenamiento). El cambio principal en la 2.10 fue un cambio relativamente menor, pero también relacionado con el disco: el tiempo de asentamiento para la cabeza fue modificado para permitir trabajar con unidades de media altura usados en el PCjr y el PC portátil. La versión 2.10 también corrigió algunos de los errores conocidos de la versión 2.00. La versión 3.10 incorporó funciones de red que fueron diseñadas para la versión 3.00, pero que no estaban preparadas cuando fue lanzada. Aquí tenemos un resumen de las principales diferencias entre las versiones.

Versión 1.00. Soportaba el formato de *diskette* de simple cara y ocho sectores. Todos los servicios básicos del DOS se incluyeron en esta versión.

Versión 1.10. Añadía el soporte para *diskettes* de doble cara. Los servicios del DOS seguían siendo los mismos.

Versión 2.00. Añadía el soporte para *diskettes* con nueve sectores (tanto de simple como de doble cara) y para el disco duro. Los servicios del DOS fueron extensamente ampliados en esta versión (● véase capítulo 17). El soporte de cartuchos fue también añadido en la versión 2.00, aunque no fue conocido hasta el lanzamiento del PCjr.

Versión 2.10. No añadió nuevos formatos de disco ni nuevos servicios del DOS, aunque ajustó el tiempo de operación del disco para favorecer al PCjr y al portátil.

Versión	Fecha	Cambio hardware
1.00	4/08/81	Modelo original PC (discos de simple cara)
1.10	7/05/82	Controlador de <i>diskette</i> de doble cara
2.00	8/03/83	Modelo XT (unidades de disco duro)
2.10	20/10/83	Modelos PCjr y PC portátil (unidades de media altura)
3.00	14/08/84	Modelo AT (controlador de <i>diskette</i> de gran capacidad)
3.10	7/03/85	Redes (unidades de disco en red)

Figura 14-1. Las seis versiones del DOS y los cambios asociados al hardware

Versión 3.00. Añadió el *diskette* de gran capacidad y los formatos de disco duro adicionales. También sentó las bases para los discos en red.

Versión 3.10. Añadió los discos en red, que incluyen la capacidad de compartir ficheros.

☐ **NOTA:** *Cada versión del DOS es compatible con las versiones anteriores excepto en aspectos muy detallados (este tipo de detalles siempre parecen ser inevitables).*

Con todas las versiones del DOS, ha habido una pregunta entre los programadores a la hora de elegir una de las versiones del DOS, porque el uso de formatos más grandes para *diskette* y los servicios del DOS ampliados excluyen el uso de las primeras versiones del DOS. Esta ha sido una situación confusa y ha llevado en el pasado a tomar algunas decisiones difíciles. Las versiones del DOS-2 ha sido durante algún tiempo la elección más corriente. Las ventas de todos los modelos PC han ido creciendo, así que el número de gente que usa las versiones del DOS-1 se ha convertido en minoría dentro de la comunidad PC. Esto hace que sea relativamente poco doloroso orientar los programas a las versiones DOS-2. Incluso con la aparición de la versión DOS-3, las DOS-2 siguen siendo la mejor elección en los tiempos que corren, pero, ¿quién puede decir durante cuánto tiempo?

CONSIDERACIONES SOBRE EL FORMATO DEL DISCO

Además de decidir qué servicios del DOS o qué versión del DOS usarán los programas, también es necesario considerar qué formato de *diskette* hay que usar para nuestros programas. La convención ha sido que los programas PC se entreguen en *diskettes* con formato de simple cara y ocho sectores, ya que este formato es el mínimo común denominador de todos los formatos del DOS y puede ser usado por cualquier versión DOS. Aunque la universalidad del formato de simple cara y ocho sectores lo hacen útil, no hay muchas más razones para emplearlo. Por el contrario, los controladores de una sola cara están ahora casi en extinción en los PC: los formatos de doble cara han ocupado su lugar hace tiempo. Y además, cualquier programa que necesite el DOS-2 puede usar el formato de nueve sectores; de hecho, éste es el formato que usan las versiones 2.00 y 2.10. Para complacer los últimos propietarios del DOS-1, usted puede querer usar el formato de doble cara y ocho sectores, pero también puede

elegir el usar el formato de doble cara con nueve sectores sin remordimientos.

Un programa puede, de manera imperfecta, detectar bajo qué versiones del DOS están funcionando, usando la llamada a la función del DOS 48 (30 hex). A no ser que esté seguro de su audiencia, incluya esta protección en sus programas y compruebe siempre que la versión del DOS instalada es correcta. (■ Véase capítulo 17 para encontrar más detalles.)

COMENTARIOS

En general, la información técnica sobre el DOS es escasa; hay una gran cantidad de detalles que IBM y Microsoft parecen mantener ocultos. Desafortunadamente hay poco que podamos hacer usted y yo sobre esta escasez de información, excepto intentar comunicar lo que hemos descubierto y señalar los errores dondequiera que ocurran. Intentaremos hacerlo en los siguientes capítulo.

□ *NOTA: La fuente de información oficial sobre los servicios del DOS es el «Manual Técnico de Referencia DOS» (DOS Technical Reference Manual), que fue presentado con la versión 2.10. La información equivalente para versiones previas del DOS se encuentra en el manual principal del DOS.*

Las interrupciones del DOS

Las cinco interrupciones principales del DOS 279

Interrupción 32 (20 hex): Terminación de programa 279

Interrupciones 37 y 38 (25 y 26 hex): Lectura y escritura de disco absoluta 280

Interrupción 39 (27 hex): Terminar permaneciendo residente 283

Interrupción 47 (2F hex): Spooler o cola de impresión 284

Las tres interrupciones de dirección del DOS 286

Interrupción 34 (22 hex): Dirección de terminación 287

Interrupción 35 (23 hex): Dirección de ruptura 288

Interrupción 36 (24 hex): Dirección del manipulador de errores críticos 289

El prefijo del segmento de programa (PSP) 292

La estructura interna del PSP 294

Un ejemplo 299

En este capítulo se tratará de los servicios del DOS que se llaman con sus propias interrupciones individuales (☛ véanse los capítulos 16 y 17 para más información sobre las llamadas funciones del DOS, que son seleccionadas mediante un número de función bajo una interrupción común). Hay nueve servicios de interrupción en total, que están anotados, en la figura 15.1. Cinco de ellos, las interrupciones 32, 37, 38, 39 y 47 (20, 25, 26, 27 y 2F) son verdaderos servicios de interrupción del DOS, cada una de ellas tiene asociada una tarea específicamente definida. Las otras interrupciones tienen usos más generales. Quizá la más importante sea la interrupción 33 (21 hex), que se utiliza para invocar las llamadas a funciones del DOS. (☛ Tratadas en los capítulos 16 y 17.) Las tres interrupciones que quedan, 34, 35 y 36 (22, 23 y 24), se usan para mantener direcciones segmentadas. Nuestros programas pueden dar valores a estas direcciones (previsiblemente usando la llamada a la función 37 del DOS) como punteros de rutinas especiales. Entonces, en las circunstancias apropiadas, el DOS llama a las rutinas mediante estas tres interrupciones de direcciones. (☛ Véase página 286).

☐ *NOTA: La doctrina oficial del DOS de IBM desaprueba el uso de los servicios de interrupción del DOS por los programadores. Por consiguiente, IBM suministra llamadas a función alternativas mediante la interrupción 33 (21 hex). Como siempre existe la posibilidad de que las nuevas versiones del DOS no soporten el uso de estos servicios de interrupción, no aprobados, es aconsejable el evitar utilizarlos y confiar en las llamadas a función del DOS para los servicios especiales.*

Interrupción		Descripción
Dec	Hex	
32	20	Terminación normal de programa
33	21	Interrupción común de las llamadas a funciones
34	22	Dirección de terminación
35	23	Dirección de ruptura
36	24	Dirección del gestor de errores críticos
37	25	Lectura de disco absoluta
38	26	Escritura de disco absoluta
39	27	Terminar permaneciendo residente
47	2F	Control de cola de impresión. (Sólo versiones DOS 3.)

Figura 15-1. Los nueve servicios de interrupción del DOS

LAS CINCO INTERRUPCIONES PRINCIPALES DEL DOS

De las nueve interrupciones del DOS, cinco son verdaderas interrupciones; es decir, que tienen programas asociados de gestión de interrupciones incorporados, cada uno de los cuales realiza una tarea particular.

Interrupción 32 (20 hex): Terminación de programa

La interrupción 32 se utiliza para salir de un programa y devolver el control al DOS. Es idéntica a la llamada a la función 0 del DOS (■ véase página 304). Estos servicios pueden ser usados intercambiablemente en cualquier versión del DOS para terminar un programa.

La interrupción 32 no cierra los ficheros automáticamente cuando termina un programa, así que se debe usar siempre las funciones 16 ó 62 para cerrar todos los ficheros modificados antes de salir. Si un fichero que ha sido modificado no es cerrado adecuadamente, su longitud modificada no será registrada en el directorio.

Un programa puede fijar tres direcciones utilizables mediante las interrupciones DOS 34, 35 y 36, como veremos brevemente. Como parte de las operaciones de borrado realizadas por la interrupción DOS 32, estas direcciones son reajustadas a los valores que tenían antes de que se ajustase el programa. Es esencial reajustar estas direcciones si el programa que llamó a la interrupción 32 fue ejecutado como «hijo» de otro programa. Esto sirve para evitar que el programa «padre» utilice rutinas destinadas al «hijo». (■ Véase la función DOS 75 (48 hex) en el capítulo 17.)

□ **NOTA:** Cuando el DOS ejecuta un programa, construye un prefijo de segmento de programa (PSP) en la dirección relativa cero en el segmento de código que es señalado por el registro CS. El PSP contiene información de control que, entre otras cosas, dice al DOS dónde ir cuando se termina un programa. (■ Se trata del PSP con detalle al final de este capítulo.) El DOS necesita que el registro CS apunte al PSP cuando se llama a la interrupción 32, servicio de terminación. Si el registro CS ha sido cambiado, se interferirá la operación de este servicio.

□ **PRECAUCION:** Si el control se pasa a una subrutina mediante una llamada lejana (FAR CALL), el registro CS será modificado. Estas subrutinas no deben utilizar la interrupción 32 para terminar un programa.

Interrupciones 37 y 38 (25 y 26 hex): Lectura y escritura de disco absoluta

La interrupción 37, y su compañera, la interrupción 38, se utiliza para leer y escribir en sectores específicos del disco. Son los únicos servicios del DOS que ignoran la estructura lógica de un disco y trabajan solamente con sectores individuales, sin prestar atención a los ficheros, el directorio, o la FAT. Todos los demás servicios DOS trabajan dentro del contexto de la estructura lógica del disco.

Las interrupciones 37 y 38 son similares a los correspondientes servicios de disco de la ROM-BIOS, excepto en que los sectores son localizados mediante un método de numeración diferente. En los servicios de la ROM-BIOS, los sectores se seleccionan mediante sus coordenadas de posición tridimensionales (pista/cilindro, cara/cabeza y sector), mientras que, con las interrupciones 37 y 38, los sectores se seleccionan por sus números secuenciales de sector. (El sistema de numeración de sectores se trató en la página 117.)

La fórmula en BASIC que convierte las coordenadas tridimensionales usadas por la ROM-BIOS en los números secuenciales de sector utilizados por el DOS es como sigue:

```
NUMERO.SECTOR.DOS = (SECTOR.BIOS - 1) + CARA.BIOS
* SECTORES.POR.CARA + PISTA.BIOS * SECTORES.POR.CARA
* CARAS.POR.DISCO
```

Y aquí están las fórmulas para convertir los números secuenciales de sector en coordenadas tridimensionales.

```
SECTOR.BIOS = 1 + NUMERO.SECTOR.DOS MOD SECTORES.POR.CARA
CARA.BIOS = (NUMERO.SECTOR.DOS / SECTORES.POR.CARA)
MOD CARAS.POR.DISCO
PISTA.BIOS = NUMERO.SECTOR.DOS / (SECTORES.POR.CARA
* CARAS.POR.DISCO)
```

□ *NOTA: Para diskettes de doble cara y nueve sectores, el formato de disco más común del PC, el valor de SECTORES.POR.CARA, es 9 y el valor de CARAS.POR.DISCO es 2. También se ha de notar que las caras y las pistas están numeradas diferentemente en el sistema de numeración de la ROM-BIOS: las caras y las pistas están numeradas empezando desde el 0, pero los sectores lo están desde el 1.*

Para seleccionar un bloque de sectores, se cargan los parámetros necesarios en diferentes registros. El número de sectores se especifica en DX, y la dirección de memoria para la transferencia de los datos

se especifica en DS:BX. La unidad de disco se selecciona situando un número en el registro AL: la unidad A es 0, y la unidad B es 1.

Aunque los servicios de la ROM-BIOS trabajan con unidades de disco verdaderas, los servicios del DOS trabajan con unidades lógicas. El DOS supone que cada ordenador tiene al menos dos unidades lógicas. Si no hay ninguna otra unidad B física, el DOS la simulará utilizando la única unidad física existente como A o como B, según se necesite. Estas unidades lógicas se puede reorganizar utilizando el comando ASSIGN del DOS.

Los resultados de los servicios de interrupción 37 y 38 se reflejan en una combinación del indicador del acarreo (*carry Flag*) (CF) y los registros AL y AH. Si no hay ningún error, CF es 0. Si hay algún error (CF=1), AL y AH contienen los códigos de error en dos grupos separados y algo redundantes. Los códigos de AL de la figura 15-2 son propiedad del DOS y están basados en aquellos utilizados con el manipulador de errores críticos mediante la interrupción 36 (● véase capítulo 15, página 289), mientras que los códigos de AH de la figura 15-3 se basan en los códigos de error que suministra la ROM-BIOS (● véase capítulo 10, página 212).

Normalmente, los manipuladores de interrupción y otras rutinas de servicio dejan la pila (*stack*) limpia cuando salen, devolviéndola su

Código de error		Significado
Dec	Hex	
12	0C	Error general, no específico
11	0B	Error de lectura
10	0A	Error de escritura
8	08	Sector no encontrado
7	07	Medio desconocido: formato de disco no reconocido
6	06	Error de búsqueda: fallo en el movimiento hacia la pista requerida
4	04	Error de CRC (comprobación de redundancia cíclica): error de paridad
2	02	Unidad no preparada (por ejemplo: no hay disco, o la puerta está abierta)
1	01	Unidad desconocida: número de unidad no válido
0	00	Error de protección contra escritura: intento de escribir en un <i>diskette</i> protegido

Figura 15-2. Los valores de los códigos de error y sus significaciones depositados en el registro AL después de un error en la lectura o escritura de un disco mediante las interrupciones DOS 37 y 38

Código de error		Significado
Dec	Hex	
128	80	Desfase: la unidad de disco no respondió
64	40	Mala búsqueda: fallo en el movimiento a la pista solicitada
32	20	Fallo del controlador: funcionamiento defectuoso del controlador del <i>diskette</i>
16	10	Mal CRC: durante la lectura se encontró una comprobación inválida de la paridad de los datos
8	08	Fallo del DMA (acceso directo a memoria)
4	04	Sector malo: el sector solicitado no está en el <i>diskette</i>
3	03	Error de escritura protegida: intento de escribir en un <i>diskette</i> protegido
2	02	Mala señal de dirección: identificador del sector inválido o no se encuentra
0	00	Otros errores

Figura 15-3. Los valores y significados de los códigos de error depositados en el registro AH después de un error en la lectura o escritura mediante las interrupciones DOS 37 y 38

tamaño y contenidos originales. Las interrupciones DOS 37 y 38 no limpian la pila deliberadamente. En vez de hacer esto, terminan y vuelven el programa dejando una palabra en la pila. Esta palabra posee los contenidos del registro de indicadores, que muestra qué valores tenían éstos cuando el programa llamó al servicio. Está hecho supuestamente para preservar los indicadores que el programa tenía antes de que se utilizase el servicio, ya que las interrupciones 37 y 38 usan los indicadores para sus códigos de retorno. Creo que es una precaución tonta, ya que cualquier programa que necesite preservar los indicadores puede hacer lo que hacen normalmente los programas cuando necesitan salvar algo: un PUSH sobre la pila. Cualquier programa que utilice las interrupciones 37 y 38 debe de hacer POP de los dos bytes de los indicadores, sacándolos de la pila después de que este servicio retorne. Pueden situarse en el registro de indicadores con un comando POPF (que debe ser hecho después de comprobar si hay un error en CF), o pueden ser descartados usando el comando POP para moverlos a alguna posición extraña, como puede ser el registro DX.

Interrupción 39 (27 hex): Terminar permaneciendo residente

La interrupción 39 llama a uno de los servicios más interesantes de los suministrados por el DOS. De hecho, es tan interesante que lo hemos utilizado en el ejemplo en lenguaje ensamblador que hay al final del capítulo.

Como la interrupción 32, la interrupción 39 termina un programa, pero no lo borra de la memoria. En vez de ello, deja una porción especificada del programa en la memoria (el programa permanece residente), y el registro del DOS de la primera zona utilizable de la memoria se modifica a la dirección de segmento inmediatamente siguiente al programa residente. La información que se hace residente, usando la interrupción 39, se convierte en una extensión del DOS y no será sobrescrita por otros programas. Abundando en el esfuerzo del IBM por evitar el uso de las interrupciones DOS, existe una función del DOS, la función 49 (31 hex), que también realiza este servicio (☛ véase página 340).

La interrupción 39 (o su función equivalente) es usada por unos cuantos programas sofisticados que actúan como mejoras añadidas al DOS. Uno de los más conocidos de estos programas es ProKey, un «mejorador» de teclado. Los programas usan típicamente este servicio para establecer una nueva rutina de manipulación de interrupciones, con intención de permanecer indefinidamente. A menudo, estas rutinas de manipulación de interrupción reemplazan a manipuladores de interrupción existentes, a fin de cambiar o ampliar su operación. Pero la parte residente no tiene por qué limitarse a manipuladores de interrupción e instrucciones de programa; puede tratarse también de datos. Por ejemplo, la misma técnica de programación puede ser usada para cargar información de estado en un área común que compartirían varios programas, permitiéndoles comunicarse indirectamente.

Normalmente, un programa que utilice esta técnica querrá dejar residente sólo una parte de sí mismo, descartando, por ejemplo, el código de inicialización. Así pues, el programa debe contener al principio la porción que permanecerá residente, y debe especificar en el registro DX la dirección relativa dentro del segmento del código del primer byte que esté más allá de la porción residente. (☛ Véase el programa de ejemplo en la página 300.)

Lo que quede residente con este servicio normalmente permanecerá residente, mientras el DOS permanezca también residente. No es insólito que varios programas diferentes dejen parte de ellos residente. Como los programas que usan esta técnica son normalmente

sofisticados y complicados, tampoco es insólito que interfieran los unos con los otros. Para operar con éxito con tal grupo de programas residentes, a veces deben ser cargados en un orden particular, y el orden debe ser descubierto experimentalmente (un truco poco elegante para los usuarios sin malicia). Si se escribe un programa usando esta técnica, se debe tener gran cuidado de asegurarse de que su comportamiento es civilizado.

Como con la interrupción 32, el servicio de terminar ordinario, el DOS reajusta los vectores de dirección de las interrupciones 34, 35 y 36 (22, 23 y 24 hex) cuando se realiza el servicio de terminar permaneciendo residente. Esto quiere decir que este servicio no puede ser usado para crear manipuladores de interrupción residentes para las interrupciones de dirección. Aunque esto puede parecer una limitación, es completamente razonable; las interrupciones de dirección no están pensadas para ser usadas globalmente; sólo están pensadas para ser usadas por programas individuales. (● Véase la sección de interrupciones de dirección DOS que sigue para profundizar más sobre este tema.)

□ NOTA: Cuando un programa tipo EXE se monta (link), puede estar marcado para ser cargado en la máxima posición de memoria disponible en vez de en la mínima, como es convencional. Estos programas no pueden usar la interrupción 39, ya que está diseñada solamente para residir en la zona baja de la memoria. ● Véase página 384 para ver más sobre el montaje y los programas de tipo EXE.

Interrupción 47 (2F hex): Spooler o cola de impresión

□ NOTA: La mayor parte del material de este capítulo se aplica a todas las versiones del DOS, aunque la interrupción 47 está disponible únicamente en las versiones del DOS 3.00 y posteriores.

Con la versión DOS 3.00 se introdujo una nueva interrupción DOS, la interrupción 47 (2F hex), para dar acceso a nuestros programas a las potentes y útiles prestaciones del *spooler* de impresión del DOS. Esta interrupción es la manera estándar de comunicarse con cualquier *spooler* de impresión instalado en el DOS, tanto si es el estándar proporcionado por el comando del DOS PRINT o si no lo es.

Seis funciones diferentes forman los servicios de control del *spooler* de impresión que están disponibles mediante la interrupción

47. Estas funciones, codificadas del 0 al 5, se llaman colocando un código de función en el registro AL y, a continuación, produciendo la interrupción 47 (2F hex).

El código de función 0 informa si está instalado o no el *spooler* de impresión. El código de retorno se pasa al registro AL. Un valor de 255 (FF hex) indica que el *spooler* está instalado y, presumiblemente, puede ser usado. Un valor de 0 indica que no está instalado, pero que puede instalarse; un valor de 1 indica que el *spooler* no está instalado y no puede ser instalado.

Estos dos últimos códigos de retorno pueden parecer algo curiosos, hasta que se examinan de cerca. Un valor de retorno de 0 parece ser un mensaje del *spooler* de impresión que dice: «No estoy aquí.» Nosotros podríamos responder: «Si no estás aquí, ¿cómo es que respondes?» La respuesta es que la contestación de 0 se genera automáticamente cuando no está instalado ningún manipulador de interrupciones que trate nuestra petición. Un valor de retorno de 1 parece decir: «Estoy aquí, pero no puedes utilizarme.» Por extraño que parezca esto, es exactamente lo que está ocurriendo. Un código de retorno de 1 significa que no nos está permitido instalar el *spooler* de impresión, porque la interrupción 47 está siendo utilizada para otro propósito por algún otro manipulador de interrupciones. Este es un asunto sugestivo con el que contar.

El código de función 1 se usa para pasar un fichero al *spooler*. Para comunicar al *spooler* lo que va a ser impreso ajustamos el par de registros DS:DX para que apunte a un área de 5 bytes, llamada paquete de presentación. El primer byte del paquete es un código de nivel (del que no conozco nada en absoluto). Los cuatro bytes restantes son las direcciones segmentadas de una cadena ASCIIZ (véase capítulo 17) que definen el nombre de ruta del fichero. El del nombre de ruta debe ser un fichero único. No están permitidos los caracteres comodín * y ?.

Cuando se pasa un fichero utilizando esta función, es añadido al final de la cola, o lista de ficheros a imprimir. Los ficheros son impresos por turno y se eliminan de la cola cuando ya han sido impresos.

El código de función 2 cancela ficheros concretos que están en la cola para su impresión. El par de registros DS:DX señala a la cadena ASCIIZ que define el fichero que va a ser retirado de la cola. En este caso, pueden ser usados los caracteres comodín de nombres de fichero * y ?. Se debe advertir que el par de registros DS:DX en la función 2, a diferencia de la función 1, señala directamente a la cadena del nombre del fichero, en vez de señalar al paquete de presentación que señala la cadena.

El **código de función 3** cancela todos los ficheros que están en la cola para ser impresos. Con las funciones 2 y 3, si se cancela el fichero que se está imprimiendo, el DOS deja de imprimir el fichero e imprime un mensaje breve a tal efecto.

El **código de función 4** da acceso a la cola de impresión a los programas para que puedan examinarla. La cola queda congelada cuando se solicita esta función, así que no hay que preocuparse de los cambios en la lista mientras la inspeccionamos. Si se hace cualquier otra llamada a la función de la interrupción 47, la cola se descongelará. La función 4 devuelve un puntero en el par de registros DS:SI, que señala a una lista de los nombres de los ficheros que hay en la cola para imprimir. Los elementos de la lista son cadenas de una longitud fija de 64 bytes. El final de la lista se indica con un elemento que comience con un byte cero.

La congelación de la cola impuesta por la función 4 no necesita parar la operación de impresión, porque no es preciso. Pero sí suspenderá la retirada de la cola de un fichero que se ha terminado de imprimir.

El **código de función 5** es esencialmente una función nula que no hace nada más que descongelar la cola de nombres de ficheros congelados mediante la función 4. (Las otras cuatro funciones también pueden hacer esto.)

LAS TRES INTERRUPCIONES DE DIRECCION DEL DOS

El DOS usa tres interrupciones, desde la 34 hasta la 36 (22, 23 y 24 hex), para tratar tres circunstancias excepcionales: el final de un programa, una acción de «ruptura» desde el teclado (Ctrl-Break o Ctrl-C en el teclado PC estándar; Fn-B, en el PCjr), y cualquier «error crítico» (normalmente un error de disco de cualquier tipo). Nuestros programas pueden modificar la acción tomada en cada una de estas tres circunstancias cambiando el correspondiente vector de interrupción para señalar a cualquier operación que elijamos. Esto es por lo que llamamos a estas interrupciones las de dirección.

El DOS mantiene unas direcciones por defecto para cada una de estas interrupciones, que se preservan al principio de la ejecución del programa y se restituyen después de que el programa termina. Esto permite a los programas el cambiar libremente estos vectores, de acuerdo con sus necesidades, sin entorpecer la operación de programas siguientes, o la del mismo DOS.

También es posible que los programas cambien los valores preservados por defecto, lo que provocaría entonces un cambio

semipermanente en la operación del DOS. Los valores por defecto se almacenan en el prefijo del segmento del programa (● véase página 292). Modificando el valor del PSP, automáticamente se cambian los valores por defecto, que se realmacena cuando el programa finaliza.

Es práctica habitual que un programa, durante su ejecución, cambie las direcciones de interrupción; por el contrario, no es normal cambiar los valores por defecto que permanecerán activos hasta el final del programa.

Interrupción 34 (22 hex): Dirección de terminación

La dirección asociada a la interrupción 34 especifica dónde será pasado el control cuando finalice el programa. Esta dirección también se copia en el PSP del programa.

Normalmente, este servicio se usa para devolver el control al intérprete de comandos del DOS, COMMAND.COM, cuando termina un programa. Aunque los otros dos vectores de interrupción cubiertos en esta sección pueden ser cambiados libremente por los programas para señalar a rutinas nuevas, este vector no es usado de esta forma. Se explica mejor como usarlo mediante un ejemplo.

Un programa —llamémoslo Prog1— puede solicitar que el DOS cargue y ejecute otro programa —al que llamaremos Prog2—. Cuando el Prog2 finalice, el DOS devolverá el control a dondequiera que indique el vector de la interrupción 34. Por consiguiente, si el valor por defecto de esta interrupción está en vigor, devuelve el control al DOS y no al Prog1. Esto puede ser lo que quiere el Prog1, pero si el Prog1 desea continuar la operación después de que el Prog2 haya terminado, el Prog1 debe cambiar su vector de interrupción de dirección de terminación para que señale a una posición dentro del Prog1, *antes* de que ejecute al Prog2. Más tarde, el Prog1 debe reajustar este vector de interrupción para que el Prog1 pueda finalizar normalmente y devolver el control al DOS.

A diferencia de otras interrupciones de dirección, esta interrupción no es generada realmente nunca. En vez de esto, el vector de interrupción se usa como lugar para almacenar una dirección segmentada. Las interrupciones desde la 29 a la 31, la 68 y la 73 (1D, 1E, 1F, 44 y 49 hex) se utilizan de modo similar para almacenar direcciones (● véase página 54).

Esta es una materia exótica. No pierda el tiempo si no lo comprende. Si usted está cualificado para usar esta característica, entonces usted probablemente lo entenderá mejor que lo que yo pueda explicarle.

Interrupción 35 (23 hex): Dirección de ruptura

La dirección asociada con la interrupción 35 señala a la rutina de manipulación de interrupción que será llamada siempre que el DOS responda a una acción de ruptura desde el teclado. La clave de ruptura se genera en un teclado PC estándar mediante Ctrl-Break, o en cualquier teclado mediante Ctrl-C (un hecho que no está anunciado extensamente).

El DOS es un poco peculiar sobre cuándo responderá a una acción de ruptura desde el teclado. En una operación estándar, el DOS sólo actúa sobre una ruptura durante ciertas funciones del teclado y de la pantalla. De todos modos, el comando BREAK ON permite a las versiones DOS 2.00 y superiores actuar sobre una ruptura en cualquier oportunidad.

La respuesta por defecto del DOS en la interrupción de ruptura es finalizar el programa o el fichero *batch* que está siendo ejecutado. Si nuestros programas utilizan su propio manipulador de interrupción de ruptura, pueden hacer que el DOS tome cualquier acción que ellos deseen, sin importar lo extensa o compleja que ésta sea. Mediante esta interrupción, los programas pueden llamar a cualquier servicio del DOS y no necesitan devolver el control al DOS (aunque deberían, para evitar el crecimiento de la pila).

Las dos acciones más comunes son ignorar completamente la acción de ruptura, o usarla como señal para salir de una operación repetida. Varios programas del DOS ilustran este segundo uso. Por ejemplo, el rudimentario programa editor de textos Edlin usa la clave de ruptura para señalar el fin del subcomando I (insertar líneas). Si se quiere que un programa ignore la tecla de ruptura, use un manipulador de interrupción que devuelva el control al programa activo. Cuando se utilice la tecla de ruptura para señalar el fin de una operación repetida, el manipulador de interrupción debe colocar un dato de señal, que la parte normal del programa inspeccionará, y después devolverá el control.

Hay dos maneras por las cuales un manipulador de interrupción puede devolver el control en esta interrupción. El método normal, para cualquier manipulador de interrupción, es utilizar la instrucción IRET (retorno de interrupción). Hay otro método peculiar para esta interrupción que da al manipulador de interrupción la opción de decir al DOS si continúa con el programa o lo finaliza (del mismo modo con que manipulador de interrupciones del DOS por defecto termina los programas cuando se pulsa la tecla de ruptura (*break*)). Para usar este método, el manipulador de interrupción pone a 1 el CF (indicador de acarreo) y finaliza con una instrucción FAR RET; si CF

es 1, señala que el DOS va a detener el programa, y si CF es 0, señala que el programa va a continuar.

□ *NOTA: Los programas pueden simular pulsación de la tecla de ruptura generando esta interrupción.*

Interrupción 36 (24 hex): Dirección de manipulador de errores críticos

La dirección asociada con la interrupción 36 señala a la rutina de manipulación de interrupción, que se llama siempre que el DOS detecta un «error crítico» (una situación de emergencia que impide que la ejecución continúe). Típicamente, el error crítico es un error de disco, aunque también se informa de otros errores, como veremos.

Cuando se llama a un manipulador de interrupción, se dispone de varias fuentes de información sobre el error en sí mismo, y sobre el estado de cosas antes de que ocurriera el error. Estas fuentes incluyen el par de registros BP:SI, la pila, el registro AH y el registro DI. Se tratarán uno a uno, porque el asunto completo es bastante complicado.

Si se está trabajando con una versión DOS 2.00, o superior, el par de registros BP:SI toma valores que señalan a un bloque de control del dispositivo. Nuestro manipulador de interrupción puede examinar este bloque para conocer más sobre el dispositivo (unidad de disco, impresora, etc.) que experimentó el error (véase el «Manual Técnico de Referencia» para conocer más sobre el bloque de control del dispositivo).

La pila contiene el juego de registros completo del programa que solicitó la llamada a función, que terminó en el error crítico. Esta información puede ser muy útil para un manipulador de error que esté íntimamente integrado en el programa activo. Suponiendo que nuestro manipulador de error va a obtener acceso a la pila por medios tradicionales, podemos situar la pila con una dirección relativa situada en BP, usando estas dos instrucciones:

```
PUSH    BP
MOV     BP,SP
```

que primero salvan el viejo BP (puntero de base) y después guardan en BP el valor de SP (puntero de la pila). Para más información sobre estas instrucciones, véase página 187 (la sección titulada «Nivel 3: Código de entrada»). Habiendo hecho esto, hallaremos los contenidos de la pila mostrados en la figura 15-4.

Dirección relativa a BP	Contenidos
0	BP que hemos introducido
2	IP:CS del servicio del DOS llamado a este manipulador
6	Indicadores del servicio DOS que llamó a este manipulador
8	AX del programa que llamó al servicio del DOS
10	BX del programa que llamó al servicio del DOS
12	CX del programa que llamó al servicio del DOS
14	DX del programa que llamó al servicio del DOS
16	SI del programa que llamó al servicio del DOS
18	DI del programa que llamó al servicio del DOS
20	BP del programa que llamó al servicio del DOS
22	DS del programa que llamó al servicio del DOS
24	ES del programa que llamó al servicio del DOS
26	IP:CS del programa que llamó al servicio del DOS
30	Indicadores del programa que llamó al servicio del DOS

Figura 15-4. Los contenidos de la pila de un programa que solicitó la llamada a función DOS que terminó en un error crítico

La naturaleza de un error crítico se señala principalmente mediante una combinación del bit de mayor peso del registro AH y el byte de menor peso del registro A1 (una curiosa elección, por cierto). Si el bit de mayor peso de AH es 0 ($AH < 128$), entonces el error tiene que ver con una operación del disco; pero si el mismo bit es 1 ($AH > 127$), entonces el error puede ser otro que no sea de disco, como veremos brevemente. Cuando el error es un error de la unidad de disco ($AH < 128$), el registro AL da el número de identificación de

Bit				Significado
2	1	0	Valor	
.	.	0	0	Error de lectura
.	.	1	1	Error de escritura
0	0	.	0	Error que implica a ficheros del sistema del DOS
0	1	.	1	Error que implica a la FAT
1	0	.	2	Error que implica al directorio
1	1	.	3	Error que implica al área de datos del disco

Figura 15-5. Los valores de bits y sus errores asociados devueltos en los bits del 0 al 2 del registro AH después de llamar a la interrupción 36

la unidad (0 es la unidad A, 1 es la B, etc.). Los bits del 0 al 2 del AH dan más información sobre el error, como se muestra en la figura 15-5.

Si AH es mayor que 127, entonces el error no es necesariamente un error de disco, aunque puede serlo. Un error de disco, que es normalmente anunciado cuando AH es mayor que 127, es un error en la FAT. En las versiones DOS-1 ocurre siempre así. Para versiones 2.00 y superiores, el programa manipulador de error debe examinar el bloque de control de cabecera del dispositivo (señalado por BP:SI). Si el valor devuelto indica que el dispositivo es un disco, entonces se sabrá que es un error FAT. Además de esta excepción, AH > 127 indica una variedad de errores para un dispositivo distinto del disco, y tenemos que confiar en los códigos básicos de error transmitidos en el byte de menor peso de DI, para definir el problema exacto (el byte de mayor peso debe de ser ignorado). Los valores de los códigos de error de DI mostrados en la figura 15-6 son esencialmente los mismos que los depositados en AL para las interrupciones 37 y 38 (25 y 26 hex).

Dependiendo de las circunstancias, un manipulador de error puede necesitar algunos de los servicios de llamadas a función DOS para informar al usuario del programa de lo que está ocurriendo.

Código de error		Significado
Dec	Hex	
12	0C	Error general, no especificado
11	0B	Error de lectura
10	0A	Error de escritura
9	09	Impresora sin papel
8	08	Sector no encontrado
7	07	Medio desconocido: formato del disco no reconocido
6	06	Error de búsqueda: movimiento a la pista solicitada fallido
5	05	Mala longitud de la estructura de petición
4	04	Error CRC
3	03	Comando desconocido
2	02	Unidad de disco no preparado (ejemplo: no hay disco, o la puerta está abierta)
1	01	Unidad desconocida: número de la unidad de disco inválido
0	00	Error de protección de escritura: intento de escribir en un <i>diskette</i> protegido

Figura 15-6. Los valores del código de error depositados en el registro DI después de que se llama a la interrupción 36

Pueden usarse los servicios simples de teclado y display, números de función desde el 0 al 12 (0 al C hex), pero los servicios con números de función superiores, que implican operaciones con disco y otros dispositivos, no deben ser usados. El uso de estos servicios, mientras se está manipulando una interrupción de una operación previa, provocará un desorden del que es poco probable que el DOS se recupere.

Normalmente, una rutina de manipulación de interrupción volverá al DOS después de cumplir su función. El DOS puede tomar tres trayectorias de acción: puede ignorar el error, probar de nuevo con la operación, o abortar el programa. Nosotros indicamos al DOS la opción que queremos que tome al cargar en el registro AL uno de los valores que se muestran en la figura 15-7.

□ *NOTA: Ya que el proceso de carga requerido antes de generar esta interrupción es bastante complejo, no es apropiado que los programas simulen un error crítico generando una interrupción 36.*

AL	Acción del DOS
0	Ignora el error y sigue adelante
1	Volver a intentar la operación (podemos haber detectado y corregido el problema)
2	Detener la ejecución (el DOS solicita la interrupción 35, generando una acción de ruptura)

Figura 15-7. Los valores que pueden ser cargados en el registro AL para decirle al DOS lo que tiene que hacer siguiendo una rutina de manipulación de interrupción

EL PREFIJO DEL SEGMENTO DEL PROGRAMA (PSP)

Cuando el DOS va a cargar un programa, primero reserva una sección de memoria, llamada segmento del programa, o segmento de código. Entonces construye un bloque de control, llamado el prefijo del segmento del programa, o PSP, en los primeros 256 (100 hex) bytes. Usualmente, el programa se carga directamente después del PSP en la dirección relativa 100 hex.

El PSP contiene una mezcla de información que el DOS usa para ayudar a la ejecución del programa. Es parte de cada programa DOS,

independientemente del lenguaje en que está escrito el programa. Sin embargo, para propósitos de programación, la información almacenada en el PSP es más relevante en los programas escritos en lenguaje ensamblador que en programas escritos en lenguajes de alto nivel. Esto se debe a que con lenguajes de alto nivel, el lenguaje está normalmente a cargo del entorno de trabajo del programa, del uso de la memoria y del control de los ficheros (todo lo que concierne al PSP). Por tanto, normalmente sólo estamos interesados y sólo somos capaces de hacer un buen uso de PSP si nuestro programa está basado en lenguaje ensamblador (aunque cualquier programa que pueda averiguar el valor del registro CS puede usarlo para obtener acceso al PSP; ● hay más información sobre esto en la página 299).

Antes de describir los distintos elementos del PSP, hay que mirar la relación entre el PSP y el programa que soporta.

El PSP siempre está situado en la dirección relativa 0 dentro del segmento de código. Aunque la posición del programa está típicamente situada directamente después del PSP en la dirección relativa 100 hex, su posición puede variar dependiendo del formato de comando del programa que se utilice. Sin tener en cuenta su posición, tan pronto como el programa recibe el control, ciertos registros se ajustan para apuntar al PSP. Para un programa simple de formato .COM, todos los registros de segmento se ajustan para señalar al principio del PSP, y el programa comienza en la dirección relativa 100 hex. Para un programa más complejo de formato EXE, que utilice la operación LINK del DOS, solamente se ajustan para señalar al PSP los registros DS y ES. El programa LINK pasa los valores a los registros CS, IP, SS y SP, y consecuentemente puede ajustar la posición de comienzo del programa, guardada en CS:IP, en una posición distinta a la dirección relativa 100 hex.

Dondequiera que esté situado el programa, la relación esencial entre el programa y su PSP sigue siendo la misma. El punto primordial es que, al principio de la ejecución de un programa, tengamos acceso al PSP mediante uno de los registros de segmento. Esto quiere decir que incluso si los registros de segmento son cambiados en el curso de la ejecución de un programa, un puntero del PSP puede ser capturado en los primeros estados del programa, permitiéndonos mantener el acceso al PSP durante la ejecución del programa.

La mejor manera de explicar cómo trabajan juntos el PSP y el programa es ir directos a la estructura interna del PSP. Revelaré el propósito y el uso potencial de cada elemento a medida que los explique.

La estructura interna del PSP

Como pronto se descubrirá, el PSP contiene una mezcla de términos bastante confusa (véase figura 15-8). Los orígenes y la historia del DOS lo llevan en diferentes direcciones (hacia atrás, el primer sistema CP/M, y hacia adelante, a los entornos de sistemas operativos tipo UNIX). Como resultado, el PSP contiene elementos que sirven a diferentes propósitos y están orientados a distintos métodos de programación. No se desconcierte con nada de esto, es tan sólo que la naturaleza de PSP tiene una historia muy diversa a sus espaldas. Trataremos los elementos en el orden en que aparecieron.

El **campo 1** contiene los bytes CD 20 hex, la instrucción de la interrupción 32 (20 hex). Como se vio al tratar la interrupción 32 en este capítulo, esta interrupción es tan sólo una de las varias formas estándar de que un programa se termine. Esta instrucción se sitúa al principio del PSP (en la dirección relativa 0), así que un programa pueda terminarse simplemente saltando a esta posición cuando el CS señale al PSP. Como se debe suponer, esto no es lo más sensato que un programa puede hacer, es mejor utilizar la interrupción o la llamada a una función apropiada. Yo supongo que este esquema, de

Campo	Dirección relativa			Descripción
	Dec	Hex	Longitud	
1	0	0	2	Instrucción INT 32
2	2	2	2	Tamaño de la memoria
3	4	4	1	Reservada; normalmente 0
4	5	5	5	Llamada a la función despachadora
5	10	A	4	Vector de terminación
6	14	E	4	Vector de ruptura
7	18	12	4	Vector de error
8	22	16	22	Usado por el DOS
9	44	2C	2	Puntero de las cadenas del entorno
10	46	2E	34	Area de trabajo del DOS
11	80	50	3	Instrucciones INT33, RETF
12	83	53	2	Reservada
13	85	55	7	Extensión del FCB N.º 1
14	92	5C	9	FCB N.º
15	101	65	7	Extensión del FCB N.º 2
16	108	6C	20	FCB N.º 2
17	128	80	1	Longitud de parámetro
18	129	81	127	Parámetros
19	128	80	128	Zona de transferencia del disco (DAT)

Figura 15-8. Las partes del prefijo del segmento del programa (PSP)

apariciencia extraña, fue diseñado como una medida de seguridad. Si un programa montado (*linked*) tiene una referencia externa sin resolver, terminará con una llamada a la dirección relativa 0, lo que significa que seguirá con esta instrucción y terminará el programa.

El **campo 2** dice cuánta memoria hay disponible listando la dirección del segmento del final de la memoria del DOS. Multiplicando este número por 16 se obtiene el total de bytes que el DOS considera utilizables. El comando CHKDSK del DOS informa de la misma cantidad. Recuerde que la cantidad de memoria dada por este campo puede no ser el tamaño físico real de memoria; por ejemplo, muchos discos RAM usan la memoria en posiciones altas y reajustan el registro del DOS de donde termina la memoria utilizable. Cualquier programa que necesite hacer uso de toda la memoria disponible, debe usar este indicador de memoria para averiguar qué cantidad de memoria puede usar, en vez del similar servicio de interrupción BIOS 18.

El entorno convencional de trabajo del DOS dedica toda la memoria disponible para cada programa, que se ejecuta por separado. En sistemas con ventanas o multitarea, donde los programas individuales deben compartir la memoria con otros, un programa puede tomar lo que necesite y devolver el resto mediante el uso de la función DOS 74 (4A hex), la función SETBLOCK. Esta función satisface las convenciones DOS para el uso de la memoria, pero no puede trabajar idealmente con algunos sistemas de ventanas; tales sistemas pueden usar el campo 4 como otro indicador de la memoria disponible.

El **campo 4** es más de lo que parece. Superficialmente es una llamada larga a la función despachadora del DOS, que puede usarse (aunque realmente no debiera) para llamar a funciones del DOS. Pero para lo que es realmente bueno este campo es para determinar indirectamente si nuestros programas tienen menos de 64K para trabajar. Como una llamada, esta instrucción contiene la dirección de la rutina despachadora de una función del DOS; como una llamada larga, contiene la dirección en un formato segmentado. Mediante un proceso demasiado extrafalario y complicado de explicar, la dirección segmentada se ajusta para que sirva para dos propósitos: no sólo apunta a la función despachadora de una función del DOS, sino que la parte de la dirección relativa también indica qué cantidad de código de segmento se puede utilizar (hasta hex FFF0, 16 bytes menos de 64K). La parte de la dirección relativa, la que nos interesa, está situada en la dirección relativa 6 dentro del PSP, a continuación del código de instrucción, situado en la dirección relativa 5.

El resultado de esto es que si el DOS tiene menos de 64K para

darle a nuestros programas, podemos usar este campo para saber cuántos bytes hay disponibles, una técnica que debería funcionar con la mayoría o con todos los sistemas de ventanas, o multitarea. Si el DOS puede darnos más de 64K, podemos conocer cuántos más mirando el campo 2. Sin embargo, como se dijo antes, este campo puede no darnos una respuesta adecuada en algunos entornos de ventanas, o multitareas.

Los campos 5, 6 y 7 contienen las direcciones segmentadas de las tres interrupciones de dirección que tratamos anteriormente en este capítulo. Los valores por defecto contenidos en estas posiciones se preservan al comienzo de la ejecución del programa y se restauran al final, esto permite a nuestros programas usar diferentes vectores de servicio, mientras el programa está ejecutándose, sin alterar la operación de programas siguientes, o al mismo DOS. Si no se proporcionan nuevos vectores, el DOS usa los valores por defecto almacenados en el PSP para señalar a la rutina de servicio por defecto. Si modificamos las direcciones almacenadas en estos campos, se pueden cambiar los valores por defecto (y sus rutinas de servicio asociadas), que son realmacenadas cuando termina el programa, provocando de este modo un cambio semipermanente en la operación del DOS. Bajo circunstancias normales, no hay buenas razones para cambiar ni tampoco para leer estos valores.

El campo 9 contiene una dirección de segmento que señala a las cadenas del entorno apoyadas por el DOS 2.00 y versiones posteriores. El entorno comienza en la dirección relativa.

□ *NOTA: Para evitar confusiones sobre la terminología, se utiliza la palabra «entorno» en este contexto para referirse al conjunto de cadenas (definidas en un momento) que el DOS usa para comunicar ciertos tipos de información entre programas. En algunos otros sitios de este libro se ha utilizado la palabra «entorno» para referirse a las convenciones de operación bajo las que trabaja un programa.*

Este entorno del DOS es una colección de cadenas ASCII; esto es, cadenas de caracteres ASCII con un CHR\$(0) que marca sus finales, que puede definir varias clases de información. El final de cada valor ajuste del entorno se marca por una cadena de longitud cero (CHR\$(0)), donde esperaríamos hallar el primer byte de la siguiente cadena. Si un valor del entorno comienza con CHR\$(0), entonces, no hay ninguna cadena en él.

Por convención, cada cadena individual del entorno es de la forma NOMBRE=valor, donde NOMBRE está escrito en mayúscu-

las y es de cualquier longitud razonable, y valor puede ser cualquier cosa. El DOS ajusta un entorno para el procesador de comandos, el cual es pasado a cualquier programa que lo llame. Normalmente, este entorno contendrá al menos el nombre COMSPEC (usado por el DOS para encontrar en el disco el fichero COMMAND.COM); y puede contener también nombres como PATH o SWITCHAR. El comando SET del DOS puede ser usado para añadir, cambiar o borrar cadenas en el entorno.

El **campo 11** contiene dos instrucciones que llamarán a una función del DOS (la interrupción 33, 21 hex) y vuelven al llamador (retorno RETF o FAR). Esta es otra forma que nos permite llamar funciones DOS semiindirectamente. Para usar esta característica, ajustamos todo lo que sea necesario para llamar a una función del DOS (seleccionando la función en AH, etc.) y entonces, en vez de realizar por las buenas una interrupción 33 (una instrucción de 1 byte), hacemos una llamada lejana a la dirección relativa 50 hex en el PSP (una instrucción de 5 bytes).

Se puede suponer que esta característica es otro reflejo del pasado, una parte de la compatibilidad del CP/M, pero fue introducida realmente con el DOS 2.00 y no funciona con versiones previas. Así que podemos suponer que esta forma de llamar a servicios del DOS, tan tosca como aparece, se diseñó mirando al futuro, y no al pasado.

Los **campos 13, 14, 15 y 16** apoyan al proceso de ficheros al viejo estilo, usando bloques de control de ficheros, o FCB. Los FCB pueden ser usados para entrada/salida (I/O) de ficheros con cualquier versión del DOS, pero su uso es desaprobado por la versión DOS 2.00 y posteriores, donde se dispone de una entrada/salida (I/O) moderna mediante el uso de números de ficheros. ■ Véanse página 324, para encontrar información sobre los bloques de control de ficheros, y la página 336, para los números de ficheros.

Este área del PSP es ajustada de tal modo, para hacer la vida fácil a los programas, que reciben uno o dos nombres de fichero como parámetros. La idea básica, y creo que una buena idea, es permitir al DOS que construya los FCB necesarios a partir de los dos primeros parámetros del programa (los parámetros en la línea de comando, que siguen al nombre del programa). Si un programa necesita uno o los dos FCB, puede abrirlos, usarlos, sin tener que decodificar los parámetros de comando y construir los FCB él mismo.

Si se usa esta característica del PSP, hay algunas complicaciones de las que hay que estar prevenido. Primero, los dos FCB coinciden donde están situados. Si el programa sólo necesita el primero, estarán bien; pero si necesita también el segundo, uno o los dos deben ser

movidos a algún sitio antes de ser usados. También se debe estar prevenido de que estos FCB pueden traer consigo extensiones de FCB, un hecho que se pasa por alto en la mayor parte de la documentación del DOS sobre el PSP. He llamado la atención sobre el hecho, proporcionando la documentación de las extensiones del FCB implicadas en la figura 15-8.

Recuerde que el uso de FCB se considera algo obsoleto, pero si se quiere utilizarlos, esta información ayudará a hacerlo.

Los campos 17 y 18 dan acceso a nuestros programas a los parámetros introducidos en la línea de comando. El campo 17 da la longitud completa de la cadena del parámetros (que puede ser tan pequeña como 0, o tan grande como 127), y el campo 18 da los contenidos.

Aquí hay algunas peculiaridades sobre la cadena que se pasa: no contiene el nombre del programa que fue llamado. En vez de esto, comienza con el carácter que va inmediatamente después del nombre del programa, que es normalmente un blanco. Los separadores, como espacios en blanco o comas, no están eliminados ni comprimidos. Si usamos la línea de comando, tenemos que estar preparados para analizarla, reconociendo los separadores estándar. A partir de DOS 2.00, la línea de comandos se descompone de una forma particular: cualquier parámetro de redirección (tales como <ENTRADA o >SALIDA) son extraídos por el DOS, y la línea de parámetros se reconstruye como si no estuvieran allí. Como resultado de estas operaciones en la cadena de comando, un programa no puede averiguar si su entrada/salida estándar está siendo redireccionada, ni su propio nombre.

□ *NOTA: Los campos 17 y 18 se solapan en parte con el campo 19 en el PSP, así que obtenga sus parámetros mientras pueda; el siguiente campo puede borrarlos.*

El campo 19 es la zona de transferencia del disco (DTA) tomada por defecto. Es una zona de *buffer* de 128 bytes que comienza en la dirección relativa 80 hex del PSP y se establece sólo en el caso de que se utilice un servicio del DOS que necesite una DTA y no se haya fijado todavía del *buffer* adecuado. Véanse los capítulos 16 y 17 para las descripciones de los servicios que usan o manipulan el DTA.

Considerando al PSP como un todo, hay que advertir que el DOS hace uso de varias zonas en los primeros 92 bytes del PSP de 256 bytes de longitud total (campos 1 al 13, dirección relativa 0 al 5B). Si se quiere modificar algo del PSP, se debe restringir a los campos 14 al 19 (desde la dirección relativa 5C a la FF). El programa de ejemplo

que sigue ilustra cómo puede dejarse intacta la primera parte del PSP, mientras que la segunda parte es usada por un programa residente.

UN EJEMPLO

Para el ejemplo de este capítulo, se creará un programa que use la interrupción del DOS de terminar quedando residente. La mayoría de nuestros ejemplos en lenguaje ensamblador son interfaces entre un lenguaje de alto nivel y las rutinas de servicio de la ROM-BIOS, o del DOS. Este ejemplo es algo diferente: es un programa autosuficiente en lenguaje ensamblador que usa la interrupción 39 para dejar a parte de sí mismo residente.

Ya que se van a ver algunas cosas que aún no se han tratado en este libro, son necesarias unas pocas anotaciones. Como se acaba de aprender en la sección anterior, cuando un programa se carga para ser ejecutado, el DOS construye un PSP delante de él, con una longitud de 256 bytes (100 hex bytes); el programa cargado va inmediatamente a continuación. El registro CS se ajusta para señalar al PSP, lo cual hace fácil al programa el acceso a la información del PSP. Además, el valor de IP, que indica la dirección relativa de comienzo o desplazamiento del programa, es siempre el mismo: 100 hex. Se hace uso de este valor cuando se utiliza CS para calcular la dirección relativa de la parte residente de nuestro programa en la línea que lee:

```
DESP.RESIDENTE EQU COMIENZO_RESIDENTE - PRINCIPIO + 100H
```

Generalmente, para un programa residente es un despilfarro de 256 bytes de memoria dejar el PSP en su lugar, el DOS todavía necesita al PSP para completar el proceso de terminación. Para evitar el derroche de todo este espacio se pueden hacer dos cosas: primero, si el programa residente necesita usar una zona de datos cuando se reactive como un manipulador de interrupción, puede usar la zona del PSP, ya que el DOS no necesita al PSP una vez que la terminación del programa se completa; segundo, el programa puede relocalizar su parte residente sobre la parte del PSP que el DOS no necesita durante la terminación. En nuestro ejemplo mostraremos cómo hacer esto, escribiendo encima de todo el PSP, excepto los primeros 92 bytes (5C hex bytes). Hacer esto es completamente seguro, pero si se reutilizan más bytes se puede interferir con el DOS.

Si se experimenta con esta operación y después se utiliza la función del DOS CHKDSK para probar cuánta memoria ha sido consumida, se encontrará que el DOS utiliza generalmente al menos otros 16 bytes para mantenimiento.

```

; ejemplo de la interrupción 39 (hex27): terminar-permaneciendo-
residente
PROGRAMA SEGMENT PARA PUBLIC 'CODE'
    ASSUME CS:PROGRAMA
TBSR    PROC
    COMIENZO:
; para empezar, saltamos al código de inicialización
    JMP     INICIALIZA
    COMIENZO_RESIDENTE
; aquí aparecería lo que quisiéramos dejar residente
    FIN_RESIDENTE:
    LONGITUD_RESIDENTE EQU  FIN_RESIDENTE - COMIENZO_RESIDENTE
    DESP_RESIDENTE     EQU  COMIENZO_RESIDENTE - COMIENZO + 100H
    CANTIDAD_PSP       EQU  92          ; la parte de PSP que guardamos
; sigue el código de inicialización; descartado cuando terminamos
    INICIALIZA:
; aquí iría cualquier código de inicialización que se necesitase para nuestro propósito:
; después va el código que moverá la parte residente sobre la mayoría del PSP:
    PUSH    CS          ; ajustar el registro de segmento...
    POP     ES          ; ...del destino
    PUSH    CS          ; ajustar el registro de segmento...
    POP     DS          ; ...de la fuente...
    MOV     DI,CANTIDAD_PSP ; ajustar el desplazamiento de destino
    MOV     SI,DESP_RESIDENTE ; ajustar el desplazamiento de la fuente
    MOV     CX,LONGITUD_RESIDENTE ; ajustar la cantidad a mover
    CLD          ; ajustar el sentido de movimientos
    REP MOVSB      ; mover los bytes residentes al PSP
; ahora estamos listos para terminar; fijamos la primera posición disponible después
; del código residente
    MOV     DX,CANTIDAD_PSP + LONGITUD_RESIDENTE
    INT     27H
TBSR    ENDP
PROGRAM ENDS
    END

```

Las funciones universales del DOS

Resumen de las funciones universales 302

- Función 0: Terminar 304
- Función 1: Entrada por teclado con eco 304
- Función 2: Salida por pantalla 305
- Función 3: Entrada serie 306
- Función 4: Salida serie 306
- Función 5: Salida por impresora 306
- Función 6: Entrada/salida directa por teclado/pantalla 306
- Función 7: Entrada directa sin eco por el teclado 307
- Función 8: Entrada por teclado sin eco 307
- Función 9: Salida por pantalla de una cadena 308
- Función 10 (A hex): Entrada por el teclado con *buffer* 308
- Función 11 (B hex): Comprobación del estado de la entrada por teclado 310
- Función 12 (C hex): Borrar el *buffer* de teclado y ejecutar una función 310
- Función 13 (D hex): Reinicializar el disco 310
- Función 14 (E hex): Seleccionar la unidad de disco activa 310
- Función 15 (F hex): Abrir un fichero 311
- Función 16 (10 hex): Cerrar un fichero 312
- Función 17 (11 hex): Buscar un primer fichero 312
- Función 18 (12 hex): Buscar el siguiente fichero 314
- Función 19 (13 hex): Borrar un fichero 314
- Función 20 (14 hex): Leer un registro de un fichero secuencial 314
- Función 21 (15 hex): Escribir un registro en un fichero secuencial 315
- Función 22 (16 hex): Crear un fichero 315
- Función 23 (17 hex): Renombrar un fichero 316
- Función 24 (18 hex): Usado internamente por el DOS 316

- Función 25 (19 hex): Informar sobre la unidad de disco activa 316
- Función 26 (1A hex): Fijar el área de transferencia del disco 316
- Función 27 (1B hex): Obtener la información de la FAT de la unidad activa 317
- Función 28 (1C hex): Obtener la información de la FAT de cualquier unidad 317
- Función 33 (21 hex): Leer un registro de un fichero aleatorio 318
- Función 34 (22 hex): Escribir un registro en un fichero aleatorio 318
- Función 35 (23 hex): Obtener el tamaño de un fichero 319
- Función 36 (24 hex): Inicializar un registro de un fichero aleatorio 319
- Función 37 (25 hex): Fijar un vector de interrupción 319
- Función 38 (26 hex): Crear un segmento de programa 320
- Función 39 (27 hex): Leer registro de un fichero aleatorio 320
- Función 40 (28 hex): Escribir registros en un fichero aleatorio 321
- Función 41 (29 hex): Analizar el nombre de un fichero 321
- Función 42 (2A hex): Obtener la fecha 323
- Función 43 (2B hex): Poner la fecha 323
- Función 44 (2C hex): Obtener la hora 323
- Función 45 (2D hex): Poner la hora 324
- Función 46 (2E hex): Activar la verificación de la escritura en disco 324

El bloque de control del fichero 324

Un ejemplo 329

En este capítulo se va a hablar de las funciones que son universales para todas las versiones del DOS. Empleando terminología del DOS, los servicios antiguos (o universales) que se tratan en este capítulo se denominan «funciones tradicionales». Los nuevos servicios introducidos con el DOS 2.00, y tratados en el capítulo 17 se denominan «funciones ampliadas».

RESUMEN DE LAS FUNCIONES UNIVERSALES

Todas las llamadas a las funciones del DOS se realizan mediante la interrupción 33 (21 hex). Las funciones individuales se seleccionan situando el número de la función en el registro AH.

Las llamadas a las funciones del DOS tradicionales están organizadas en los grupos lógicos que aparecen en la figura 16-1. Haciendo un esfuerzo por hacer la figura lo más clara posible, he organizado y descrito estas llamadas a función de una manera un poco diferente a como están en el «Manual Técnico de Referencia del DOS». La figura 16-2 muestra las llamadas a las funciones de forma individualizada.

Antes de entrar en detalles con estas funciones, quiero advertirle que algunos aspectos del diseño y organización de algunas de estas funciones, particularmente las comprendidas entre la 1 y la 12, son estrafalarios (por no decir algo más). Son así, por razones históricas. Muchos de los detalles del DOS, y especialmente los detalles de las funciones del DOS, se diseñaron para imitar prácticamente los servicios proporcionados por el CP/M. Esta fue una elección importante y meditada, y se realizó con objeto de hacer mucho más fácil la conversión del *software* de 8 bits del CP/M al de 16 bits del IBM PC y del DOS. Aunque la creación del DOS proporcionó la oportunidad de abandonar y suprimir los fallos del pasado, esta oportunidad no fue

Función		Grupo
Dec	Hex	
0	00	Función sin dispositivo
1-12	01-0C	Entrada/salida de un carácter en un dispositivo
13-36	0D-24	Manejo de ficheros
37-38	25-26	Más funciones sin dispositivo
39-41	27-29	Más manejo de ficheros
42-46	2A-2E	Más funciones sin dispositivo

Figura 16-1. Los grupos lógicos de las llamadas a funciones tradicionales del DOS

Función			Función		
Dec	Hex	Descripción	Dec	Hex	Descripción
0	0	Terminar: fin del programa	24	18	Usado internamente por el DOS
1	1	Entrada por teclado con eco	25	19	Informar sobre la unidad de disco activa
2	2	Salida por pantalla			
3	3	Entrada serie	26	1A	Fijar el área de transferencia al disco
4	4	Salida serie	27	1B	Obtener la información de la FAT de la unidad activa
5	5	Salida por impresora			
6	6	Entrada/salida directa por teclado/pantalla	28	1C	Obtener la información de la FAT de cualquier unidad
7	7	Entrada directa sin eco por el teclado	24	1D	Usado internamente por el DOS
8	8	Entrada por el teclado sin eco	30	1E	Usado internamente por el DOS
9	9	Salida de cadena caracteres por pantalla	31	1F	Usado internamente por el DOS
10	A	Entrada por el teclado con <i>buffer</i>	32	20	Usado internamente por el DOS
11	B	Comprobación del estado de la entrada por teclado	33	21	Leer un registro de un fichero aleatorio
12	C	Borrar el <i>buffer</i> de teclado y hacer una función	34	22	Escribir un registro en un fichero aleatorio
13	D	Reinicializar el disco	35	23	Obtener el tamaño de un fichero
14	E	Seleccionar la unidad de disco activa	36	24	Inicializar un registro de un fichero aleatorio
15	F	Abrir un fichero	37	25	Fijar un vector de interrupción
1	10	Cerrar un fichero	38	26	Crear un segmento de programa
17	11	Buscar el primer fichero	39	27	Leer registros de un fichero aleatorio
18	12	Buscar un primer fichero	40	28	Escribir registros en un fichero aleatorio
19	13	Borrar un fichero			
20	14	Leer un registro de un fichero secuencial	41	29	Analizar el nombre de un fichero
21	15	Escribir un registro en un fichero secuencial	42	2A	Obtener la fecha
22	16	Crear un fichero	43	2B	Poner la fecha
23	17	Renombrar un fichero	44	2C	Obtener la hora
			45	2D	Poner la hora
			46	2E	Activar la verificación

Figura 16-2. Las llamadas a las funciones tradicionales del DOS activadas mediante la interrupción 33 y seleccionadas en el registro AH

aprovechada (acción desafortunada, pero prudente, en mi opinión). El rediseño limpio de los servicios del DOS empezó realmente durante el desarrollo de la versión 2.00 y se realizó con las funciones ampliadas (☛ Véase capítulo 17.)

En las páginas que siguen detallaremos las 46 llamadas a las funciones del DOS, que se utilizan universalmente en todas las versiones del DOS.

Función 0: Terminar

La función 0 del DOS se usa para finalizar un programa y devolver el control al DOS. Es funcionalmente igual a la interrupción 32 (20 hex) del DOS de la que se habló en el capítulo 15. Cualquiera de los dos servicios se puede utilizar para salir de un programa.

Las versiones del DOS 2.00 y superiores proporcionan una forma de finalizar mediante la función 76 (4C hex), que deja un código de retorno (un código de error) en el registro AL cuando termina un programa. Los ficheros *batch* pueden utilizar estos códigos de retorno usando el subcomando del DOS ERRORLEVEL. Se utilizará la función 76 en vez de la función 0 si se desea registrar cuando finaliza un programa cualquier error que haya podido ocurrir. (● Véase capítulo 17).

Al igual que la interrupción 32 del DOS, esta función no cierra los ficheros automáticamente cuando el programa termina. Para asegurar que la longitud de un fichero que acaba de modificarse queda registrada en el directorio del fichero, use las funciones de cerrar ficheros 16 ó 62 antes de llamar a la función 0. Además, le corresponde al programa asegurarse de que la dirección PSP está en el registro CS antes de salir. (Como recordará, PSP contiene la dirección de terminación que le dice al DOS dónde ha de ir cuando un programa ha terminado; esto ya se trató en el capítulo 15.)

Función 1: Entrada de teclado con eco

La función 1 espera la introducción de un carácter por el dispositivo estándar de entrada, y lo deposita en el registro AL cuando lo recibe. Es comparable a otras funciones del teclado; en particular, se parece a las funciones 6, 7 y 8.

Así trabaja la función 1: la pulsación de las teclas dan como resultado un carácter ASCII, que se deposita como un byte en AL, e inmediatamente este servicio informa del suceso. Las 97 acciones especiales de teclado que darán lugar a algo distinto de un carácter ASCII (● véase página 150) generan dos bytes que necesitan dos llamadas consecutivas a este servicio.

La manera estándar de usar este servicio es ver si hay un 0 en AL. Si AL no es 0, es que existe un carácter ASCII. Si AL es 0, es que hay un carácter especial (que debe ser leído), y por lo tanto la función debe ser repetida inmediatamente para obtener el código que representa la acción asociada a la tecla especial. (● Véase página 152, donde hay una lista con las teclas, los códigos y sus significados.)

Como con todos los servicios de entrada a través del teclado del DOS, el código de búsqueda de caracteres ASCII no está disponible, aunque los servicios de la ROM-BIOS los hacen accesibles mediante lo que se denomina byte auxiliar. (● Véase página 358.)

Las funciones de servicio de teclado del DOS se clasifican principalmente por tres criterios: si esperan una entrada o si informan de que no ha habido entrada; si reflejan la entrada en la pantalla o no; y si la operación de la tecla de ruptura (*break-key*) está permitida. (Recuerde que las operaciones del DOS estándar asociadas a Ctrl-Break o Ctrl-C sólo están activas en un número limitado de operaciones. De cualquier modo, a partir de la versión 2.00, el DOS introdujo el comando BREAK ON, que le da al DOS la posibilidad de actuar sobre la tecla de ruptura bajo cualquier circunstancia.) La función 1 realiza 3 operaciones: espera la entrada, refleja la entrada por pantalla, y si detecta una operación de tecla de ruptura, ejecuta la interrupción 35, que es la interrupción de dirección de ruptura.

□ *NOTA: En las versiones del DOS 2.00 y superiores, este servicio de «teclado» está asociado al dispositivo de entrada estándar del DOS. Por defecto es el teclado, pero puede ser redirigido a cualquier otro dispositivo de entrada.*

● Si desea evitar el tener que esperar cuando la entrada no está lista, pero desea usar este servicio, véase la función 11, que además informa de si la entrada está lista o no. Véanse las funciones 8 y 12 para ver variaciones sobre este servicio.

Función 2: Salida por pantalla

La función 2 permite escribir un único carácter ASCII en la pantalla (y en las versiones del DOS 2.00 y posteriores, en el dispositivo estándar de salida). El carácter se sitúa en DL.

En general, este servicio actúa con los caracteres de control ASCII, tal como el de retroceso (*backspace*) o el de retorno del carro (*carriage return*). En el caso del carácter de retroceso, el cursor de la pantalla puede moverse hacia atrás hasta el principio de la línea actual. Contrariamente a la información que aparece en el «Manual Técnico de Referencia del DOS» (y en todos los manuales del DOS desde el primero hasta el último), no se escribe el carácter blanco (20 hex) después de que el cursor se ha movido, lo que borraría cualquier carácter previo. En vez de esto, cualquier información existente permanece intacta.

Función 3: Entrada serie

La función 3 introduce un carácter en AL desde el dispositivo auxiliar estándar, que se conoce normalmente como AUX: o COM1: Por arte de magia del comando MODE, se puede alterar para recibir entradas desde otros dispositivos, tales como el COM2:. Normalmente, la fuente de los datos de entrada es el primer puerto serie RS-232.

☐ *NOTA: Este servicio espera una entrada y no da información de estado sobre las muchas desgracias que puede sufrir un puerto serie. Si quiere conocer el estado del puerto serie, use los servicios de la ROM-BIOS (☛ véase capítulo 12.)*

Función 4: Salida serie

La función 4 saca el carácter contenido en el registro DL por el dispositivo auxiliar estándar. (☛ Véanse los comentarios de la función 3.)

Función 5: Salida por impresora

La función 5 saca un byte almacenado en DL por el dispositivo de impresión estándar, normalmente conocido como PRN: o LPT1: (pero con el comando MODE puede ser cualquier otro dispositivo). En ausencia de cualquier redirección DOS, la impresora estándar es siempre la conectada al primer adaptador en paralelo, incluso si se usa un puerto en serie para la salida de la impresora.

Función 6: Entrada/salida directa por teclado/pantalla

La función 6 es un servicio complejo y estrafalario que combina las operaciones de entrada por teclado y de salida por pantalla, en un paquete poco metódico. Como con todo, en las versiones del DOS 2.00 y posteriores, la entrada/salida no está asociada al teclado y a la pantalla, sino más bien a los dispositivos estándar de entrada y salida (los cuales son por defecto el teclado y la pantalla).

Así es como trabaja este servicio: el registro AL se usa para la entrada y el DL para la salida. Si DL es 255 (FF hex), entonces AL está preparado para recibir un carácter de entrada. El indicador de cero (ZF) señala si la entrada está lista. Si ZF es 1, no está la entrada lista, y si ZF es 0, hay un byte de entrada en AL. Si DL no es 255,

entonces se supone que contiene un carácter de salida legítimo, que se envía al dispositivo de salida estándar.

La función 6 no espera a que se produzca una entrada por teclado, ni hace eco por la pantalla, y la operación de la tecla de ruptura (*break-key*) no está activa. (☛ Véase la función 1 para encontrar una explicación.)

☛ Compare este servicio con las funciones 1, 7 y 8. Véase la función 12 para encontrar una variación de este servicio.

Función 7: Entrada directa sin eco por el teclado

La función 7 espera a que se produzca la entrada de un carácter por el dispositivo estándar de entrada, y lo deposita en el registro AL. No refleja la entrada por la pantalla y no admite la operación de la tecla de ruptura (*break-key*).

La función 7 trabaja de la misma forma que la función 1: las pulsaciones de las teclas de carácter ASCII se depositan en el registro AL como bytes únicos. Las 97 acciones especiales de teclado, que no dan como resultado caracteres ASCII (☛ véase página 150), generan los bytes, que son almacenados mediante dos llamadas consecutivas a este servicio.

La manera estándar de usar este servicio es buscar un 0 en AL. Si AL no es 0, tenemos un carácter ASCII. Si AL es 0, es que hay un carácter especial. Como ese carácter debe ser leído, entonces se repite la función para obtener el pseudo-código de exploración que representa a la tecla especial pulsada. (☛ Véase página 152, donde existe una lista de las teclas, los códigos y sus significados.) Como en el caso de los servicios de entrada del teclado del DOS, los códigos de búsqueda de los caracteres ASCII no están disponibles, aunque los servicios de la ROM-BIOS permiten conocerlo mediante lo que se denomina byte auxiliar. (☛ Véase página 150.)

☛ Compare este servicio con las funciones 1, 6 y 8. Si quiere usar este servicio, pero evitando esperar cuando la entrada no está lista, véase la función 11, que informa de si está lista la entrada o no. Véase la función 12 para ver una variación de este servicio.

Función 8: Entrada por teclado sin eco

La función espera una entrada, no produce eco en la pantalla y se interrumpe cuando se pulsa la tecla de ruptura. Es idéntica a la función 1, excepto en que no refleja la entrada en la pantalla (ni en el dispositivo estándar de salida).

☛ Véase lo que se habló sobre la función 1 para encontrar la descripción de esta función. Compare este servicio con las funciones 1, 6 y 7. Si quiere usar este servicio, pero evitando el esperar cuando la entrada no está lista, véase la función 11, que informa de si está o no está lista la entrada. Véase la función 12 para encontrar una variación de este servicio.

Función 9: Salida por pantalla de una cadena

La función 9 envía una cadena de caracteres a la pantalla o al dispositivo estándar de salida (que por defecto es la pantalla). El par de registros DS:DX proporciona la dirección donde empieza la cadena, y el carácter '\$', CHR\$(36), se usa para señalar el final de la cadena.

☐ *NOTA: Por malas razones históricas, en la literatura del DOS se hace referencia a esta función como «impresión de cadena». Sería mucho mejor que pensáramos en ella como «salida por pantalla de una cadena», que es el nombre que se le ha dado aquí.*

Mientras que este servicio puede ser tremendamente más útil y conveniente que los servicios de pantalla byte a byte (funciones 2 y 6), queda enturbiado por el uso de un carácter representable por pantalla, \$, como delimitador de cadena. Esto no es un error reciente; es otro subproducto de la compatibilidad con el CP/M. A menos que sepa que no hay absolutamente ninguna posibilidad de sacar un signo dólar, debe evitar el uso de este servicio.

☛ A propósito, las funciones del DOS ampliadas (véase capítulo 17) usan CHR\$(0) como delimitador de la cadena. Esta práctica sigue la convención ajustada por el sistema operativo UNIX y por el lenguaje de programación C.

Función 10 (A hex): Entrada por el teclado con buffer

La función 10 es un servicio estupendo que pone la potencia de las teclas de edición del DOS a trabajar en los programas. El servicio obtiene como entrada una cadena completa, que es pasada a los programas toda entera, en vez carácter a carácter. En el caso de que la entrada provenga en realidad de teclado, y que no haya sido redirigida desde ningún otro sitio, la persona que está escribiendo la cadena de caracteres tiene disponible las teclas de edición del DOS. Cuando la tecla de Intro es pulsada (o aparece en el fichero de

entrada el retorno del carro (*carriage return*), CHR\$(13)), la operación de entrada queda completada, y la cadena pasa al programa.

Aparecen muchas ventajas cuando se usa este servicio, particularmente cuando los programas necesitan cadenas de caracteres completas y coherentes, en vez de entradas byte a byte. Los dos beneficios principales que tiene son: que hace innecesario realizar el esfuerzo de escribir programas específicos para realizar el tratamiento de la entrada, y que a los usuarios de los programas se les da un conjunto familiar de herramientas para realizar la edición de entrada: las convenciones de edición del DOS.

Para usar este servicio, se debe suministrar al DOS un área para el *buffer* de entrada, donde serán construidas las cadenas introducidas. El par de registros DS:DX forma el puntero de este *buffer*. Se le informa el DOS del tamaño de trabajo del *buffer*, colocándolo en el primer byte del *buffer* el número de bytes que el DOS puede usar para formar una cadena de entrada. El segundo byte de *buffer* lo utiliza el DOS para informar del número efectivo de bytes que se han introducido. El DOS coloca la cadena de entrada, que estará formada completamente por caracteres ASCII, a partir del tercer byte del *buffer*, inclusive. El final de la cadena de entrada se marca con el carácter de retorno de carro, CHR\$(13). El retorno de carro se sitúa dentro del *buffer*, por lo que es necesario contar con sitio para él, pero no está incluido en la cuenta de los caracteres que realiza el DOS y que se almacena en el segundo byte.

Según estas reglas, el *buffer* más grande que puede tener es de 255 bytes de trabajo, y la cadena más larga que el DOS puede devolver es una unidad menor que la longitud de trabajo. Debido a los dos primeros bytes del *buffer* que se usan para dar la información de estado, el tamaño efectivo de trabajo del *buffer* es dos bytes menor que el tamaño total del *buffer*. Esto puede explicar algunos de los misterios que hay en las convenciones sobre las entradas que tienen el DOS y el BASIC. Si la cadena es más larga de lo que el DOS puede situar en el *buffer* (que es un byte menos de su longitud de trabajo), entonces el DOS rechaza cualquier entrada adicional, produciendo *beeps*, hasta que se encuentra un retorno de carro.

Puede probar algunos elementos de esta función contando las entradas que aceptará el intérprete de comandos del DOS. Experimentos simples revelarán que el intérprete de comandos usa un tamaño de trabajo del *buffer* de 128 bytes (la longitud total es de 130 bytes). El DOS protestará por cualquier entrada que no sea el retorno de carro (*carriage return*) al sobrepasar el 127 byte.

● Véase la función 12 para encontrar una variación de este servicio.

Función 11 (B hex): Comprobación del estado de la entrada por teclado

La función 11 informa de si hay alguna entrada lista desde el teclado (o desde el dispositivo estándar de entrada). Es especialmente útil en conjunción con los servicios de entrada del teclado sin espera, funciones 1, 7 y 8. Se informa de que la entrada está lista cuando el contenido de AL es 255 (FF hex). Si no es así, AL contiene un 0.

La operación estándar de tecla de ruptura está activa en este servicio. (● Véase lo que se habló en la función 1 para obtener una explicación.)

Función 12 (C hex): Borrar el buffer de teclado y ejecutar una función

La función 12 limpia el *buffer* asociado al teclado situado en RAM y llama a uno de cinco servicios del DOS: función 1, 6, 7, 8 o A. El registro AL se usa para seleccionar cuál de estas funciones será realizada después de que el *buffer* se vacíe. Con el *buffer* limpio de caracteres extraños, esta función fuerza a que el sistema espere una nueva entrada antes de que actúe la función llamada.

Advierta que desde que, dado que se puede llamar la función 6, el servicio complementario no tiene por qué ser una entrada por teclado, puede ser una salida por pantalla.

Función 13 (D hex): Reinicializar el disco

La función 13 reinicializa el disco y borra todos los *buffers* de fichero. No cierra automáticamente los ficheros cuando termina el programa. Para asegurarnos de que queda registrada la longitud del fichero que se ha modificado en el directorio, se usa las funciones de cierre de ficheros 16 ó 62 antes de llamar a esta función.

Contrariamente a lo que indicaba el primer manual del DOS, este servicio no cambia la unidad de discos activa a A (ni hace ningún otro cambio), ni aun en el caso de las versiones del DOS que se indican.

Función 14 (E hex): Seleccionar la unidad de disco activa

La función 14 selecciona cuál va a ser la unidad de disco activa e informa del número de unidades instaladas. La unidad se selecciona

en el registro DL, 0 indica el A, y 1 el B, etc. El número de unidades se devuelve en AL. Una vez que el DOS conoce cuántas unidades de disco hay instaladas, se puede utilizar cualquier número para indicar cuál es la unidad de disco activa, desde el 0 hasta una menos que el número total de unidades.

Es necesario recordar unas cuantas cosas cuando se usa este servicio. Primero, no hay ningún hueco en los identificadores de las unidades de disco, ya que están numerados consecutivamente. Segundo, si, físicamente, sólo hay una unidad de disco, el DOS simulará una segunda unidad, la 1 (unidad B). Y tercero, el código ASCII de la letra identificadora de una unidad se halla sumando al código del carácter A el número asociado a la unidad. En el caso poco usual de que haya más de 26 discos, pueden aparecer algunas «letras» bastante peculiares.

La función 25 (19 hex) informa cuál es el número de la unidad activa, y las funciones 14 y 25 se pueden combinar para conocer el número de unidades sin necesidad de alterar la unidad activa. Esto se hace en lenguaje ensamblador de la forma siguiente:

```
MOV  AH,25          ; informa de la unidad actual
INT  33             ; llama a función
MOV  DL,AL          ; copia el número de la unidad activa
MOV  AH,14          ; fija la unidad activa
INT  33             ; llamada a función
```

En este momento, AL contendrá el número de unidades sin haber alterado a la unidad por defecto. Para convertir el número en la letra de la unidad mayor, puede añadir esta instrucción:

```
ADD  AL,'A'-1
```

Función 15 (F hex): Abrir un fichero

La función 15 abre un fichero de la manera tradicional del DOS, usando el bloque de control de fichero (*File Control Block*) (FCB). El FCB es un conjunto formado por 128 registros lógicos que contienen un conjunto de información suministrados por el programa sobre el fichero, que el DOS usa para localizar el fichero y los datos. (● Véase página 24 para encontrar más detalles sobre el FCB.) Los programas apuntan hacia el FCB, usando el par de registros DS:DX. El DOS intentará abrir el fichero, utilizando las especificaciones del FCB. Se informa del resultado obtenido, almacenando en AL un 255 para indicar que ha habido un fallo al abrir el fichero, y un cero para indicar el éxito. Recuerde que un fichero debe de existir para poder ser abierto (● véase la función 22 para encon-

trar un servicio creador de ficheros). Normalmente, esta función se usa cuando se abre un fichero para introducir datos, y la función 22 se usa cuando se abre un fichero para sacar datos.

Cuando la apertura tiene éxito, el DOS rellena muchos campos del FCB. Si se especifica la unidad de disco por defecto mediante la función 14, el DOS rellenará los datos de esa unidad. (Esto se hace de manera poco convencional; en este caso, el disco A se indica mediante un 1, en vez de por un 0, que es lo habitual en la mayoría de las operaciones realizadas con unidades de disco.) El DOS también rellena la fecha y la hora del fichero.

Algo poco usual ocurre con el campo del FCB que indica cuál es el tamaño de registro que se desea utilizar al leer y escribir el fichero. Normalmente se suministra este campo después de que el fichero se ha abierto. De todos modos, al abrir el fichero, el DOS inicializa por defecto este campo a un tamaño de registro de 128 bytes (80 hex). Se puede usar este tamaño de registro, o cambiarlo, dependiendo de la aplicación.

Muchos programas editores de texto usan el tamaño de registro de 128 bytes para realizar las operaciones de entrada/salida de forma eficiente. Irónicamente, el editor del DOS, Edlin, no lo hace así. Véase página 328 para profundizar más sobre el campo de tamaño del registro.

Función 16 (10 hex): Cerrar un fichero

La función 16 cierra un fichero utilizando un puntero que indica el FCB, colocado en el par de registros DS:DX. Si la operación tiene éxito, en AL se almacena un 0, y si la operación falla, en AL se almacena un 255 (FF hex).

Un fichero se debe cerrar después de cualquier operación con objeto de actualizar el directorio. El DOS hace un intento inteligente para detectar que estamos cerrando el mismo fichero en el mismo disco que lo abrimos, comparando la unidad especificada en el directorio con la que hay en el FCB abierto. Esto ofrece alguna protección contra el posible lío de la información que puede ocurrir cuando un usuario cambia los *diskettes* y escribe en el nuevo disco antes de que el fichero viejo se encuentre cerrado.

Función 17 (11 hex): Buscar un primer fichero

La función 17 comienza la operación de búsqueda de varios ficheros que coincidan con unas especificaciones dadas. El par de

registros DS:DX apunta hacia el FCB, que contiene el nombre del fichero que se busca. El uso pretendido para este servicio es manipular nombres de ficheros que incluyan los caracteres comodín ? y *. Este servicio comienza el proceso, y el siguiente servicio, la función 18, continúa la búsqueda de los ficheros subsiguientes.

AL señala el fracaso (255, FF hex) o el éxito (0). Si este servicio tiene éxito, el DOS introducirá el nombre del fichero en el campo «nombre de fichero» del FCB, dejando al fichero listo para abrir. El FCB también contendrá la información necesaria para continuar la búsqueda con el servicio dado por la función 18.

❑ **PRECAUCION:** *La información que necesita el servicio de búsqueda del siguiente será destruida por cualquier operación realizada en el fichero, si se usa el FCB. Si quiere buscar varios ficheros y además realizar entradas/salidas en ellos, debe guardar la información que hay en el FCB.*

Si el FCB tiene una extensión FCB (☛ véase página 324), entonces puede nombrar los atributos del fichero que desea buscar. Existe una lógica a seguir para realizar una búsqueda por atributos. Si se especifica cualquier combinación de bits de atributo del directorio, sistema u ocultos, la búsqueda actuará sobre ficheros normales, y también sobre cualquier fichero que tenga estos atributos. Si se especifica el atributo de etiqueta del volumen, esta función sólo buscará en el directorio algo que tenga ese atributo. Con las versiones del DOS previas a la 2.00 no se podía usar ni el directorio ni los atributos de etiquetas de volumen para realizar la operación de búsqueda del fichero. Los atributos de archivo y de sólo-lectura no se pueden utilizar como criterio de búsqueda en ninguna versión del DOS.

Este es un buen servicio para usar, incluso cuando no se pretende que los programas operen con más de un fichero. Incluso si el primer fichero encontrado es el único con el que se vaya a trabajar, al utilizar este servicio antes de abrir un fichero, le damos a los programas la flexibilidad de aceptar nombres de ficheros con caracteres comodín. Esta es la lógica usada para operar en varios ficheros.

```

10 ' Llame a la función 17: encontrar el primer fichero
20 WHILE FICHERO ENCONTRADO
30   SALVAR FCB PARA BUSCAR-SIGUIENTE
40 ' haga lo que sea necesario para salvar el campo FCB
50 RECUPERAR FCB PARA BUSCAR-SIGUIENTE
60 ' Llame a la función 18: buscar el siguiente fichero
70 WEND

```

Función 18 (12 hex): Buscar el siguiente fichero

La función 18 encuentra el siguiente de una serie de ficheros, siguiendo la preparación estructural realizada por la función 17.

☛ Véase la función 17 para conocer los detalles sobre la operación preparatoria y los códigos de retorno, así como una precaución especial.

Función 19 (13 hex): Borrar un fichero

La función 19 borra ficheros que se ajusten al FCB, señalado mediante el par de registros DS:DX. AL es 0 si la operación tiene éxito, y AL es 255 (FF hex) si la operación ha fallado, indicando que no hay ficheros en el directorio que se ajusten a los requisitos dados.

☛ En este capítulo podrá encontrar más detalles sobre el FCB.

Función 20 (14 hex): Leer un registro de un fichero secuencial

La función 20 se usa para leer el siguiente registro en un fichero secuencial. Antes de llamar a esta función, se debe de usar el par de registros DS:DX para apuntar hacia el FCB del fichero. El número de registros secuenciales se toma de los valores encontrados en el FCB. Después el fichero se lee, y si se realiza la operación con éxito o con éxito parcial, el dato se coloca en la zona de transferencia del disco. (☛ Véase la función 26.)

El DOS incrementa los campos de dirección del registro en el FCB después de cada lectura para facilitar el acceso secuencial del fichero. Se pueden cambiar los campos de dirección si se quiere saltar de un sitio a otro del fichero, pero esto no es aconsejable en ficheros secuenciales, ya que es mejor usar las funciones de entrada/salida aleatorias 33 y 34, cuando se necesite realizar acceso aleatorio a los registros de un fichero. (☛ Véase página 327 para tener información sobre la curiosa diferencia entre los métodos de cálculo de los números de los registros secuenciales y aleatorios.)

AL se usa para informar de los resultados de la lectura. El éxito completo se señala cuando AL es 0; si AL es 1, señala un fin de fichero, indicando que no se ha leído ningún dato; si AL es 2, señala que el dato puede haber sido leído, pero que el DTA (área de transferencia del disco) no tiene espacio suficiente para un registro completo, y si AL es 3, señala un fin de fichero con una lectura parcial de registro (el registro se ha rellenado con bytes a cero).

Función 21 (15 hex): Escribir un registro en un fichero secuencial

La función 21 permite escribir un registro en un fichero secuencial, y es el compañero del anterior servicio, función 20. Los registros DS:DX señalan al FCB, donde se encuentra la dirección del registro en que se va a almacenar. Después de leer la dirección, el DOS toma el dato del DTA (véase la función 26) y lo escribe en el disco.

Después de que el servicio ha terminado, AL contiene un código de información: AL=0 informa del éxito; AL=1 informa de que el disco está lleno; AL=2 informa de que en el segmento de transferencia del disco no hay espacio suficiente para escribir el registro. Advierta que el segmento de transferencia del disco, interno del DOS, debe de tener suficiente capacidad para acomodar los registros en el área de transferencia del disco.

Es importante señalar que el dato queda escrito desde un punto de vista lógico por este servicio, pero esto no quiere decir necesariamente que ha quedado escrito físicamente. El DOS pondrá los datos en un *buffer* hasta que tenga suficientes para llenar un sector de disco completo; llegado este momento, pasará a escribirlos. Hasta el momento en que un fichero se cierra, puede haber datos en el *buffer* del DOS, que desde el punto de vista de los programas hayan sido escritos, pero puede que no hayan sido transferidos todavía al disco. Puede ser un problema si un programa finaliza de manera anormal.

Función 22 (16 hex): Crear un fichero

La función 22 encuentra, o crea, un elemento de directorio para un fichero. El servicio, en primer lugar, busca en el directorio un fichero que se corresponda con el nombre, y si no encuentra ninguno, busca un sitio vacío. Entonces, dado un puntero a un FCB en DS:DX, el fichero se abre. Normalmente, esta función se usa para abrir un fichero de salida, y la función 15 se usa para abrir un fichero de entrada.

Si AL es 0, indica una operación con éxito. Si AL es 255 (FF hex), señala el fracaso, normalmente debido a una falta de espacio en el directorio. Cuando se abre un fichero con esta función, su longitud se ajusta a 0. Si se escribe información nueva en un fichero ya existente y después seguimos esta función con la función 16, que cierra el fichero, entonces los contenidos antiguos del fichero se pierden. Esto no ocurriría si se abre el fichero usando la función 15, que inicializa el campo del tamaño que hay en el fichero FCB con el encontrado en el directorio.

Función 23 (17 hex): Renombrar un fichero

La función 23 renombra ficheros en un FCB modificado señalado por DS:DX. Para realizar la operación de renombrar, el FCB se manipula de manera especial. Aunque la unidad y el nombre original del fichero se encuentren situados en sus posiciones habituales, el nuevo nombre de fichero se sitúa en la dirección relativa 16 del FCB, comenzando en el campo normalmente reservado para el tamaño del fichero (☛ véase página 327).

AL=0 indica un éxito completo, y AL=255 (FF hex) indica que o no se han encontrado ficheros que renombrar o que el nuevo nombre de fichero existía.

Si el nuevo nombre del fichero contiene caracteres comodín, como * o ?, se interpreta que significan lo mismo que en el nombre antiguo, y los caracteres en el nombre original que corresponden a las posiciones de los caracteres comodín no se cambian.

Función 24 (18 hex): Usado internamente por el DOS

La función 24 y las funciones desde la 29 hasta la 32 son usadas internamente por el DOS para sus propias aplicaciones internas. Aunque hay disponible alguna información limitada sobre estos servicios, es desaconsejable usar esta información, o difundirla. Cualquier función DOS que no sea anunciada públicamente como un servicio utilizable no puede ser de fiar, siendo posiblemente utilizadas en futuras revisiones del DOS.

Función 25 (19 hex): Informar sobre la unidad de disco activa

La función 25 informa de cuál es la unidad de discos activa, colocando la información en AL, el código numérico estándar es unidad A=0, unidad B=1, etc. (☛ Véase la función 14 para encontrar un ejemplo de cómo puede ser usada esta función en lenguaje ensamblador.)

Función 26 (1A hex): Fijar el área de transferencia del disco

La función 26 fija el área de transferencia del disco para que el DOS la utilice al realizar las entradas/salidas de ficheros. La situación del DTA se especifica mediante el par de registros DS:DX. Hay un DTA por defecto, de 128 bytes, disponible en el PSP con una dirección relativa de 80 hex.

Función 27 (1B hex): Obtener la información de la FAT de la unidad activa

La función 27 devuelve información clave sobre el disco situado en la unidad activa. La función 28 realiza el mismo servicio con cualquier unidad. La función 54, tratada en el siguiente capítulo, realiza casi un servicio idéntico.

Se obtiene la siguiente información: AL contiene el número de sectores por *cluster* (1 sector para *diskettes* de una sola cara, 2 para los de dos caras). CX contiene el tamaño en bytes de los sectores del disco (512 bytes en todos los formatos estándar PC). DX contiene el número total de *cluster* en el disco. Y DS:BX señala un byte en el área de trabajo del DOS que contiene el identificador de la FAT (véase página 135 para encontrar los detalles sobre la FAT). Antes de la versión 2.00 del DOS, el par de registros DS:BX señalaba a la FAT completa del disco (que estaba en memoria, y cuyo primer byte era el byte identificador). En las siguientes versiones, la FAT no está necesariamente situada por completo en un lugar, así que es más seguro suponer que DS:BX señala sólo al byte identificador.

Cuidado con una posibilidad que es peligrosa: esta función identificador hace que DS:BX señale al byte identificador de la FAT, que no está situado en el área del segmento de datos. Esto quiere decir que el valor normal de este registro (DS) queda modificado por esta función. Normalmente, esto desorganiza el funcionamiento de los programas, que confían en que el contenido del registro DS permanece estable. Para evitar este problema, es mejor preservar y restaurar el valor de DS tras este servicio. Aquí hay un ejemplo de cómo se debe hacer:

PUSH	DS	; salvar la dirección DS
MOV	AH,27	; pedir esta función
INT	33	; solicitar la llamada a funciones DOS
MOV	AH,[BX]	; coger el byte identificador FAT
POP	DS	; reemplazar la dirección DS

Este pequeño problema es un ejemplo de lo fácil que es caer en errores imprevistos cuando se diseñan programas. Si esta función utilizase el registro de segmento extra (ES) en vez del DS, entonces no habría ninguna trampa en la que podrían caer los programadores.

Función 28 (1C hex): Obtener la información de la FAT de cualquier unidad

La función 28 trabaja de la misma manera que la función 27, excepto en que puede informar de cualquier unidad de discos, no sólo

de la unidad activa. Antes de llamar a este servicio, ajuste DL al identificador especial de la unidad, donde 0=unidad activa, 1=unidad A, 2=unidad B, etc. (Advierta que no se usa el método de numeración convencional, que especifica a la unidad A como 0 y a la B como 1.)

Función 33 (21 hex): Leer un registro de un fichero aleatorio

La función 33 lee un registro situado en una localización cualquiera de un fichero. Después de situar el FCB del fichero mediante DS:DX, se debe de especificar el registro aleatorio que se quiere leer, almacenando su número asociado en el campo del registro aleatorio del FCB. Después de que se ha hecho esto, el dato se lee del DTA asociado al fichero aleatorio.

En AL se almacenan los mismos códigos que en una lectura secuencial: AL=0 indica una lectura con éxito; AL=1 indica el fin de fichero, sin más datos disponibles; AL=2 significa que el espacio que hay en el segmento de transferencia del disco es insuficiente, y AL=3 es un fin de fichero, con un registro parcial de datos disponible.

❑ *NOTA: El programa debe de almacenar continuamente en el campo del registro aleatorio el valor que se corresponde con cada registro aleatorio que lee. Si no se hace así, el DOS incrementa automáticamente los campos de forma secuencial, inicializándolos para que valgan cada vez uno más, haciendo después de una lectura aleatoria, lecturas secuenciales.*

Contraste esta función con la función 39, que puede leer más de un registro aleatorio, y con la función 20, que lee registros secuenciales. Véase la función 36 para encontrar más datos sobre la activación del campo de registro aleatorio.

Función 34 (22 hex): Escribir un registro en un fichero aleatorio.

La función 34 escribe un registro en cualquier posición de un fichero. Después de que el par de registros DS:DX señale al FCB del fichero, el programa debe dar un valor al campo de registro aleatorio del FCB el valor que se corresponde con el registro aleatorio en el que se quiere escribir. Después de que el campo de registro aleatorio se llena, el dato se escribe en el DTA asociado al fichero.

AL utiliza los mismos códigos que para una escritura secuencial; 1 significa que el disco está lleno; 2 indica que el espacio en el segmento de transferencia del disco es insuficiente.

□ *NOTA: El programa debe dar valores continuamente al campo de registro aleatorio para que se corresponda con cada registro aleatorio en el que se quiere escribir. Por el contrario, el DOS incrementa automáticamente los campos secuenciales, ajustándolos para acceder con facilidad al registro siguiente. A causa de esto, es a menudo más conveniente combinar los dos procesos siguiendo a una escritura aleatoria con escrituras secuenciales.*

● Contraste esta función con la función 40, que puede escribir más de un registro aleatorio y con la función 21, que permite escribir registros secuenciales. Véase la función 36 para encontrar más detalles sobre los valores del campo de registro aleatorio.

Función 35 (23 hex): Obtener el tamaño de un fichero

La función 35 informa sobre el tamaño de un fichero en términos del número de registros que tiene, DS:DX señala al FCB del fichero que se quiere conocer. Antes de llamar a la función, el FCB se debe dejar sin abrir y el campo de tamaño de registro del FCB rellenado. Si almacena en el registro del tamaño un 1, se darán los datos en bytes, que es la forma normal.

Si la operación tiene éxito, AL contiene un 0, y el tamaño del fichero aparece en el FCB. Si el fichero no se encuentra, AL contiene 255 (FF hex).

Función 36 (24 hex): Inicializar un registro de un fichero aleatorio

La función 36 da valores al campo de registro aleatorio en correspondencia con el bloque secuencial y con los campos de registro del FCB. Esto facilita el paso de operaciones de entrada/salida secuenciales a operaciones de entrada/salida aleatorias. Los registros DS:DX señalan al FCB de un fichero abierto.

Función 37 (25 hex): Fijar un vector de interrupción

La función 37 se usa para fijar un vector de interrupción. El par de registros DS:DX contiene el vector de una subrutina de control de

una interrupción, y AL contiene el número de la interrupción. El vector de interrupción de 4 bytes se sitúa en la tabla de vectores, y se llama mediante el número de la interrupción solicitada, que acompaña a la instrucción INT.

Cualquier programa que llegue a manejar con soltura los vectores de interrupción puede fijarlos con facilidad. Este servicio evita parte del trabajo sucio necesario para cambiar un vector de interrupción con seguridad. Para el caso de los vectores que señalan a los manipuladores de la interrupción, en vez de señalar a las tablas, la porción del segmento sería normalmente el código de segmento (CS), que debe ser transferido a DS. Advierta de nuevo la torpe elección de DS en lugar de ES, que entorpece el acceso a los datos de los programas.

☛ Para encaminar los contenidos del vector de interrupción, véase la función 53 (35 hex) en el siguiente capítulo.

Función 38 (26 hex): Crear un segmento de programa

La función 38 se usa para crear un nuevo segmento de programa cuando se desea preparar un subprograma, que se cargará de forma separada, para la ejecución. DX se usa para proporcionar la dirección del segmento del nuevo programa. El prefijo de segmento del programa en curso (PSP) se copia en los primeros 256 (100 hex) bytes de la nueva área de segmento, creando un nuevo PSP. Este nuevo PSP queda actualizado con la nueva información de la memoria y de los vectores de interrupción. Después de que este servicio ajusta el PSP, se pueden usar los servicios convencionales de entrada del DOS para leer un fichero de programa tipo COM, colocado en el área inmediatamente contigua al PSP. Con versiones avanzadas del DOS (2.00 y posteriores), el proceso completo de usar *overlays* es mucho más fácil.

☛ Véase el siguiente capítulo, particularmente la función 75, y para encontrar una explicación del prefijo del segmento de programa (PSP), véase página 292.

Función 39 (27 hex): Leer registros de un fichero aleatorio

A diferencia de la función 33, la función 39 lee uno o más registros, comenzando a partir de cualquier posición del fichero; DS:DX apunta al FCB del fichero que va a ser leído, y el número de registro aleatorio se toma de su FCB. CX contiene el número de registros deseados, que necesariamente deben ser mayor que 0.

Los códigos devueltos son los mismos que los de la función 33: AL=0 significa que la lectura ha tenido éxito; AL=1 indica fin de fichero (si los registros fueron leídos y el último registro está completo); AL=2 indica problemas en el segmento de transferencia del disco (a menudo problemas con la dirección relativa del segmento que supera FFFF hex, y esto no está permitido), y AL=3 indica el fin del fichero, cuando el último registro leído está incompleto, y se ha rellenado con ceros.

Sin importar el resultado, en CX se almacena el número de registros leídos, incluyendo cualquier registro parcial, y el campo de registro aleatorio del FCB queda almacenado con el número del siguiente registro aleatorio.

● Compare esta función con la función 33, que permite leer solamente un registro.

Función 40 (28 hex): Escribir registros en un fichero aleatorio

A diferencia de la función 34, la función 40 escribe uno o más registros, comenzando a partir de cualquier posición del fichero. DS:DX apunta al FCB del fichero que va a ser escrito y el número del registro aleatorio se elige de este FCB. CX contiene el número de registros deseado, y en este caso CX puede ser 0. Cuando es 0, se usa como una señal al DOS para que ajuste la longitud del fichero con la posición del registro aleatorio especificado. Esto hace más sencillo el que los programas puedan manipular ficheros aleatorios: si se han borrado registros al final de un fichero, este servicio permite a los programas truncar el fichero en este punto, ajustando la longitud y dejando libre de este modo más espacio en el disco.

Los códigos de retorno son los mismos que en la función 34: AL=0 indica una escritura con éxito, y AL=1 significa que no hay disponible más espacio en el disco. Sin importar el resultado, en CX queda almacenado el número de registros escritos.

● Contraste esta función con la función 34, que escribe un solo registro aleatorio.

Función 41 (29 hex): Analizar el nombre de un fichero

La función 41 analiza una línea de comando buscando un nombre de fichero con la forma UNIDAD:NOMBRE.EXT. Si se encuentra un fichero, crea un FCB. La función 41 es especialmente útil para procesar los parámetros del nombre de un fichero que se llama desde un programa. Los parámetros analizados están diseñados cuidadosa-

mente para que sea cómodo para un programa fijar una unidad, un nombre de fichero y una extensión por defecto, que pueda ser modificado por un comando.

El par de registros DS:SI señala hacia la cadena donde se encuentran las especificaciones de los ficheros. El par de registros ES:DI señala a la posición de memoria que se va a rellenar con un FCB. Los bits del 0 al 3 de AL controlan cómo será analizado el fichero.

Si el bit 0 es 1, la función examina más allá de los separadores (por ejemplo, espacios en blanco) para encontrar las especificaciones del fichero. Si el bit 0 es 0, esta operación no se realiza y se supone que las especificaciones del fichero están a partir de la posición del primer byte de la línea de comando.

Si el bit 1 es 1, entonces el byte de la unidad de disco en el FCB sólo se llenará si aparece en las especificaciones del fichero que está siendo examinado. Esto permite al FCB tener su propia unidad por defecto (que puede ser anulada), en vez de usar la unidad por defecto del DOS.

Si el bit 2 es 1, el nombre del fichero que aparece en el FCB sólo será cambiado si se encuentra un nombre de fichero válido en las especificaciones del fichero. Esto permite a los programas colocar un nombre de fichero por defecto, que puede ser anulado por el comando.

Si el bit 3 es 1, la extensión del nombre de fichero en el FCB sólo será cambiada si se encuentra un nombre de fichero válido.

Cuando se realiza el análisis, la puntuación que aparece precediendo dentro de y al final de las especificaciones del fichero se reconoce. Si se encuentra el carácter comodín *, se traduce al formato más elaborado de ?.

Como es usual, AL informa del resultado. AL = 0 señala el éxito con un único nombre de fichero; AL = 1 señala el éxito con caracteres globales (* o ?), lo que alerta sobre la necesidad de realizar el proceso de encontrar-el-primero,-encontrar-el-siguiente (véanse las funciones 17 y 18); AL = 255 (FF hex) señala el fallo (que generalmente significa algún problema en las especificaciones del fichero).

Para facilitar el proceso repetido, DS:SI (en realidad sólo SI) será actualizado para que apunte más allá de la entrada analizada. Si el análisis no tiene éxito, el segundo byte del FCB (ES:DI+1) estará en blanco.

Como los servicios del DOS universal realizados al viejo estilo, esta función no puede manipular nombres de ruta, lo que limita —pero no elimina— su utilidad.

Función 42 (2A hex): Obtener la fecha

La función 42 informa de la fecha actual. La fecha se sitúa en CX y DX. DH contiene el número del mes (del 1 al 12); DL contiene el día del mes (de 1 a 28, 29, 30 ó 31, según sea el apropiado), y CX contiene el año (desde 1980 hasta 2099).

El día de la semana se comunica mediante este servicio, situando un valor comprendido desde el 0 hasta el 6 en el registro AL, que significa desde el domingo hasta el sábado. Esta característica del día de la semana es, por así decirlo, una huérfana. Ha aparecido en el DOS desde el lanzamiento 1.10, pero no fue ni mencionada hasta la versión 2.00. En los dos manuales, 2.00 y 2.10, está incorrectamente descrita como parte de la función de obtener la hora, y no como parte de la función de obtener la fecha. Comenzando con el DOS 3.0, el manual la explica como debe ser. ☛ Vaya al ejemplo que aparece en la página 329 para ver cómo puede usar este servicio.

Función 43 (2B hex): Poner la fecha

La función 43 da valor al registro del DOS que contiene la fecha, de la misma forma que comunica la fecha la función 42. La fecha se escribe en CX y DX. DH contiene el número de mes (del 1 al 12); DL el día del mes (del 1 al 28, 29, 30 ó 31) según sea apropiado; CX contiene el año (desde 1980 hasta 2099), y AL el día de la semana (del 0 al 6).

☛ Véase la función 42 por encontrar más explicaciones. En el ejemplo de la página 329 podrá ver cómo usar este servicio.

Función 44 (2C hex): Obtener la hora

La función 44 informa de la hora del día. La hora se calcula a partir del conteo de impulsos de reloj (*clicks*) de la ROM-BIOS (☛ Véase página 247.) El DOS responde a la señal de límite de medianoche de la ROM-BIOS y actualiza a las 24 horas.

La cuenta de *clicks* se convierte en una hora significativa, y se sitúa en los registros CX y DX. CH contiene la hora (de 0 a 23, a modo de un reloj de 24 horas); CL, los minutos (de 0 a 59); DH, los segundos (de 0 a 59), y DL, las centésimas de segundo (de 0 a 99).

El periodo real de los *clicks* de reloj es aproximadamente .054 segundos, así que no se puede comunicar el tiempo exactamente en centésimas de segundo y ningún suceso puede ser medido más exactamente que en $\frac{1}{20}$ segundos. Sin embargo, he confirmado que el algoritmo usado por el DOS para calcular las centésimas produce una

distribución uniforme sobre los 100 valores. Por ello, es razonable usar el valor de las centésimas como semilla para la generación de números pseudoaleatorios.

Contrariamente a lo que aparece en el «Manual Técnico de Referencia del DOS», no se comunica el día de la semana en esta función. El día de la semana se comunica con la función 42, que es la función que permite obtener la fecha, como es lógico para cualquier persona sensata.

Función 45 (2D hex): Poner la hora

La función 45 da valores a la hora del día. La hora se especifica en los registros CX y DX. CH contiene la hora (de 0 a 23, en un reloj de tipo 24 horas); CL contiene los minutos (de 0 a 59); DH contiene los segundos (de 0 a 59), DL contiene las centésimas de segundo (de 0 a 99).

Función 46 (2E hex): Activar la verificación de la escritura en disco

La función 46 controla la verificación de las operaciones de escritura en el disco. Cuando la verificación está activada, cada escritura en el disco viene seguida por una lectura de verificación. Esta acción no compara los datos escritos, sino que comprueba el CRC, que es una compleja comprobación de la paridad de la grabación realizada.

Cuando se use este servicio, en DL debe haber un 0, lo que sugiere que puede haber algunas variaciones no anunciadas de esta función. En AL se almacena un 0 o un 1, lo que permite desactivar o activar la verificación.

Con las versiones del DOS 2.00 y posteriores, la función ampliada 84 (54 hex) se puede usar para obtener información sobre el estado del conmutador de la verificación (■ véase página 359).

EL BLOQUE DE CONTROL DE FICHERO

Las funciones tradicionales que trabajan con ficheros de disco lo hacen a través del bloque de control de fichero (FCB), que es un área de 44 bytes que contiene información descriptiva sobre los ficheros que se están usando. Como se puede ver en muchas de las llamadas a función, no sólo se tiene acceso al FCB, sino que se tiene un completo control sobre él. Encontraremos que las nuevas llamadas a función del DOS, introducidas con la versión 2.00 (■ y tratadas en el

capítulo 17), mantienen oculta la mayor parte de la información de control sobre un fichero (u otro dispositivo). En vez de esto, trabajaremos solamente con un simple número de identificación de fichero, llamado número de fichero (*file handle*), y el DOS hará el resto. Pero la discusión sobre los números de fichero se verá en el siguiente capítulo. Aquí se cubrirá el formato y el uso del FCB.

Hay dos partes principales en el FCB: el FCB en sí, que es un área de 37 bytes, y la extensión del FCB, que es un añadido casi opcional de 7 bytes que precede al FCB. La mayor parte del FCB almacena información de control sobre los ficheros, incluyendo el nombre del fichero y la especificación de la unidad de disco, al igual que el tamaño y el número de registros. La extensión de 7 bytes indica que el fichero tiene atributos especiales (■ véase página 130 para encontrar más información sobre los atributos de fichero).

La situación con la extensión del FCB es más que peculiar. Solamente se usa la extensión cuando se está trabajando con ficheros que tienen atributos no convencionales, como los ocultos y los ficheros del sistema. Parece lógico suponer que sólo se necesite esta extensión en estas circunstancias, y, si no se está usando ficheros especiales, no es necesario reservar el espacio de almacenamiento de 7 bytes. En la práctica, es imprescindible reservar el espacio para la extensión del FCB, porque la presencia de la extensión se señala mediante el valor 255 (FFF hex) en su primer byte. Para colmo de males, si no está usando la extensión del FCB, tiene que estar seguro de que está marcada como no usada, lo que quiere decir que el área de memoria de 7 bytes está realmente en uso, aunque hay que decirle al DOS que no se necesita. Esta es la clase de diseños toscos que vuelve locos a los programadores responsables, y hace que todos, incluso los diseñadores del DOS, anhelan las funciones ampliadas expuestas en el capítulo 17.

El FCB se direcciona desde el principio de la parte principal, y todos los campos dentro del FCB se tratan en términos de su dirección relativa a partir del comienzo de la parte principal. Esto sitúa a la ampliación del FCB en una dirección relativa negativa de -7. Seguiré las convenciones del DOS y describiré los diferentes campos en estos términos.

Los campos mostrados en la figura 16-3, que son utilizados y controlados por el DOS, no se deben alterar por nuestros programas. Los demás campos sí los podemos modificar. Los marcados Us-1 se deben ajustar antes de que se abra el fichero, y los marcados Us-2, antes de que se lea o escriba en el fichero. Veremos el uso y codificación de las partes del FCB en el orden en que aparecen en la tabla.

Dirección relativa (dec)	Tamaño (bytes)	Controlado por	Descripción
-7	1	Us-1	Señal de ampliación activa de: FF = SI, en cualquier otro caso indica no activada
-6	5	Us-1	Nominalmente no usada: debe estar ajustada a ceros
-1	1	Us-1	Atributo del fichero, cuando la extensión está activa
0	1	Us-1	Número especial de unidad
1	8	Us-1	Nombre del fichero o nombre del dispositivo
9	3	Us-1	Ampliación del nombre del fichero
12	2	Us-2	Número del bloque actual
14	2	Us-2	Tamaño del registro
16	4	DOS	Tamaño del fichero en bytes
20	2	DOS	Fecha del fichero (codificada en bits, como en el directorio del fichero)
22	10	DOS	Area de trabajo de control DOS
32	1	Us-2	Número de registro actual (byte con «sig-»
33	4	Us-2	Número de registro aleatorio

Figura 16-3. Descripción de los campos del bloque de control de fichero

Direcciones relativas -7 y -6. La señal de extensión debe de tener el valor de 255 (FF hex) si se quiere utilizar la extensión del FCB, o cualquier otro valor si no se desea utilizar. La literatura del DOS indica que el siguiente campo de 5 bytes debe contener ceros, aunque no es importante este hecho.

Dirección relativa -1. El campo de atributos del fichero debe contener los atributos especiales del fichero que se va a abrir. Esto no es necesario para los ficheros ordinarios, que tienen un atributo de 0. Los atributos que se deben especificar son los atributos de oculto, del sistema o de directorio. El atributo de la etiqueta es irrelevante, y la especificación del atributo no es necesaria para acceder a un fichero sólo de lectura. ■ Véase página 130 para leer más sobre codificación de atributos.

Dirección relativa 0. El número especial de unidad es el byte con dirección relativa 0 y se emplea para indicar la unidad con la que se desea trabajar. No es igual al número identificador estándar de unidad, en el que el 0 indica el controlador A, el 1 indica el controlador B, etc. En vez de esto, el número utiliza un formato más flexible, donde el 1 indica la unidad A, el 2 la unidad B, y así sucesivamente. La flexibilidad proviene de usar el 0 para indicar la unidad activa, cualquiera que sea. Antes de que un fichero se abra, se

puede especificar la unidad deseada, o indicar la unidad por defecto con un 0. Cuando el fichero se abre, el DOS cambia el 0 por el número específico de la unidad, que se puede examinar mediante la función 25 (19 hex). Cuando se trabaje con este campo, hay que tener cuidado en no confundir los valores especiales de la unidad con los números convencionales, que son una unidad menos que los números usados aquí.

Direcciones relativas 1 y 9. Los dos campos situados en los bytes cuyas direcciones relativas son 1 y 9 dan el nombre de fichero y su extensión. Siguiendo las convenciones estándar del DOS, estos campos están justificados a la izquierda, y se completa a la derecha con caracteres blancos (CHR\$(32), 20 hex). Siguiendo con las convenciones del DOS, se pueden usar las letras mayúsculas o las minúsculas. Si el nombre del fichero es un nombre de dispositivo que el DOS reconoce, tal como CON:, AUX:, COM1:, COM2:, LPT1:, LPT2:, PRN:, o NUL:, el DOS usará ese dispositivo en vez de un fichero de disco.

❑ *NOTA: Este es un buen momento para indicar que el mecanismo del FCB no está preparado para trabajar con nombres de ruta. Siempre que use el FCB, se hace referencia al directorio activo en cualquier unidad de disco. Para poder realizar un uso flexible de rutas y subdirectorios, véanse las nuevas funciones ampliadas en el capítulo 17.*

Direcciones relativas 12 y 32. Para realizar operaciones con ficheros secuenciales se emplean los campos del bloque activo y de registro activos. Se emplean para tener en cuenta en qué zona del fichero estamos actuando. El uso de estos campos es bastante singular. En vez de emplear un único número de registro, el número de registro está dividido en una parte alta y otra baja, denominado números de bloque y de registro. Para hacer las cosas aún más extrañas, los bloques constan de 128 registros, en vez de los 256 registros que un número de 1 byte permitiría representar. Esto es por lo que aparece descrito el número de registro como un byte con «signo» en la figura 16-3. Realmente no es de signo, ya que los valores negativos no están permitidos; significa simplemente una advertencia de que los valores permitidos cubren desde el 0 hasta el 127, y no desde el 0 hasta el 255.

El primer registro de un fichero es el registro 0 del bloque 0. Este formato está diseñado así para permitir un cálculo rápido de la posición de cada registro secuencial. Mediante este sistema de localización de registros, el DOS realmente realiza operaciones con ficheros secuenciales como una variación de las operaciones con

ficheros aleatorios. Se debe rellenar estos dos campos antes de realizar la primera operación en un fichero secuencial, tanto de lectura como de escritura. El DOS automáticamente incrementará el campo con cada operación realizada posteriormente. Está permitido saltar registros en un fichero secuencial modificando los campos —una táctica que es buena para un fichero de entrada, y peligrosa para un fichero de salida.

Dirección relativa 14. El campo de tamaño del registro comienza en el byte de dirección relativa 14 y da el tamaño en bytes de los registros lógicos del fichero. Cuando un programa le pide al DOS que lea o escriba un registro, el tamaño lógico del registro es el número de bytes transferidos entre los *buffers* del disco y el área de datos del programa. Este valor no está relacionado con el fichero tal como lo ve el DOS o con el fichero real almacenado en el disco, sino que indica cómo ve el programa al fichero cada vez.

El mismo fichero de datos se puede manejar con varios tamaños de registro. A no ser que el fichero esté realmente constituido por registros de longitud fija, normalmente se trata al fichero como si tuviera el registro de longitud de 1 byte, a fin de conservar espacio. Es costumbre en muchos programas de tratamiento de textos el usar una longitud de registro de 128 bytes para ficheros ASCII. Esto reduce el número de llamadas a funciones del DOS que se necesitan para trabajar con el fichero a menos de una centésima de las que serían necesarias con una longitud de registro de 1 byte. Naturalmente hay que pagar un pequeño precio por esta trampa: el tamaño del fichero almacenado por el DOS puede ser demasiado grande, y estos programas deben hacer un pequeño esfuerzo extra para mantener los datos en orden. Cuando se abre un fichero mediante la función 15, el DOS rellena la longitud del registro a 128 bytes, por defecto. Si se quiere otra longitud, como, por ejemplo, para realizar operaciones con 1 único byte, debemos cambiarlo después de que el fichero esté abierto.

Dirección relativa 16. El campo de tamaño de fichero, cuya dirección relativa es 16, indica el tamaño del fichero en bytes. Se toma el valor del directorio del fichero y se sitúa en el FCB cuando se abre el fichero. En el caso de un fichero de salida, este campo lo va cambiando el DOS conforme crece el fichero, y cuando se cierra, este valor se coloca en el directorio. Si se cambia este campo en el último momento, se puede conseguir algún control de última hora sobre el tamaño del fichero de salida, pero sea cuidadoso si lo intenta. Si queremos leer un subdirectorio como si fuese un fichero (lo que requiere el uso de la ampliación del FCB para indicar el atributo de tipo directorio), este campo será 0 cuando se abra el fichero, porque,

en el directorio, un subdirectorio aparece con la indicación de longitud cero. Para leer con éxito el subdirectorio es necesario llenar este campo con algún valor alto arbitrario. También es necesario recordar algo más: cuando se usa la operación de renombrar (función 23), el nuevo nombre del fichero se coloca en la dirección relativa 16, que es exactamente el lugar donde está situado el campo de tamaño del fichero.

Dirección relativa 20. La fecha del fichero está codificada en un campo de 2 bytes, de la misma manera que aparecen en el directorio del fichero, usando el formato MM/DD/AA. Este campo lo rellena el DOS cuando se abre el fichero con la información tomada del directorio. (● Véase página 132 para encontrar más información sobre la codificación de la fecha.)

Dirección relativa 33. El campo de registro aleatorio se usa durante las operaciones de lectura y escritura mediante acceso aleatorio, del mismo modo que los números de bloque y de registro se usan durante operaciones secuenciales. Este campo está formado por 4 bytes, un entero de 32 bits, los cuales se pueden descomponer fácilmente en bytes o en palabras. Los registros están numerados desde el 0, lo que hace fácil calcular la dirección relativa de cualquier registro, multiplicando el número del registro aleatorio por la longitud del registro. Debemos rellenar este campo antes de realizar cualquier operación con ficheros aleatorios. El DOS lo deja inalterado.

UN EJEMPLO

Para realizar un ejemplo en lenguaje ensamblador en esta sección, he elegido algo bastante interesante y astuto. Es una rutina que he desarrollado para usar dentro de los programas *Norton Utilities*, así que el ejemplo es un programa real, modificado solamente para intensificar los comentarios.

El propósito de esta rutina es calcular el día de la semana de cualquier día que esté dentro del rango de trabajo del DOS, desde el martes 1 de enero de 1980 hasta el jueves 31 de diciembre de 2099. En algunos casos resulta útil que algunos programas sean capaces de calcular el día de la semana de cualquier fecha por la que se pueda preguntar. Por ejemplo, el DOS lleva la cuenta de la fecha y de la hora en la que cada fichero fue cambiado por última vez. Debido a que se usa muy a menudo esta información para encontrar cuándo trabajamos por última vez con un fichero, puede ser conveniente conocer también el día de la semana. De hecho, el día de la semana es a menudo más significativo que la fecha.

Aunque hay varios algoritmos interesantes e inteligentes ya publicados para calcular el día de la semana, el trabajo de escribir un programa de día de la semana es generalmente bastante tedioso. Parece como si tuviéramos pocas veces a mano estos algoritmos publicados cuando los necesitamos, y si los tenemos, suelen estar apresados en un lenguaje que no es adecuado para nuestro propósito. Comenzando con la versión 2.00, el DOS incorporó un cálculo del día de la semana, que puede librarnos de la tarea de escribir el nuestro. El programa del DOS sólo indica el día actual de la semana, pero esto no es obstáculo. Se puede cambiar temporalmente la fecha del DOS por la fecha que interesa, y entonces tener la información DOS del día de la semana. Esto es lo que hace la siguiente rutina en lenguaje ensamblador.

Además, siendo un poco astuto, esta subrutina es interesante porque ilustra el uso de tres llamadas a funciones del DOS, que operan juntas para obtener un resultado. Esto también ilustra perfectamente las complicaciones menores que están relacionadas con el guardar y recuperar cosas del *stack*. Como veremos aquí, el uso del *stack* tiene que estar cuidadosamente orquestado para que los valores no se coloquen en el sitio que no les corresponde.

Esta subrutina particular, llamada DIASEMANA, se ajusta a la forma necesitada para que la utilice con el compilador Lattice/Microsoft C. La rutina se llama con tres variables, como parámetros de entrada, que indican el mes, el día y el año que nos interesa. La rutina devuelve el día de la semana en la forma de un entero comprendido entre 0 y 6 (desde el domingo hasta el sábado). Lo que se ajusta perfectamente a la convención del lenguaje C, proporcionando un índice para elegir entre un conjunto de cadenas de caracteres que dan los nombres de los días. Se puede usar esta subrutina de esta forma:

```
NOMBRES_DIA (DIASEMANA (DIA,MES,AÑO))
```

Es importante darse cuenta de que esta rutina trabaja a ciegas con la fecha sin comprobar si la fecha es válida o si está comprendida dentro del rango de las fechas aceptadas por el DOS. También hay que advertir que esta rutina requiere la versión del DOS 2.00, o superior. Aquí está nuestra subrutina:

```
PGROUP  GROUP  PROG
PUBLIC  DIASEMANA
PROG    SEGMENT BYTE PUBLIC 'PROG'
        ASSUME  CS:DIASEMANA
DIASEMANA  PROC    NEAR
```

```

; Estrategia: obtener y salvar la fecha antigua
;
;      fijar la nueva fecha
;      obtener el día de la semana
;      reponer la fecha antigua
PUSH    BP
MOV     BP,SP

MOV     AH,2AH      ; obtener la fecha (para salvar)
INT     21H         ; llamada a función DOS

PUSH    CX          ; salvar la fecha antigua...
PUSH    DX          ; ...en el stack

MOV     CX,[BP+8]   ; year
MOV     DH,[BP+6]   ; mes
MOV     DL,[BP+4]   ; día
MOV     AH,2BH      ; ajustar la fecha temporalmente
INT     21H         ; llamada a función DOS
MOV     AH,2AH      ; obtener la fecha para obtener el día de la semana
INT     21H         ; llamada a función DOS
POP     DX          ; restaurar la fecha antigua...
POP     CX          ; ...desde el stack
PUSH    AX          ; salvar en el stack el día de la semana
MOV     AH,2BH      ; reajustar la fecha antigua
INT     21H         ; llamada a función DOS
POP     AX          ; recuperar del stack el día de la semana
MOV     AH,0        ; limpiar la parte alta
POP     BP
RET
DIASEMANA    ENDP
PROG         ENDS
END

```


Las funciones nuevas del DOS

Mejoras realizadas en el DOS-2 334

Mejoras de las funciones ampliadas del DOS 334

Los controladores de dispositivos instalables 337

Resumen de la ampliación de funciones del DOS 338

Función 47 (2F hex): Obtener la dirección del DTA 340

Función 48 (30 hex): Obtener el número de versión del DOS 340

Función 49 (31 hex): KEEP avanzado, Terminar permaneciendo residente 340

Función 51 (33 hex): Obtener/modificar el estado de Ctrl-Break 341

Función 53 (35 hex): Obtener el vector de interrupción 341

Función 54 (36 hex): Obtener el espacio libre del disco 342

Función 56 (38 hex): Obtener información dependiente del país 342

Función 57 (39 hex): MKDIR. Crear un directorio 346

Función 58 (3A hex): RMDIR. Borrar un directorio 346

Función 59 (3B hex): CHDIR. Cambiar el directorio 346

Función 60 (3C hex): CREAT. Crear un fichero 347

Función 61 (3D hex): Abrir un fichero 347

Función 62 (3E hex): Cerrar el número del fichero 349

Función 63 (3F hex): Leer desde un fichero, o dispositivo 349

Función 64 (40 hex): Escribir en fichero, o dispositivo 349

Función 65 (41 hex): Borrar fichero 350

Función 66 (42 hex): Mover el puntero del fichero 350

Función 67 (43 hex): CHMOD. Obtener/modificar los atributos del fichero 351

Función 68 (44 hex): IOCTL. Control de entrada/salida para los dispositivos 351

Función 69 (45 hex): DUP. Duplicar el número del fichero 353

Función 70 (46 hex): CDUP. Forzar la duplicación del número de un fichero 354

Función 71 (47 hex): Obtener el directorio activo 354

Función 72 (48 hex): Asignar memoria 355

Función 73 (49 hex): Liberar la memoria asignada 355

Función 74 (4A hex): SETBLOCK. Modificar el bloque de la memoria asignada 355

Función 75 (4B hex): EXEC. Cargar/ejecutar un programa 356

Función 76 (4C hex): Terminar un proceso 358

Función 77 (4D hex): Obtener el código de retorno de un subprograma 358

Función 78 (4E hex): FIND FIRST. Comenzar la búsqueda de los ficheros 358

Función 79 (4F hex): Continuar la búsqueda del fichero 359

Función 84 (54 hex): Obtener el estado de verificación 359

Función 86 (56 hex): Renombrar un fichero 360

Función 87 (57 hex): Obtener/modificar la fecha y la hora de un fichero 360

Funciones añadidas por el DOS 3.00 361

Función 89 (59 hex): Obtener el código de error ampliado 361

Función 90 (5A hex): Crear un fichero transitorio 364

Función 91 (5B hex): Crear un fichero nuevo 364

Función 92 (5C hex): Bloquear/desbloquear el acceso a un fichero 365

Función 98 (62 hex): Obtener la dirección del PSP 365

Habiéndome centrado en las funciones «tradicionales» del DOS en el último capítulo, en éste trataré de las funciones del DOS que son nuevas a las versiones avanzadas del DOS, desde la versión 2.00 en adelante. De acuerdo con la terminología DOS, los servicios «antiguos» o «universales», tratados en el capítulo 16, son las funciones tradicionales, y los servicios «nuevos» de los que trata este capítulo son las funciones ampliadas.

Las funciones ampliadas del DOS 2.10 son idénticas a las de la versión 2.00. La versión DOS 3.00 introdujo algunos cambios en unas pocas de las funciones existentes, y también añadió seis funciones nuevas. En este capítulo se encontrará cada función DOS-2 descrita con detalles, describiendo las modificaciones del DOS 3.00 cuando quiera que ocurran. Las nuevas funciones ampliadas del DOS 3.00 se describen con detalle al final del capítulo.

■ Véase capítulo 15 para recordar las interrupciones del DOS, que son los servicios y detalles afines que no están clasificados como funciones. Después, véase en capítulo 16 las llamadas a una función del DOS tradicionales, que pueden ser usadas con cualquier versión del DOS.

MEJORAS REALIZADAS EN EL DOS-2

Con casi cada nueva versión del DOS hay cambios en la manera en que el DOS opera y en el número de servicios que proporciona a los programadores. La introducción del DOS 2.00 trajo consigo los cambios más dramáticos: añadió 33 servicios nuevos a los 42 existentes; cambió la forma de acceder a la información de un fichero como resultado de estos servicios nuevos e hizo posible adaptar el DOS para trabajar con casi cualquier dispositivo de *hardware* mediante el uso de programas llamados controladores de dispositivos instalables. Antes de exponer con detalle las funciones ampliadas, comentaré brevemente cómo algunas de estas realizaciones afectan a nuestros hábitos de programación.

Mejoras de las funciones ampliadas del DOS

Muchos de los servicios ampliados introducidos con el DOS 2.00 y el DOS 3.00 tienen tres características importantes que afectan directamente al modo en que usamos los servicios. Primero, la mayoría de los servicios devuelve un juego de códigos de error estándar en el registro AX; segundo, todas las funciones que usan una entrada de cadena requieren un formato de cadena especial,

conocido como formato ASCIIZ (una cadena seguida por un byte de ceros), y tercero, muchas de estas funciones ampliadas DOS usan un número de 16 bits, llamado título o número del fichero (*file handle*), en vez de un FCB, para llevar la cuenta de los ficheros y dispositivos de entrada/salida con los que se comunica un programa. Trataré cada una de estas realizaciones por turno en las siguientes páginas.

Los códigos de error estándar

Muchas de las funciones nuevas ampliadas del DOS devuelven un juego estándar de códigos binarios de error, llamados códigos de retorno (*return codes*), cuando el servicio ha terminado de operar. Están anotados en la figura 17-1. Generalmente, los códigos de retorno señalan que un servicio ha fallado por alguna razón. Pueden ser usados por nuestros programas para determinar la naturaleza del problema y la acción adecuada que debe ser tomada.

Los códigos de retorno estándar se sitúan en el registro AX (aunque podría usarse sólo medio registro, ya que todos los códigos tienen valores inferiores a 255). El indicador de acarreo (CF) se usa para señalar un error (al menos, esto es para lo que está, en teoría; he descubierto al experimentar con estos servicios que no se puede depender del CF para señalar condiciones que usted o yo podemos pensar que son llamadas de una señal de error). Afortunadamente, los códigos de error estándar son de fiar, y se puede contar con los valores mostrados en la figura 17-1 para detectar los errores. El que

Código de error			Código de error		
Dec	Hex	Significado	Dec	Hex	Significado
1	1	Número de función no válido	10	A	Entorno no válido (véase el comando SET del DOS)
2	2	Fichero no encontrado	11	B	Formato no válido
3	3	Ruta no encontrada	12	C	Código de acceso no válido
4	4	No hay número disponible, todos usados	13	D	Dato no válido
5	5	Acceso denegado	14	E	No usado
6	6	Número no válido	15	F	Especificación de unidad no válida
7	7	Bloques de control de memoria destruidos	16	10	Intento de suprimir el directorio activo
8	8	Memoria insuficiente	17	11	Dispositivo diferente
9	9	Dirección de bloque de memoria no válida	18	12	No hay más ficheros que encontrar

Figura 17-1. Los códigos de error estándar situados en el registro AX tras una función fallida, y sus significados

se pueda o no usar CF como señal es otra cuestión. Yo aconsejaría que se probara la función experimentalmente para ver lo que ocurre con la bandera CF, o que se ignorara completamente la bandera y que se probaran los códigos específicos que pueden presentarse en el registro AX. En las descripciones de las funciones que suministran códigos de retorno encontrará una lista de los códigos que suelen ser devueltos por un error en la función.

❑ *NOTA: La versión DOS 3.00 no sólo ofrece los códigos de error estándar, sino también otros códigos de error mediante la función 89. Véase página 361.*

La cadena ASCIIZ

Siempre que cualquier función ampliada precise cadenas de longitud variable, tal como un nombre de ruta, se utiliza un formato conocido como cadena ASCIIZ. Una cadena ASCIIZ está compuesta por una serie de caracteres convencionales ASCII, seguida por un byte cero, que marca el final de la cadena. Una típica cadena para un nombre de ruta puede tener una forma parecida a ésta:

```
A:\DIRECTORIO1\DIRECTORIO2\NOMBREFICHERO.EXT(BYTE DE CEROS)
```

La especificación de unidad es opcional y las barras son aceptadas como delimitadores de ruta.

El formato de la cadena ASCIIZ es comúnmente usado tanto por el sistema operativo UNIX como por el lenguaje de programación C; es uno de los muchos elementos nuevos con un sabor característico de C/UNIX introducidos con las versiones DOS-2.

Número de fichero (*file handles*)

Las nuevas funciones ampliadas del DOS tratan los ficheros de una manera más interna que los servicios del DOS tradicionales. Los servicios tradicionales hacen uso de unos bloques de control de ficheros (FCB), a los que se tiene acceso y control casi completo. Los servicios ampliados mantienen la información sobre un fichero (o dispositivo) de una forma no accesible al usuario.

En los servicios tradicionales se indica el fichero con el que se está trabajando, señalando un FCB, y se utilizan los servicios para realizar las tareas de manejo de fichero, tales como localizar, señalar los datos y determinar el tamaño del fichero. Por el contrario, en los servicios ampliados se trabaja con el número del fichero, que no es más que un número de 16 bits devuelto a través del registro AX. Este número identifica el fichero, o el dispositivo, con el que se está trabajando y

Número	Uso	Apunte por defecto
0	Entrada estándar (normalmente del teclado)	CON:
1	Salida estándar (normalmente de la pantalla)	CON:
2	Error de salida estándar (normalmente de la pantalla)	CON:
3	Dispositivo auxiliar de salida (dispositivo AUX:)	AUX:
4	Impresora estándar (dispositivo LPT1: o PRN:)	PRN:

Figura 17-2. Los cinco números estándar disponibles por cada fichero

automáticamente realiza la mayor parte de las operaciones de manejo de ficheros.

El DOS mantiene un control completo sobre los números de fichero, y los da siempre que se crea, o se abre, un fichero. Hay cinco números estándar, numerados desde el 0 hasta el 4, que están siempre disponibles para cada programa. (Véase la figura 17-2.) El DOS suministra otros números de número mayores cuando se necesitan.

Las funciones tradicionales permiten que sean abiertos al mismo tiempo hasta 99 ficheros (y los FCB correspondientes) (**AT** en el AT se permiten 255). De todos modos, las funciones ampliadas, operando bajo versiones DOS-2, dan un máximo de 20 números de fichero a cualquier programa, limitando así el número de ficheros que un programa puede abrir realmente. El DOS 3.00 proporciona, a petición, más números, pero toma por defecto el límite de 20 ficheros del DOS-2. Esta limitación en las funciones ampliadas afecta solamente a los programas muy complejos, que requieren gran cantidad de ficheros abiertos al mismo tiempo.

Los controladores de dispositivos instalables

Otras dos características adicionales que fueron introducidas con el DOS 2.00 son también dignas de atención: los controladores de dispositivos instalables, y el controlador ANSI (generalmente llamado ANSI.SYS), un controlador de dispositivos que es estándar en muchas adaptaciones del DOS.

Los controladores de dispositivos instalables son programas que se crean para unir dispositivos *hardware* al sistema de ordenador, sin tener que reescribir la BIOS. Un controlador de dispositivo puede ser una rutina que apoya un nuevo *hardware*, tal como un *joystick* o un ratón, o puede ser una rutina que modifique la operación del

hardware estándar, para que pueda realizar tareas no disponibles mediante los servicios del DOS. (El controlador ANSI es un ejemplo del último tipo; permite modificar la salida de la pantalla, o la entrada, del teclado sin usar las rutinas de servicio del DOS.) Desde que los controladores de dispositivos son parte del DOS, no es necesario descender a la BIOS ni al nivel de *hardware* para instalar nuevos dispositivos, una importante característica para los programas que operan en entornos con ventanas, o multitareas.

Los controladores de dispositivos no están relacionados directamente con las funciones ampliadas del DOS; así que me reservo una exposición más detallada de los controladores de dispositivos para el apéndice A. Recuerde que el hecho de situar la exposición sobre los controladores de dispositivos al final del libro, no quiere decir de ninguna manera que disminuya su importancia. Todos los programadores que estén preocupados por el rango y la longevidad de sus programas, deben estar al menos familiarizados con el uso y operación de los controladores de los dispositivos. ➤ Véase apéndice A.

RESUMEN DE LA AMPLIACION DE FUNCIONES DEL DOS

Todas las funciones ampliadas del DOS se invocan mediante el número de interrupción 33 (21 hex). Las funciones individuales son seleccionadas, situando el número de función en el registro AH. Cualquier programa que use las funciones ampliadas debe comprobar el número de versión del DOS para asegurarse de que las funciones están disponibles. La función ampliada 48 (30 hex) proporciona este servicio.

Función		Grupo
Dec	Hex	
47-56	2F-38	Grupo ampliado de las funciones
57-59	39-3B	Grupo de directorio
60-70	3C-46	Grupo ampliado de manejo de los ficheros
71	47	Grupo del directorio
72-75	48-4B	Grupo ampliado de manejo de la memoria
76-79	4C-4F	Grupo ampliado de las funciones
84-87	54-57	Grupo ampliado de las funciones
88-98	58-62	Añadidos del DOS 3.00

Figura 17-3. Los grupos lógicos de llamadas ampliadas a una función del DOS

Las funciones están organizadas en los grupos lógicos de la figura 17-3. En un esfuerzo por hacer los agrupamientos lógicos de las llamadas a función lo más claramente posible, los he organizado y descrito de una forma un poco distinta que en el «Manual de Referencia Técnica del DOS». Puede comparar mi organización con la de IBM, para asegurarse de que no hay ningún malentendido. La figura 17-4 es una lista de las llamadas individuales a una función.

Función		Descripción	Función		Descripción
Dec	Hex		Dec	Hex	
47	2F	Obtener la dirección del DTA	71	47	Obtener el directorio activo
48	30	Obtener el número de versión del DOS	72	48	Asignar la memoria
49	31	KEEP: terminar-permaneciendo-residente avanzado	73	49	Memoria libre asignada
50	32	No usado	74	4A	SETBLOCK: modificar el bloque de memoria asignada
51	33	Obtener/modificar el estado de Ctrl-Break	75	4B	EXEC: cargar/ejecutar un programa
52	34	No usado	76	4C	Terminar el proceso
53	35	Obtener el vector de interrupción	77	4D	Obtener el código de retorno de un subprograma
54	36	Obtener el espacio libre del disco	78	4E	FIND FIRST: empieza la búsqueda de ficheros
55	37	No usado	79	4F	Continuar la búsqueda de ficheros
56	38	Obtener información dependiente del país	80	50	No usado
57	39	MKDIR: crear un directorio	81	51	No usado
58	3A	RMDIR: borrar un directorio	82	52	No usado
59	3B	CHDIR: cambiar el directorio actual	83	53	No usado
60	3C	CREAT: crear un fichero	84	54	Obtener el estado de verificación
61	3D	Abrir un fichero	85	55	No usado
62	3E	Cerrar el número del fichero	86	56	Renombrar fichero
63	3F	Leer desde un fichero o un dispositivo	87	57	Obtener/modificar la fecha y la hora de un fichero
64	40	Escribir en un fichero o en un dispositivo	88	58	No usado
65	41	Borrar el fichero	89	59	Obtener el código de error ampliado
66	42	Mover el puntero del fichero	90	5A	Crear un fichero transitorio
67	43	CHMOD: obtener/modificar los atributos del fichero	91	5B	Crear un fichero nuevo
68	44	IOCTL: control de entrada/salida para los dispositivos	92	5C	Bloquear/desbloquear acceso fichero
69	45	DUP: duplicar el número del fichero	93	5D	No usado
70	46	CDUP: forzar la duplicación del número del fichero	94	5E	No usado
			95	5F	No usado
			96	60	No usado
			97	61	No usado
			98	62	Obtener la dirección del PSP

Figura 17-4. Las llamadas ampliadas a funciones del DOS, llamadas mediante la interrupción 33 y seleccionadas en el registro AH

Función 47 (2F hex): Obtener la dirección del DTA

La función 47 devuelve la dirección de la zona de transferencia del disco, o DTA, que es usada actualmente por el DOS. La dirección se deposita en el par de registros ES:BX. Dado que se usa el registro de segmento para devolver la porción del segmento en vez del registro DS, este servicio no tendrá conflictos con la operación normal de un programa. ■ Compare esto con la función 26, expuesta en el página 316.

Función 48 (30 hex): Obtener el número de versión del DOS

La función 48 devuelve el número principal y secundario de la versión. El número principal de la versión está en AL (para el DOS 2.10, sería 2), y el número secundario de la versión está en AH (que sería 10 para el DOS 2.10). Este servicio es nuevo en las versiones DOS avanzadas, y no existía en las versiones DOS-1. Afortunadamente, cuando la función 48 es usada con cualquier versión DOS-1 se obtiene un resultado consistentemente identificable: el número principal de la versión será depositado como 0 en el registro AL. Desafortunadamente, el número secundario de la versión no es de fiar en las versiones DOS-1.

Cualquier programa que haga uso de las características avanzadas del DOS puede usar esta función para determinar si está siendo usada la versión del DOS apropiada. La mejor prueba es comprobar si el número principal de la versión es igual o más grande que 2 (o 3, si se están usando los servicios del DOS-3). Si la función da como información un número menor que 2 (o que 3), el programa deberá interrumpirse dando un mensaje.

Función 49 (31 hex): KEEP avanzado. Terminar permaneciendo residente

La función 49 es la versión avanzada de la interrupción universal del DOS 39, el servicio de terminar permaneciendo residente. Además de acabar un programa, la función 49 permite comunicar un código de retorno, que se sitúa en el registro AL y puede ser examinado con ERRORLEVEL del proceso por lotes del DOS.

Cualquier programa que finalice de esta forma le debe decir al DOS la parte que se mantiene en memoria y la que se desecha. Esto se hace pasando un valor de segmento en DX, que especifica el final

de la parte residente. Esto controla la cantidad del programa que va a ser conservada y la cantidad que va a ser liberada.

Un programa puede pedir memoria adicional usando los servicios de asignación de memoria, las funciones desde la 72 a la 75. Cualquier cantidad de memoria que sea asignada por estos servicios no es liberada por el valor especificado en DX. DX sólo lleva la cuenta de la memoria que asignó el DOS cuando el programa fue cargado inicialmente, una asignación que no está bajo el control del programa. Generalmente si asignamos memoria libremente en un programa residente, estaremos andando en la cuerda floja (pero con cuidado se puede hacer).

Función 51 (33 hex): Obtener/modificar el estado de Ctrl-Break

La función 51 comunica, o controla, el estado de procesos Ctrl-Break. Recordará que el DOS actúa sobre Ctrl-Break sólo bajo un conjunto limitado y peculiar de circunstancias (■ véase página 289). Una de las mejoras aparecidas en la versión DOS 2.00 es que se puede ampliar la detección de Ctrl-Break para que tenga lugar durante cualquier operación del DOS. Para la compatibilidad con las versiones DOS-1, esta detección ampliada de Ctrl-Break puede ser opcional. Se puede controlar de tres formas: mediante el comando BREAK, mediante la inicialización de fichero CONFIG.SYS y mediante esta función, que permite a cualquier programa hacer una parada de Ctrl-Break.

Si AL tiene el valor 0, entonces podemos solicitar el estado de ruptura, del que se informa en DL. DL = 0. significa que la ruptura está desactivada; DL = 1 significa que está activada (la parada extra es efectiva). Si AL toma el valor a 1, podemos ajustar el estado de ruptura usando el mismo código en DL. DL = 0 deshabilita la ruptura, y DL = 1 habilita la ruptura.

Función 53 (35 hex): Obtener el vector de interrupción

La función 53 devuelve el vector de interrupción del número de interrupción especificado en el registro AL. El vector se devuelve en el par de registros ES:BX.

Hay varias utilidades para este servicio. La más obvia es preservar el vector de interrupción actual antes de cambiarlo con la función 37 (25 hex) para que pueda ser realmacenado más tarde. Otro uso es simplemente comprobar el valor actual del vector; por ejemplo,

podemos usar esta función para ver si el ajuste por defecto del vector ha sido cambiado. Una aplicación particularmente útil es averiguar el valor de los vectores de interrupción que señalan a tablas en vez de señalar a subrutinas de manejo de interrupciones. Conocido el valor del vector, acceder a la información de la tabla es una cuestión simple.

Función 54 (36 hex): Obtener el espacio libre del disco

La función 54 proporciona un montón de información interesante y útil sobre espacio de un disco (mucho más que el espacio libre que anuncia el nombre del servicio). Antes de llamar a este servicio, se selecciona con el registro DL la unidad en la que se está interesado: DL = 0 indica la unidad por defecto; DL = 1 indica la unidad A; DL = 2 indica la unidad B; etc. Advierta la diferencia entre esta nomenclatura y la más convencional, en la que 0 = unidad A. Como en todos los servicios del DOS, y a diferencia de los servicios de la ROM-BIOS, se trata de unidades lógicas, no de unidades físicas.

Si hay un error (por ejemplo, una unidad no válida), se comunica en el registro AX con FFFF hex. De otro modo, AX contiene los sectores por unidad de asignación (*cluster*), CX contiene los bytes por sector, BX contiene el número total de *cluster* disponibles y DX contiene el número total de *cluster*.

A partir de estos números, podemos calcular muchas cosas interesantes sobre el disco. Por ejemplo, podemos usar estas fórmulas para calcular lo siguiente:

CX * AX	' bytes por <i>cluster</i>
CX * AX * BX	' número total de bytes libres
CX * AX * DX	' espacio total de almacenamiento
(BX * 100) / DX	' porcentaje de espacio libre

Si S fuera el tamaño en bytes de un fichero, entonces se podría calcular el número de sectores ocupados de la siguiente manera:

$$(S + CX * AX - 1) / (CX * AX)$$

Fórmulas similares darían el número de *cluster* y la cantidad y proporción de espacio que está asignado a un fichero, pero que no es usado (el «espacio inactivo»).

Función 56 (38 hex): Obtener información dependiente del país

La función 56 proporciona un servicio interesante que permite a los programas ajustarse automáticamente a diferentes monedas

internacionales y convenciones del formato de tiempo, usando una tabla de información específica del país suministrada por el DOS. La versión DOS-2 de este servicio comunica un juego de información del país muy pequeña. La ampliación DOS-3 comunica la información para cualquier código del país en la tabla del DOS.

El registro AL debe estar ajustado a 0 para obtener la información del país estándar. Para el DOS 3.00, o versiones posteriores, el registro AL puede ser ajustado a un código de país predeterminado. (El código del país es el mismo código de 3 dígitos usado en el sistema telefónico.) Para acceder a más de 255 códigos de país, AL puede ser ajustado a 255 (FF hex) y el código del país puede ser colocado en el registro BX. Si el código de país solicitado es inválido, el DOS activa el indicador de acarreo y sitúa un código de error en AX. De otro modo, BX contiene el código del país, y la zona de 32 bytes se llena con la información específica del país mostrada en las figuras 17-5 y 17-6. El par de registros DS:DX señala al principio del área de 32 bytes.

Los programas que operan bajo las versiones DOS-2 sólo pueden solicitar el juego estándar de información específica de país codificada en el DOS, ajustando AL a 0. Los programas que operan bajo las versiones DOS-3 pueden solicitar la información estándar de cualquier país cuyo código exista en la tabla (AL = código), o pueden simplemente ir buscando códigos válidos. Si se quiere explorar los códigos, se puede escribir un programa sencillo que pruebe todos los posibles códigos de país, y exponer los válidos.

La zona de información de 32 bytes es codificada de manera un tanto diferentes en las versiones DOS-2 y en las versiones posteriores DOS-3. Aunque la codificación pueda parecer similar, los dos formatos no son compatibles.

Campo	Dirección relativa	Tamaño (bytes)	Descripción
1	0	2	Código de fecha y hora
2	2	2	Cadena del símbolo monetario
3	4	2	Cadena del separador de millares
4	6	2	Cadena del separador de decimales
5	8	24	Sin usar

Figura 17-5. La información dependiente del país suministrada por las versiones DOS-2 y situada en la zona de 32 bytes señalada por DS:DX

Campo	Dirección relativa	Tamaño (bytes)	Descripción
1	0	2	Código de la fecha y la hora
2	2	5	Cadena del símbolo monetario (formato ASCIIZ)
3	7	2	Cadena del separador de millares (formato ASCIIZ)
4	9	2	Cadena del separador decimal (formato ASCIIZ)
5	11	2	Cadena del separador de la fecha (formato ASCIIZ)
6	13	2	Cadena del separador de la hora (formato ASCIIZ)
7	13	1	Situación del símbolo monetario: 1 = antes; 0 = después
8	16	1	Decimales en el formato de moneda
9	17	1	Formato de la hora: 1 = reloj de 24 horas; 0 = 12 horas
10	18	4	Dirección de llamada del mapa de mayúsculas/minúsculas
11	22	2	Cadena del separador de la lista (formato ASCIIZ)
12	24	8	Reservadas

Figura 17-6. La información dependiente del país suministrada por las versiones DOS-3 mediante la función 56 (38 hex) cuando AL=código del país

El **campo 1** (en ambos formatos) contiene un entero de una palabra cuyos valores especifican el formato de la hora y de la fecha. Hay tres valores predefinidos para esta palabra, y tres formatos de fecha y hora correspondientes. (● Véase figura 17-7.) Podrían añadirse otros en el futuro.

Valor	Uso	Hora	Fecha
0	Americano	h:m:s	m-d-a
1	Europeo	h:m:s	d-m-a
2	Japonés	h:m:s	a-m-d

Figura 17-7. Los tres formatos de hora y fecha predefinidos, cuyos códigos se especifican en los primeros dos bytes de la zona de 32 bytes señalada por DS:DX

En el formato del DOS-2 los siguientes campos contienen tres cadenas ASCIIZ que constan de dos bytes cada una; un byte de datos seguido por el byte cero estándar que finaliza las cadenas ASCIIZ.

El **campo 2** contiene la primera cadena, que nos da el símbolo monetario (un signo de dólar (\$) en los Estados Unidos).

El **campo 3** guarda la segunda cadena, que da el símbolo usado para apuntar las unidades de millar en los números (en U.S., es una coma, como en el número 12,345; otros países usamos un punto, o un espacio en blanco).

El **campo 4** guarda la tercera cadena, que da el símbolo decimal (en Estados Unidos, en un punto, como en 3.00; otros países usamos una coma).

Los 24 bytes restantes de la zona de información de 32 bytes no se usan en el formato del DOS-2.

El formato del DOS-3 comienza con la misma palabra entera que indica el formato de la fecha y la hora en el formato del DOS-2. A continuación hay las tres mismas cadenas ASCIIZ que definen los símbolos de la moneda, los millares y los decimales —con una diferencia importante: a la cadena ASCIIZ del símbolo monetario del campo 2 se le asignan cinco bytes, permitiendo que el símbolo monetario sea desde un único símbolo (por ejemplo, un signo de dólar, o de yen) hasta 4 letras (por ejemplo, una de las cadenas de moneda usadas por el DOS 3.00 es DKR, que representa a la corona danesa).

Los **campos 5 y 6** son también de dos bytes de longitud. El primero da la puntuación usada en las fechas (por ejemplo, “-” en 7-4-1985); el segundo da el símbolo usado para puntuar la hora (por ejemplo, “:” en 12:34).

El **campo 7** es un único byte que se usa para indicar dónde debe ir situado el símbolo monetario. Un cero sitúa el símbolo detrás de la cantidad (10 DKR), un uno sitúa al símbolo delante de la cantidad (\$10.000).

El **campo 8** contiene un entero de un byte que especifica cuántas posiciones decimales se usan en la moneda. Por ejemplo, el valor sería 2 para la moneda de Estados Unidos (dólares y centavos) y 0 para la moneda italiana (lira).

El **campo 9** es un campo de 1 byte que está codificado en bits para especificar el formato de la hora. Solamente el primer bit (el bit 0) se usa realmente; si el bit 0 es 0, se usa un reloj de 12 horas, y si es 1, se usa un reloj de 24 horas.

El **campo 10** guarda una dirección segmentada de 4 bytes de una subrutina que se usa para determinar el uso de las letras mayúsculas y minúsculas.

El **campo 11** contiene una cadena ASCIIZ de 2 bytes que da el símbolo usado para separar términos en una lista, como las comas en la lista A, B, C y D.

El **campo 12** guarda los restantes ocho bytes del área de 32 bytes, que se reservan para uso futuro.

Función 57 (39 hex): MKDIR. Crear un directorio

La función 57 crea un subdirectorio, tal como lo hace el comando MKDIR del DOS. Para llamar a este servicio, se introduce una cadena ASCIIZ que contiene el nombre de ruta del nuevo directorio, seguido de un byte cero para delimitar la cadena. El par de registros DS:DX señala a la dirección de la cadena ASCIIZ. Los errores se comunican en AX en el formato estándar para los comandos ampliados (☛ véase página 335). Los posibles códigos de errores son: 3 (ruta no encontrada) y 5 (acceso denegado).

Función 58 (3A hex): RMDIR. Borrar un directorio

La función 58 borra un subdirectorio, del mismo modo como lo hace el comando RMDIR del DOS. Para llamar a este servicio, introducimos una cadena ASCIIZ que contiene el nombre de ruta del directorio que queremos retirar, seguido de un byte cero para delimitar la cadena ASCIIZ. Los errores se comunican mediante AX en el formato estándar de los comandos ampliados (☛ véase página 335). Los códigos de error posibles son: 3 (ruta no encontrada) y 5 (acceso denegado).

El DOS no retirará ni el directorio activo ni cualquier directorio que tenga ficheros, o subdirectorios. En este caso se indica el código 5 (acceso denegado) —o eso dice la documentación del DOS. El sentido común dice que el intento de retirar el directorio activo debe acabar en el código más específico de error 16 (intento de retirar el directorio activo).

Función 59 (3B hex): CHDIR. Cambiar el directorio activo

La función 59 cambia el directorio activo, de la misma manera como lo hace el comando CHDIR. Para llamar a este servicio, se introduce una cadena ASCIIZ que contiene el nombre de ruta del nuevo directorio, seguido de un byte cero para delimitar la cadena ASCIIZ. DS:DX contiene la dirección de la cadena de entrada. Los

errores se comunican mediante AX en el formato estándar de los comandos ampliados (☛ véase página 335). El único código posible de error es 3 (ruta no encontrada).

Función 60 (3C hex): CREAT. Crear fichero

La función 60 abre un fichero ya existente, o crea uno nuevo, lo que es la operación estándar de encontrar o crear para ficheros de salida. La función 22 es prácticamente similar (☛ tratada en la página 315).

Para llamar a este servicio, se proporciona una cadena ASCIIZ que contiene el nombre de ruta y el nombre del fichero, seguido de un byte cero para delimitar la cadena. El par de registros DS:DX señala a la dirección de la cadena ASCIIZ. CX, o realmente CL, contiene el atributo del fichero. (☛ Véase página 130, más información sobre los atributos de fichero y los valores de los bits de atributo.) El número del fichero se deposita en AX.

Esta función abre cualquier fichero, nuevo o viejo, para el acceso de lectura/escritura, pero supone que se estamos escribiendo en un fichero de salida, y ajusta a cero la longitud del fichero existente.

Los códigos posibles de retorno son: 3 (ruta no encontrada), 4 (no hay número (*handle*) disponible) y 5 (acceso denegado). El código 5 puede indicar que no hay sitio para un nuevo elemento de directorio, o que el fichero existente está marcado como de sólo lectura y no puede ser abierto para salida. Ya que el registro AX se usa para devolver el número del fichero o el código retorno, podemos y debemos confiar en que el indicador CF señale un error (☛ véase la explicación de la página 335).

Función 61 (3D hex): Abrir fichero

La función 61 es la forma más general de abrir un fichero. Se proporciona el camino del nombre y el nombre del fichero en la forma de cadena ASCIIZ, acabada en un byte cero. Como con las otras funciones de entrada/salida a ficheros, DS:DX señala la cadena. También indicamos cómo queremos usar el fichero, situando un código de modo en el registro AL. Los 8 bits de AL están divididos en los cuatro campos que se muestran en la figura 17-8.

El modo de apertura para las versiones DOS-2 es simple: solamente se usan los bits de acceso (AAA en la figura 17-8) y todos los demás bits se ajustan a cero. Los valores de códigos de acceso están definidos en la figura 17-9.

Bit								Uso
7	6	5	4	3	2	1	0	
1	.	.	.	.	.	.	.	Indicador de herencia (DOS-3)
.	S	S	S	.	.	.	.	Modo de compartición (DOS-3)
.	.	.	.	R	.	.	.	Reservado para uso futuro (DOS-3)
.	.	.	.	.	A	A	A	Código de acceso (DOS-2 y DOS-3)

Figura 17-8. Los códigos situados en el registro AL para especificar el modo de apertura para la función 61

El DOS 3.00 usa los códigos de herencia y de compartición, así como el código de acceso. Los códigos del DOS-3 son más complicados, ya que deben tener en cuenta los problemas de compartición de ficheros (una característica introducida con el DOS 3.00).

El bit 7, el bit de herencia, indica si hay, o si no hay, «procesos hijos» (programas que se ejecutan como subprogramas semiindependientes bajo este programa) que heredarán el uso de este fichero. Si el bit 7 es 0, los procesos hijos tendrán el uso del fichero. Si el bit 7 es 1, el fichero es privado para este programa, y los procesos hijos no tendrán acceso directo a él. (Pero, como cualquier otro programa, pueden solicitar el programa ellos mismos, compartiéndolo.)

Los bits del 4 al 6, los bits de modo de compartición (SSS en la figura 17-8), definen lo que ocurrirá cuando se realice un intento para reabrir el mismo fichero más de una vez. Hay cinco modos de compartición: el modo de compartibilidad (SSS = 000), el modo de rechazo de lectura/escritura (SSS = 001), el modo de rechazo de escritura (SSS = 010), el modo de rechazo de lectura (SSS = 011) y el modo de ningún rechazo (SSS = 100). El modo de compartición es bastante complejo y es mejor acudir al «Manual de Referencia Técnica del DOS» para buscar una explicación.

El bit 3, señalado como reservado, en la figura 17-8, debe estar ajustado a 0.

Bit			Uso
2	1	0	
0	0	0	Acceso de lectura (solamente)
0	0	1	Acceso de escritura (solamente)
0	1	0	Acceso de lectura o escritura

Figura 17-9. Los valores del código de acceso de la función 61 en el registro AL

Hay que advertir que si no se especifica ni el atributo del fichero ni el tamaño del registro, este servicio encontrará cualquier fichero existente, incluyendo los escondidos, y ajustará el tamaño del registro a un byte, por defecto.

Los códigos posibles de retorno son: 2 (fichero no encontrado), 4 (fichero no encontrado), 5 (acceso denegado) y 12 (código de acceso no válido). Ya que el mismo registro (AX) es usado para devolver el número del fichero y el código de retorno, se puede y se debe confiar en el indicador CF para señalar un error. (● Véase explicación en la página 335.)

Función 62 (3E hex): Cerrar el número del fichero

La función 62 cierra un fichero y elimina todos los *buffers* asociados con el número dado en el registro BX. BX debe contener el número del fichero que fue devuelto por la operación de apertura del fichero. El único código de error posible es el 6 (número no válido).

Función 63 (3F hex): Leer desde fichero, o dispositivo

La función 63 lee el fichero, o dispositivo (que actúa como un fichero), asociado con el número del fichero dado en BX. El registro CX especifica el número de bytes a leer. DS:DX señala a la dirección del *buffer* dónde serán situados los datos que se lean. Después de que la función se ha realizado, AX contiene el número de bytes leídos. Si este valor es cero, significa que el programa ha intentado leer el final de un fichero.

Los códigos de error posibles son el 5 (acceso denegado) y el 6 (número no válido). Ya que el mismo registro (AX) se usa para devolver el número de bytes que se han leído, o el código de retorno, podemos y debemos confiar en el indicador CF para indicar un error (● véase explicación en el capítulo 17).

Función 64 (40 hex): Escribir en fichero, o dispositivo

La función 64 escribe en el fichero, o el dispositivo, con el número de fichero dado en BX. CX especifica el número de bytes a ser escritos y DS:DX señala a la dirección de los bytes de datos. Después de que la función se completa, AX contiene el número real de bytes escritos.

Hay dos maneras de detectar errores. Una manera es mediante un código de verdadero error señalado por CF. Si CF señala un error, los códigos de error posibles devueltos en AX son el 5 (acceso denegado)

y el 6 (número no válido). Si no es detectado un error evidente, pero la función todavía termina en error, podría significar que no hay suficiente espacio en el disco para escribir el fichero completo. Para comprobar esto, mire si hay menos bytes escritos (AX) de los que se pretendía (CX).

Función 65 (41 hex): Borrar fichero

La función 65 borra el elemento de directorio de un fichero. El fichero se especifica por una cadena ASCII, con el nombre de ruta y el nombre del fichero, seguidos por un byte cero final. El par de registros DS:DX señala a la cadena.

Los ficheros de sólo lectura no pueden ser borrados por esta función. Para borrar un fichero de sólo lectura, debe cambiarse el atributo del fichero a 0, usando la función 67 (43 hex). Los caracteres comodín ? y * no pueden ser usados.

Los códigos de error que pueden ser devueltos en AX son el 2 (fichero no encontrado) y el 5 (acceso denegado).

Función 66 (42 hex): Mover el puntero del fichero

La función 66 se usa para cambiar la posición lógica de lectura/escritura en un fichero. Para llamar a este servicio, se carga BX con un número de fichero y después se especifica la posición nueva del puntero, cargando en AL la posición de comienzo del puntero, y el número de bytes que se quieren mover en el par de registros CX:DX. La dirección relativa del byte en CX:DX es un entero largo sin signo de 32 bits. CX es la parte de mayor peso de la dirección relativa (que es 0, salvo que la cantidad de dirección relativa sea superior a 65 535) y DX es la parte de menor peso.

La posición de arranque especificada en AL se llama «código de método», y hay tres opciones: si AL = 0, la dirección relativa se toma del principio del fichero y el puntero se mueve CX:DX bytes a partir de ese punto; si AL = 1, se toma la dirección relativa de la posición actual, y si AL = 2, la dirección relativa se toma del fin de fichero. En este último caso, normalmente se ajusta la dirección relativa en CX:DX a 0 para averiguar el tamaño del fichero. Si se ajusta la dirección relativa a 0 y se pide el método 0, se vuelve al principio del fichero.

Después de que la función ha sido realizada, el par de registros DS:AX contiene el puntero del fichero como una dirección relativa en bytes desde el principio del fichero. El puntero se devuelve como un

entero largo de 32 bits, con la parte de mayor peso en DX y la de menor peso en AX. Advierta la curiosa elección del par de registros.

Los códigos posibles de error son el 1 (número de función no válido, que se refiere a la subfunción del código de método) y el 6 (número no válido). Ya que el mismo registro (AX) se usa para devolver tanto parte de la nueva posición como el código de retorno, se puede y se debe confiar en que el indicador CF señale el error. (☛ Véase página 335.)

Función 67 (43 hex): CHMOD. Obtener/modificar los atributos del fichero

La función 67 obtiene, o modifica, los atributos de un fichero. (☛ Véanse los detalles sobre los atributos del fichero en la página 130.) DS:DX señala a una cadena ASCIIZ, que proporciona la designación del fichero en cuestión (los caracteres comodín ? y * no pueden ser usados). Un valor de AL=0 modificará los atributos con los valores encontrados en CX; AL=1 ajustará los valores encontrados en CX. En ambos casos, los valores son tomados realmente del registro CL.

Los códigos posibles de error son el 2 (fichero no encontrado), 3 (ruta no encontrada) y 5 (acceso denegado).

Función 68 (44 hex): IOCTL. Control de entrada/salida para los dispositivos

La función 68 realiza un número de operaciones de control de entrada/salida, que se agrupan todas juntas en un paquete (☛ véase figura 17-10). AL selecciona una de las diez subfunciones, numeradas desde el 0 hasta el 8, y 11. (Las subfunciones 8 y 11 se aplican a las versiones DOS-3.) BX devuelve el número del fichero.

Subfunciones 0 y 1. Estas subfunciones obtienen o modifican la información del dispositivo contenido en DX mediante una complicada codificación de bits. El bit 7 se ajusta a 1 para los dispositivos, y a 0 para los ficheros de disco. Para los dispositivos, los bits del 0 al 5 se listan en la figura 17.11. Para los ficheros, los bits 0 al 5 proporcionan el número de la unidad de disco: un valor de 0 representa a la unidad A, un valor de 1 representa a la unidad B, etc.

Subfunciones de la 2 a la 5. En estas cuatro funciones, CX especifica el número de bytes a leer, o escribir, del canal de control de la unidad (no de un fichero dentro de la unidad), y el par de registros DS:DX señala el área de datos. Para las subfunciones 4 y 5, BL contiene el número especial de unidad (0 = por defecto, 1 = unidad A, 2 unidad B, etc.).

Código	Descripción
0	Obtener la información de dispositivo (devuelta en DX)
1	Modificar la información de dispositivo (desde DX, DH debe ser 0)
2	Lectura (véanse notas)
3	Escritura (véanse notas)
4	Lectura desde disco (véanse notas)
5	Escritura en disco (véanse notas)
6	Obtener el estado de la entrada (véanse notas)
7	Obtener el estado de la salida (véanse notas)
8	Si el dispositivo es retirable; solamente en las versiones DOS-3 (véanse notas)
11	Cambiar el número de intentos de acceso; solamente las versiones DOS-3 (véanse notas)

Figura 17-10. Las diez operaciones de control de entrada/salida (E/S), proporcionadas por la función 68 mediante el registro AL

Los datos sólo pueden ser leídos y escritos usando estas subfunciones si el dispositivo, o la unidad de disco, puede procesar cadenas de control. El estado de lectura/escritura se indica por el bit 14 en los bits de estado de DX.

Subfunciones 6 y 7. En estas subfunciones, el estado de entrada o salida se comunica en el registro AX. Si AX = 255 (FF hex), el dispositivo/unidad está listo para entrada/salida; si AX = 0, no está listo.

Subfunción 8. Esta subfunción, que se aplica sólo al DOS-3 y a las versiones posteriores, indica si un dispositivo tiene soportes que se pueden extraer o retirar (los *diskettes* en una unidad de disco son retirables; el disco duro no lo es). La subfunción 8 puede ser extremadamente útil, ya que permite a un programa conocer si tiene que vigilar un cambio de disco o si puede confiar en que el mismo disco siempre está allí. Un dispositivo retirable se indica con AX = 0; un disco fijo, con AX = 1, y un dispositivo no válido se indica con AX = 15 (F hex).

La subfunción 11 (B hex). Esta subfunción sólo se aplica al DOS-3 y a las versiones posteriores, y se usa para controlar los intentos que se hacen para resolver conflictos de compartición de ficheros. Los problemas en la compartición de ficheros suelen ser transitorios, ya que algunos programas usan ficheros sólo brevemente. El DOS intenta más de una vez el acceso a un fichero compartido antes de comunicar un conflicto, con la esperanza de que, en el tiempo

Dit																Uso
F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	X	1 = Entrada estándar de consola
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	X	1 = Salida estándar de consola
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	X	1 = Dispositivo nulo (no se necesita I/O)
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	X	1 = Dispositivo de reloj
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	X	1 = Dispositivo especial (puede significar cualquier cosa)
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	X	1 = Los datos se pasan en bruto; 0 = Los datos se pasan procesados
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	X	0 = Fin de fichero; 1 = No es fin de fichero (de entrada)
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	X	1 = Dispositivo; 0 = Fichero de la unidad de disco
.	.	.	.	.	.	.	R	.	.	.	.	.	.	.	.	Reservado
.	.	.	.	.	.	.	R	.	.	.	.	.	.	.	.	Reservado
.	.	.	.	.	.	.	R	.	.	.	.	.	.	.	.	Reservado
.	.	.	.	.	.	.	R	.	.	.	.	.	.	.	.	Reservado
.	.	.	.	.	.	.	R	.	.	.	.	.	.	.	.	Reservado
.	.	.	.	.	.	.	R	.	.	.	.	.	.	.	.	Reservado
.	X	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1 = El dispositivo puede procesar cadenas de control durante la lectura/escritura
R	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	Reservado

Figura 17-11. Los ajustes de bits del registro DX para las subfunciones 0 y 1 de la función 68

intermedio, el bloqueo del acceso desaparezca. Esta subfunción ajusta el número de reintentos, lo que se especifica en el registro DX, y el intervalo de tiempo entre ellos (en unidades no especificadas), que se ha de indicar con un valor de conteo en el registro CX.

Función 69 (45 hex): DUP. Duplicar el número del fichero

La función 69 duplica el número de un fichero abierto y devuelve un nuevo número de número que se refiere al mismo fichero, o dispositivo. Todas las acciones realizadas con un número serán automáticamente reflejadas en el otro número —el nuevo número no actúa independientemente en ningún modo. Véase la función 70 para un servicio similar.

El registro BX contiene el número del fichero anterior y AX devuelve el nuevo número del fichero. Los códigos posibles de error son el 4 (no hay número disponible) y el 6 (número no válido).

Hasta ahora no he podido encontrar una forma de usar esta función, pero estoy seguro de que debe haber una. Estas cosas no se crean como broma.

Función 70 (46 hex): CDUP. Duplicación forzada del número del fichero

La función 70, similar a la función 69, duplica el número de un fichero. Sin embargo, en este caso se utiliza un número existente (uno que ha sido usado para algún otro propósito) en vez de que el DOS cree un nuevo número. Si el segundo número se refiere a un fichero abierto, el fichero se cierra antes de que ocurra cualquier acción. Una vez llamada esta función, todas las acciones que se realicen con un número se reflejan automáticamente en el otro, así que el nuevo número no actúa independientemente.

El registro BX contiene el número viejo, y CX contiene el segundo número. Cuando se completa la operación, el número del fichero en CX se referirá al mismo dispositivo, o fichero, que BX. El único código posible de error es el 6 (número no válido).

Aunque este servicio parezca algo peculiar, tiene algunos usos evidentes. Por ejemplo, un programa puede redirigir cualquiera de los dispositivos E/S estándar (véanse una lista y una descripción en la página 353) usando el siguiente método. Supóngase que el programa desea redirigir dinámicamente la salida de la impresora a un fichero. Primero, se abre el fichero, lo que devuelve un número en BX. Entonces este número puede ser duplicado sobre el número estándar de impresora cargando el número 4 en CX. Después de esto, cualquier salida de la impresora estándar irá al fichero abierto. Para restaurar la dirección normal de la impresora, pero manteniendo el número para un uso posterior, hay que salvar el número normal de la impresora duplicando sobre otro número antes de que se copie el nuevo número sobre el número de la impresora. Si ha entendido esto, se puede considerar cualificado para intentarlo. Véase un ejemplo de esta técnica en la función 75 (4B hex).

Función 71 (47 hex): Obtener el directorio activo

La función 71 comunica el directorio actual en la forma de una cadena ASCIIZ. Se debe especificar el número de la unidad en DL (0 = defecto, 1 = unidad A, 2 = unidad B). El par de registros DS:SI señala el área de datos que contendrá el nombre de ruta, que puede ser hasta de 64 bytes de largo. El DOS devuelve el nombre de ruta entero del directorio activo de la unidad especificada, incluyendo el directorio raíz. El nombre de ruta siempre es seguido por un byte cero —el delimitador de la cadena ASCIIZ.

Aunque esta función devuelve el nombre del fichero entero, el nombre no incluye ni el identificador de la unidad (como A:) ni la

barra inclinada del principio de la raíz (como en A:\). Según estas reglas, si el directorio activo es el directorio raíz (un caso normal en los *diskettes*), entonces el resultado será una cadena nula. Si se precisa una representación inteligible del directorio, debe prefijar la información devuelta por esta función con los indicadores de unidad y raíz (como A:\). El único código posible de error es el 15 (especificación de unidad inválida).

Función 72 (48 hex): Asignar memoria

La función 72 asigna dinámicamente la memoria. Se indica en BX el número de párrafos (unidades de 16 bytes) que se quieren asignar. A la vuelta, AX señala al segmento del bloque de memoria asignada.

Los códigos posibles de error son el 7 (bloques de control de memoria destruidos) y el 8 (memoria insuficiente). Si la función falla al asignar la memoria, el registro BX devolverá el tamaño del bloque más grande disponible.

Función 73 (49 hex): Liberar la memoria asignada

La función 73 devuelve al DOS la memoria que fue asignada mediante la función 72. El registro ES señala a la dirección del segmento del bloque que está siendo devuelto. Este es el mismo valor que la función 72 (48 hex) devuelve en el registro AX.

Los códigos posibles de error son el 7 (bloques de control de memoria destruidos) y el 9 (dirección del bloque de memoria no válida).

Función 74 (4A hex): SETBLOCK. Modificar el bloque de la memoria asignada

La función 74 se usa para incrementar, o decrementar, el tamaño de un bloque de memoria que fue asignada mediante la función 72. El registro ES señala a la dirección del segmento del bloque que será modificado. El registro BX contiene el nuevo tamaño del bloque en párrafos (unidades de 16 bytes).

Si una petición de incrementar el espacio no puede ser satisfecha, entonces BX devuelve el tamaño del bloque de memoria disponible (en párrafos).

Los códigos de error posibles son el 7 (bloques de control de memoria destruidos), el 8 (memoria insuficiente) y el 9 (dirección del bloque de memoria no válida).

Función 75 (4B hex): EXEC. Cargar/ejecutar un programa

La función EXEC permite a un programa cargar un subprograma en memoria y, opcionalmente, ejecutarlo. El par de registros DS:DX señala a una cadena ASCIIZ con el nombre de ruta y el nombre del fichero que va a ser cargado. El par de registros ES:BX señala a un bloque de parámetros que contiene la información de control de la operación de carga. AL especifica si el subprograma va a ser ejecutado después de ser cargado.

Si AL es 0, se carga el subprograma, se crea un prefijo del segmento del programa (PSP) y se ejecuta el programa. En este momento el control pasa al subprograma; solamente vuelve al programa cuando el subprograma termina. Si AL es 1, el subprograma se carga, no se crea ningún PSP y el programa no se ejecuta automáticamente, aunque se puede saltar hacia él. La variante AL = 1 se usa normalmente para cargar un añadido (*overlay*) de un programa. Es también una manera simple y efectiva de cargar datos en la memoria.

Cuando AL es 0, el bloque señalado por ES:BX es de catorce bytes de largo y contiene la información mostrada en la figura 17-12. Cuando AL es 1, el bloque señalado por ES:BX es de una longitud de cuatro bytes y contiene la información mostrada en la figura 17-13.

Cuando un programa es cargado y ejecutado, cualquier número de fichero que esté activo resulta disponible para el subprograma. Como se mencionó en la función 70 (46 hex), un programa puede redirigir los números de entrada/salida estándar y usar esta técnica para influir en la operación de un subprograma. Por ejemplo, se puede llamar al programa filtro de ordenación estándar para ordenar un grupo de

Dirección relativa	Tamaño (bytes)	Descripción
0	2	Dirección del segmento de la cadena del entorno
2	4	Puntero segmentado de la línea de comandos
6	4	Puntero segmentado del primer FCB activo
10	4	Puntero segmentado del segundo FCB activo

Figura 17-12. La información del bloque de control EXEC señalado por ES:DX cuando AL = 0. Cada uno de estos términos se refiere a la información construida en el PSP del programa que está siendo cargado

Dirección relativa	Tamaño (bytes)	Descripción
0	2	Dirección del segmento donde va a ser cargado el fichero
2	2	Factor de relocalización del programa (aplicado sólo a los programas de formato (EXE))

Figura 17-13. La información en el bloque de control EXEC que es señalado por ES:DX cuando AL = 1

ficheros, y dejar la salida, los mensajes y los mensajes de error donde se quiera.

Antes de usar esta función, se debe de estar seguro de que hay memoria suficiente para cargar el programa usando la función 74 (4A hex). Ya que el proceso de carga del programa se realiza como parte del intérprete de comandos, la parte semirresidente del intérprete debe estar intacta, o debe ser recargada desde el disco, si ha sido alterada.

Entre los programas que se pueden cargar y ejecutar se encuentra el intérprete de comandos DOS. Si se quiere, se puede cargar y pasarle una cadena de comandos, que podría llamar a un fichero *batch* —un fichero *batch* que nuestro programa original podría haber construido dinámicamente—. Este fichero *batch* podría llamar a sus programas y al final realizar el comando EXIT, que acabaría con la ejecución del intérprete de comandos secundario. En ese punto, el programa original recobraría el control. Esta facilidad proporciona grandes y complicadas posibilidades.

❑ **PRECAUCION:** *La función de cargar y ejecutar destruye la mayoría de los registros, incluyendo los registros de la pila, SS y SP. Estos deberían ser preservados antes de esta función y restaurados después. Obviamente la pila no puede ser usada para ello. Una manera poco elegante de tratar el problema es almacenar los registros de pila dentro del código del programa. Si se hace esto, las primeras instrucciones ejecutadas después de una vuelta de este servicio se dedicarán a restaurar los registros necesarios. Este tipo de técnica de programación (modificar directamente instrucciones de programa) es considerada detestable por partidarios de la correcta programación. Pero nunca prometimos que no se ensuciaría las manos.*

Los códigos posibles de retorno desde esta función son el 1 (número de función inválido), el 2 (fichero no encontrado), el 5 (acceso denegado), el 8 (memoria insuficiente), el 10 (medio inválido) y el 11 (formato inválido).

Función 76 (4C hex): Terminar un proceso

La función 76 finaliza un programa y devuelve un código de retorno. Si el programa fue llamado como un subprograma, el código de retorno se puede encontrar mediante la función 77. Si el programa se llamó como un comando DOS, entonces el código de retorno puede ser comprobado en un fichero *batch*, usando la opción del DOS ERRORLEVEL. El código de retorno se comunica en el registro AL.

Cuando se realiza esta función, el DOS cierra automáticamente cualquier fichero abierto con la función 61 (3D hex), y presumiblemente también con la función 60.

Función 77 (4D hex): Obtener el código de retorno de un subprograma

La función 77 obtiene el código de retorno de un subprograma terminado con la función 76. Hay dos partes en la información devuelta: AL comunica el código de retorno emitido por el programa; AH comunica cómo terminó el programa, y tiene cuatro resultados posibles: AH = 0 indica un final voluntario normal; AH = 1 indica finalización por el DOS debida a Ctrl-Break; AH = 2 indica la terminación por el DOS, debida a un error crítico de un dispositivo, y AH = 3 indica un final voluntario usando la función 49 (31 hex) terminar permaneciendo residente.

Función 78 (4E hex): FIND FIRST. Comenzar la búsqueda de ficheros

La función 78 busca el primero de los ficheros que se ajusta a una designación de fichero. Da valores a DS:DX para que señale a una cadena ASCII, que señale el nombre de ruta y el nombre del fichero que queremos buscar. El nombre del fichero puede contener los caracteres comodín ? y *. El registro CX (realmente CL) da la especificación de atributos que será utilizada para buscar el fichero. Si se encuentra un fichero, el DOS genera 43 bytes de información en la zona de transferencia del disco (DTA). (● Véase figura 17-14.)

La cadena ASCII, al final del área de información, almacena el nombre del fichero en su notación convencional, incluyendo un punto entre el nombre del fichero y la extensión. Si la extensión está en blanco, el punto no aparece.

Este servicio es similar a la función 17 (11 hex) del DOS. Los usos de los atributos del fichero en esta función de búsqueda son los mismos que en la función 17. (● Véase capítulo 16.)

Offset	Tamaño (bytes)	Descripción
0	21	Zona utilizada por el DOS para encontrar al siguiente (véase la función 79)
21	1	Atributo del fichero encontrado
22	2	Marca de la hora del fichero
24	2	Marca de la fecha del fichero
26	4	Tamaño del fichero en bytes
30	13	Nombre del fichero y extensión (cadena ASCIIZ)

Figura 17-14. La información devuelta en el DTA después de llamar a la función 78 (4E hex)

La búsqueda de atributos sigue una lógica particular. Si se especifica cualquier combinación de los bits de atributo de oculto, sistema o directorio, la búsqueda proporcionará ficheros normales, y también cualquier fichero con esos atributos. Si se especifica el atributo de etiqueta del volumen, la búsqueda sólo proporcionará un elemento de directorio con ese atributo. Los bits de archivo y de sólo lectura no se aplican a las operaciones de búsqueda. Los atributos de directorio, etiqueta de volumen, archivo y de sólo lectura no se aplican en las versiones anteriores a la 2.00.

Los códigos de error devueltos en AX son el 2 (fichero no encontrado) y el 18 (no hay más ficheros que encontrar). CF no señala el error.

Función 79 (4F hex): Continuar la búsqueda del fichero

La función 79 continúa la búsqueda de ficheros que fue comenzada mediante la función 78. Utiliza la información contenida al principio de la DTA, la cual no debe ser alterada.

El único código de error normal es el 18 (no hay más ficheros que encontrar). Para este código, CF no señala error.

☛ Véase también la función 78.

Función 84 (54 hex): Obtener el estado de verificación

La función 84 nos comunica el estado actual del conmutador de verificación, que controla si las operaciones de escritura en el disco son verificadas o no. AL = 0 indica que no son verificadas, AL = in-

dica que sí lo son. La función 46 (2E hex) activa el conmutador de verificación. (● Véase página 324.)

Esta función saca a colación una molesta incongruencia en los servicios del DOS: mientras algunos pares de los servicios de obtener/modificar están integrados en una única función (como la siguiente función 87), otros están divididos en dos funciones separadas, como la función 84 y la 46.

Función 86 (56 hex): Renombrar un fichero

Como el comando estándar RENAME del DOS, la función 86 cambia el nombre de un fichero. Pero también puede mover un elemento de directorio de un fichero desde un directorio a otro. El fichero en sí no se mueve, sino sólo el elemento de directorio, lo que significa que las rutas del directorio, nueva y vieja, deben estar en la misma unidad. Esta es una característica verdaderamente fabulosa y útil, y es bastante decepcionante que no sea parte del comando RENAME.

La función necesita dos entradas: las designaciones de fichero para ambos nombres el nuevo y el viejo.

Estos pueden ser designadores completos, con los componentes de la unidad y la nota. Las unidades especificadas o implicadas deben ser las mismas, así que el nuevo elemento del directorio estará en la misma unidad que el fichero. Los caracteres comodín * y ? no se pueden usar, ya que esta función trabaja solamente con los ficheros simples.

Como es usual, ambos designadores de fichero se suministran en la forma de cadenas ASCIIZ con un byte cero marcando el final. El par de registros DS:DX señala a la cadena del nombre antiguo, y ES:DI señala a la nueva cadena.

Los códigos posibles de error son el 3 (ruta no encontrada), el 5 (acceso denegado) y el 17 (ruta no encontrada).

Función 87 (57 hex): Obtener/modificar la fecha y la hora de un fichero

La función 87 obtiene, o modifica, la fecha y la hora del fichero. Recuerde que cada fichero se marca con la fecha y la hora en que fue creado, o modificado, por última vez. AL = 0 obtiene los datos y AL = 1 ajusta los datos.

El fichero se selecciona situando el número del fichero en BX, lo que significa que este servicio sólo se aplica a ficheros que han sido abiertos usando las funciones del DOS, tratadas en este capítulo.

Advierta, por tanto, que ajustar la marca de la hora de un fichero con este servicio sólo tendrá efecto cuando el fichero se cierre con éxito.

La fecha y la hora se sitúan en los registros CX y DX, con el mismo formato con el que fueron almacenadas en el directorio del disco, aunque en un orden un poco distinto. En esta función la hora se sitúa en CX, y la fecha en DX.

Contrariamente a lo que dice la documentación del DOS, la información de la fecha y la hora se sitúa en su formato convencional, con las partes de mayor peso en CH o DH, y las partes de menor peso en CL y DL.

La fecha y la hora pueden ser codificadas, o decodificadas, usando las siguientes fórmulas:

$$CX = \text{HORAS} * 2048 + \text{MINUTOS} * 32 + \text{SEGUNDOS} / 2$$
$$DX = (\text{AÑO} - 1980) * 512 + \text{MES} * 64 + \text{DIA}$$

Los códigos posibles de error en este servicio son el 1 (número de función inválido, basado en la subfunción seleccionada en AL, no en el número de función principal) y el 6 (número inválido).

FUNCIONES AÑADIDAS POR EL DOS 3.00

Hasta ahora, en este capítulo, hemos tratado las funciones que fueron introducidas con el DOS 2.00. El DOS 3.00 trajo mejoras para unas pocas funciones del DOS-2 y aportó cinco funciones nuevas a la familia de las funciones ampliadas.

Función 89 (59 hex): Obtener el código de error ampliado

La función 89 se usa después de que ocurre un error. Proporciona información sobre los errores que ocurren bajo las siguientes circunstancias: dentro de un manipulador de interrupción de error crítico; después de que una función solicitada mediante la interrupción estándar 33 (21 hex) ha comunicado un error mediante el indicador de acarreo (CF), y después de operaciones de ficheros al viejo estilo, utilizando FCB, que comuniquen un código de retorno de 255 (FF hex). No trabajará con las funciones del DOS que no comuniquen errores en CF, aun cuando hayan incurrido en un error.

Este servicio se llama de la manera estándar, situando el código de la función 89 (59 hex) en el registro AH. Para permitir que se tengan en consideración los cambios inevitables que con el tiempo se dan en este tipo de funciones, se debe especificar también un código de versión en el registro BX. Para la versión DOS 3.00, este código es el 0.

Se devuelven cuatro señales de información independientes cuando se termina este servicio: AX contiene el código error ampliado; BH indica la clase de error; BL da el código de cualquier opción que puede tomar nuestro programa, y CH da un código de localización que intenta mostrarnos dónde ocurrió el error.

Los códigos de error comunicados en AX se organizan en tres grupos: del código 1 al 18 se usan para los errores de llamadas a función (funciones de la interrupción 33), del código 19 al 33 se usan para errores de los manipuladores de error crítico (de la interrupción 36), y los códigos entre el 32 y el 83 se usan para los errores que son nuevos para los servicios de la versión DOS-3.00. Un código 0 indica que no hay error que comunicar por este servicio.

La figura 17-15 lista los códigos de error ampliados; la 17-16, las clases de error; la 17-17, los códigos de acción, y la 17-18, los códigos de localización.

Código	Significado	Código	Significado
1	Número de función inválido	19	Disco protegido contra escritura
2	Fichero no encontrado	20	Identificador desconocido
3	Ruta no encontrada	21	Unidad de disco no preparada
4	No hay número disponible	22	Comando no definido
5	Acceso denegado. (Ej. intento de escribir en un fichero de sólo lectura)	23	Error de datos del disco
6	Número inválido	24	Mala longitud en la estructura de petición
7	Bloques de control de memoria no válidos	25	Error de búsqueda en disco
8	No hay bastante memoria	26	Formato de disco desconocido
9	Dirección del bloque de la memoria inválido	27	Sector del disco no encontrado
10	Cadenas del comando SET inválidas («entorno»)	28	Impresora sin papel
11	Formato inválido (no se especifica de qué)	29	Error de escritura
12	Código de acceso a fichero inválido	30	Error de lectura
13	Datos inválidos	31	Fallo general
14	Reservado	32	Violación de compartición de un fichero
15	Especificación de unidad inválida	33	Violación del bloqueo de un fichero
16	petición de borrar el directorio activo	34	Cambio incorrecto de disco
17	No es el mismo dispositivo	35	No hay FCB disponible
18	No hay más ficheros que encontrar	80	El fichero ya existe
		81	Reservado
		82	No se puede hacer
		83	Fallo de la interrupción de error crítico

Figura 17-15. Los códigos de error ampliados devueltos en el registro AH siguiendo la ejecución de la función 89 (59 hex)

Código	Significado	Código	Significado
1	Sin recursos: no queda más de lo que se solicitó	6	Error del <i>software</i> del sistema: otros problemas del DOS
2	Situación transitoria: intente de nuevo más tarde	7	Error en la aplicación: es culpa del usuario
3	Autorización: no se está autorizado; alguien más podría estarlo	8	El asunto solicitado no es encontrado
4	Error interno del DOS: no es culpa del usuario	9	Mal formato (ej., disco irreconocible)
5	Fallo del <i>hardware</i>	10	Asunto bloqueado
		11	Error del soporte (ej., el disco comunica un error CRC)
		12	Ya existente
		13	La clase de error es desconocida

Figura 17-16. Las clases de error depositadas en el registro BH siguiendo la ejecución de la función 89 (59 hex)

Código	Significado	Código	Significado
1	Inténtelo de nuevo	4	Acabe el programa, pero de acuerdo con el usuario (cerrar ficheros, etc.)
2	Inténtelo de nuevo, después de un intervalo de espera	5	Acabe inmediatamente
3	Pida al usuario que lo fije (ej., cambie el disco); vea también código 7	6	Ignore el error: no es importante
		7	Vuelva a procesar después de la acción del usuario; vea también el código 3

Figura 17-17. Las acciones sugeridas depositadas en el registro BL por la ejecución de la función 89 (59 hex)

Código	Significado	Código	Significado
1	Desconocido: disculpe	3	Reservado
2	Error en el bloque del dispositivo (ej., unidad de disco)	4	Error de dispositivo serie (ej., impresora)
		5	Error de memoria

Figura 17-18. Los códigos de localización depositados en el registro CH por la ejecución de la función 89 (59 hex)

Función 90 (5A hex): Crear un fichero transitorio

La función 90 crea un fichero para uso transitorio, probablemente teniendo cuidado en la tarea de encontrar un nombre que no tenga conflictos con ningún fichero existente. Se proporcionan dos parámetros: el atributo del fichero, situado en el registro CX, y el nombre de ruta del directorio donde será creado el fichero. Si no se quiere especificar una ruta particular, se puede proporcionar al DOS una cadena nula, que indica que use el directorio activo de la unidad activa.

El nombre de ruta debe ser una cadena ASCIIZ y estar señalada por el par de registros DS:DX. La cadena del nombre de ruta debe estar lista para tener adjunto el nombre del fichero creado: esto significa que la cadena debe terminar con el carácter de la barra inclinada hacia atrás que se usa para señalar rutas de directorios. Se deben añadir también 12 bytes para dejar suficiente espacio al DOS para añadir a la cadena un nombre de fichero.

De vuelta, si hay un error en esta operación, el indicador de acarreo (CF) valdrá 1 y el código de error estará en AX. También el nombre de fichero se devolverá adjunto a la cadena de la ruta.

Este servicio se llama «crear un fichero transitorio». Solamente para sugerir su pretendido propósito. Realmente no hay nada transitorio en el fichero que se crea, ya que el DOS no lo borra automáticamente; nuestros programas deben encargarse de esa tarea.

Función 91 (5B hex): Crear un nuevo fichero

La función 91 es similar a la función 60 (3C hex), que es (erróneamente) llamada función de crear ficheros. La función 60 se diseñó realmente para buscar un fichero, y para crear uno si el fichero solicitado no existía. Como contraste, la función 91 es una función pura para crear un fichero, y se producirá un error si el fichero ya existe.

Como con la función 60, el registro CX toma los valores de atributo del fichero, y DS:DX señala a la dirección del nombre de ruta y del nombre del fichero (que se almacena como una cadena ASCIIZ). De vuelta, CF señala los errores, y AX devuelve los códigos de error.

Hay muchos casos en los que un programa intentará utilizar un fichero con un nombre determinado si éste existe, o crear un fichero con ese nombre si no existe. Este es el tipo de situación para la que está mejor preparada la función 60. Aun con todo, hay otras

circunstancias en que un programa puede no desear alterar ficheros ya existentes, sino solamente abrir un fichero que todavía no existe. Esta es la clase de situación para la que la función 91 está mejor preparada.

Función 92 (5C hex): Bloquear/desbloquear el acceso a un fichero

La función 92 se usa para bloquear ciertas partes de un fichero, de tal modo que puedan ser compartidos por varios programas sin que uno interfiera con las operaciones de otro. Si un programa bloquea una parte de un fichero, puede usar, o cambiar, esa parte del fichero mientras esté bloqueada, seguro en el conocimiento de que ningún otro programa será capaz de usar esa parte mientras permanezca bloqueada. Como usted puede haber imaginado, el bloqueo de ficheros sólo se usa en operaciones de ficheros compartidos.

Hay seis parámetros que determinan qué porción de un fichero será bloqueada. AL indica si estamos bloqueando (AL = 0) o desbloqueando (AL = 1) una porción del fichero. BX da el número del fichero. CX y DX son tratados juntos como un entero largo de 4 bytes, que especifica la dirección relativa del primer byte de la parte bloqueada en el fichero. SI y DI también forman un entero largo de 4 bytes, que especifica la longitud de la parte bloqueada. El primer registro en cada una de estos pares de registros (CX o SI) da la parte de mayor peso del entero.

No está autorizado desbloquear porciones de un fichero por etapas, o en combinación; una petición de desbloqueo debe acompañar exactamente una petición de bloqueo previa. Estamos avisados de que los bloqueos deben ser retirados antes de cerrar un fichero; contrariamente a lo que cabría esperar, el cerrar un fichero no borrará necesariamente los bloqueos que permanezcan en nuestro fichero.

Función 98 (62 hex): Obtener la dirección del PSP

La función 98 obtiene la dirección del prefijo del segmento del programa y lo deposita en BX como una dirección de segmento.

En el mundo convencional del DOS, los programas sitúan sus PSP en los primeros 256 bytes del segmento del código. Esto significa que la dirección del párrafo del PSP es la misma que el contenido del registro del segmento del código (CS). Sin embargo, como los ordenadores personales y el DOS empiezan a ser más complejos,

puede no ser siempre tan simple. Por ejemplo, en el modo protegido del microprocesador 80286 del AT, los registros de segmento son tratados de una nueva manera. Este servicio existe para proporcionar (eso espero) una manera permanente y práctica de tratar los registros de segmento en el futuro.

Resumen de los servicios del DOS

Resumen breve 368

Resumen amplio 370

Este capítulo es un resumen de las rutinas de servicio del DOS, y se diseñó para ser usado como una guía de referencia rápida. Para detalles sobre la operación específica de cada servicio y algunos comentarios sobre su operación, véanse los capítulos del 15 al 17. Una vez que se comprendan los servicios del DOS, estas tablas deberán proporcionar toda la información de programación que se necesite.

RESUMEN BREVE

Nueve son las interrupciones del DOS que se llaman a través de sus números. Cinco de estas interrupciones están listadas en la figura 18-1. Las cuatro interrupciones no mostradas en esta tabla se usan para propósitos especializados: la interrupción 33 (21 hex) se usa para llamar a cada una de las 80 funciones del DOS, y las interrupciones desde la 34 hasta la 36 son interrupciones de dirección que se usan para señalar a subrutinas especiales. ➤ Véase capítulo 15 para más información.

Las funciones universales del DOS, mostradas en la figura 18-2, son llamadas mediante la interrupción 33 (21 hex); el número de la función se sitúa en el registro AH. Las funciones universales pueden ser usadas con cualquier versión del DOS. ➤ Véase capítulo 16 para más información.

Las nuevas llamadas ampliadas a funciones del DOS pueden ser usadas solamente con las versiones DOS 2.00, o superiores. Son llamadas mediante la interrupción 35 (21 hex) y el número de función se sitúa en el registro AH. ➤ Véase capítulo 17 para más información. La figura 18.3 lista todas las funciones nuevas del DOS, incluyendo aquellas que fueron introducidas con la versión DOS 3.00. (Estas funciones no pueden ser usadas con versiones anteriores.)

Interrupción		Descripción
Dec	Hex	
32	20	Terminación de programa: ir a un final normal
37	25	Lectura de disco absoluta
38	26	Escritura del disco absoluta
39	27	Terminar permaneciendo residente
47	2F	Control de la cola de impresión (<i>spooler</i>) (DOS 3.00 y posteriores)

Figura 18-1. Las cinco interrupciones principales del DOS

Función		Descripción	Función		Descripción
Dec	Hex		Dec	Hex	
0	0	Terminar: fin del programa	23	17	Renombrar un fichero
1	1	Entrada del teclado con eco	25	19	Comunicar la unidad activa
2	2	Salida al display	26	20	Fijar la zona de transferencia del disco
3	3	Entrada serie	27	1B	Obtener información FAT de la unidad activa
4	4	Salida serie	28	1C	Obtener información FAT de cualquier unidad
5	5	Salida de impresora	33	21	Leer un registro de un fichero aleatorio
6	6	Entrada/salida directa teclado/display	34	22	Escribir un registro en un fichero aleatorio
7	7	Entrada directa del teclado sin eco	35	23	Obtener el tamaño de un fichero
8	8	Entrada del teclado sin eco	36	24	Fijar el campo de un registro aleatorio
9	9	Escribir cadena en pantalla	37	25	Fijar un vector de interrupción
10	A	Entrada del teclado con <i>buffer</i>	38	26	Crear un segmento de programa
11	B	Comprobación del estado de entrada del teclado	39	27	Leer registros de un fichero aleatorio
12	C	Borrar el teclado y ejecutar función	40	28	Escribir registros en un fichero aleatorio
13	D	Rearrancar el disco	41	29	Analizar el nombre del fichero
14	E	Seleccionar la unidad activa	42	2A	Obtener la fecha
15	F	Abrir un fichero	43	2B	Fijar la fecha
16	10	Cerrar un fichero	44	2C	Obtener la hora
17	11	Buscar un primer fichero	45	2D	Fijar la hora
18	12	Buscar el siguiente fichero	46	2E	Activar/desactivar la verificación de escritura en disco
19	13	Borrar un fichero			
20	14	Leer un registro de un fichero secuencial			
21	15	Escribir un registro en un fichero secuencial			
22	16	Crear un fichero			

Figura 18-2. Las funciones universales del DOS

Función		Descripción	Función		Descripción
Dec	Hex		Dec	Hex	
47	2F	Obtener la dirección del DTA	71	47	Obtener el directorio activo
48	30	Obtener la versión del DOS	72	48	Asignar memoria
49	31	KEEP: terminar permaneciendo residente avanzado	73	49	Liberar memoria asignada
			74	4A	SETBLOCK: modificar un bloque de memoria asignada
51	33	Obtener/ajustar control break			
53	35	Obtener el vector de interrupción	75	4B	EXEC: cargar/ejecutar un programa
54	36	Obtener el espacio libre del disco	76	4C	Terminar un proceso
56	38	Obtener información dependiente del país	77	4D	Obtener el código de retorno de un subprograma
57	39	MKDIR: crear directorio	78	4E	FIND FIRST: empezar la búsqueda de ficheros
58	3A	RMDIR: borrar directorio			
59	3B	CHDIR: cambiar el directorio activo	79	4F	Continuar la búsqueda de ficheros
60	3C	CREAT: crear un fichero	84	54	Obtener el estado de verificación
61	3D	Abrir un fichero	86	56	Renombrar un fichero
62	3E	Cerrar un número del fichero	87	57	Obtener/modificar la fecha y la hora de un fichero
63	3F	Leer desde fichero o dispositivo			
64	40	Escribir en un fichero o dispositivo	<i>Funciones del DOS 3.00</i>		
65	41	Borrar fichero			
66	42	Mover el puntero del fichero	89	59	Obtener el código de error ampliado
67	43	CHMOD: obtener/modificar los atributos del fichero	90	5A	Crear un fichero transitorio
			91	5B	Crear un fichero nuevo
68	44	IOCTL: control de entrada/salida para los dispositivos	92	5C	Bloquear/desbloquear el acceso a un fichero
69	45	DUP: duplicar el número del fichero	98	62	Obtener la dirección del PSP
70	46	CDUP: duplicación forzada del título			

Figura 18-3. Las funciones nuevas del DOS disponibles con el DOS 2.00 y las versiones posteriores

RESUMEN AMPLIO

En la última sección listamos brevemente todos los servicios del DOS, de manera que los servicios individuales puedan ser encontrados por su número de función. En esta sección hemos ampliado la lista para mostrar los valores de los registros para los parámetros de entrada y salida. Ya que cada versión nueva del DOS introduce unas cuantas funciones que no pueden ser usadas con las versiones anteriores, hemos incluido el número de versión del DOS en esta tabla. Las versiones del DOS están codificadas en la siguiente manera:

- DOS 1 La función puede ser usada con todas las versiones del DOS.
- DOS 2 La función debe ser usada con las versiones DOS 2.00 o superiores.
- DOS 3 La función puede ser usada con versiones DOS 3.00 o superiores.

Servicio	Función (hex)	Registro		Versión
		Entrada	Salida	
Funciones de control de programa				
Terminar: Fin del programa	0	AH = 00		DOS1
Crear segmento de programa	26	AH = 26 DX = dirección de segmento		DOS1
KEEP: Terminar permaneciendo residente avanzado	31	AH = 31 AL = código de retorno DX = dirección de segmento de memoria para liberar	AX = código de retorno	DOS2
Obtener/ajustar control <i>break</i>	33	AH = 33 AL = 00 para obtener AL = 01 para ajustar DL = código si se ajusta	AX = código de retorno DL = estado actual 00 = desactivado 01 = activado	DOS2
EXEC: Cargar/ejecutar un programa	4B	AH = 4B AL = código de subfunción (página 356) DS:DX = puntero de la cadena ASCIIZ ES:BX = puntero del bloque de control	AX = código de retorno	DOS2
Terminar un proceso	4C	AH = 4C AL = código de retorno		DOS2
Obtener el código de retorno de un subprograma	4D	AH = 4D	AX = código de retorno	DOS2
Obtener la dirección del PSP	62	AH = 62	BX = dirección de segmento del PSP	DOS3
Funciones de control de teclado				
Entrada del teclado con eco	1	AH = 01	AL = carácter	DOS1
Entrada directa del teclado sin eco	7	AH = 07	AL = carácter	DOS1
Entrada del teclado sin eco	8	AH = 08	AL = carácter	DOS1
Entrada de teclado con <i>buffer</i>	A	AH = 0A DS:DX = puntero del <i>buffer</i> de entrada		DOS1
Comprobar el estado de la entrada del teclado	B	AH = 0B	AL = FF si hay carácter disponible AL = 00 si no hay carácter disponible	DOS1

Figura 18-4. Un resumen de los servicios del DOS

Servicio	Función (hex)	Registro		Versión
		Entrada	Salida	
<i>Funciones de control de teclado (continuación)</i>				
Borrar el teclado y ejecutar función	C	AH = 0C AL = número de función (1, 6, 7, 8 o A) DL = carácter de salida cuando lo hay	AL = carácter de entrada cuando lo hay	DOS1
<i>Funciones de control de la pantalla</i>				
Salida a la pantalla	2	AH = 02 DL = carácter de salida		DOS1
Escribir cadena en pantalla	9	AH = 09 DS:DX = puntero de la cadena de salida		DOS1
<i>Funciones de entrada/salida de la consola</i>				
Entrada/salida directa teclado/pantalla	6	AH = 06 DL = si FF carácter de salida; si 00-FE petición de entrada	AL = carácter de entrada	DOS1
<i>Funciones de entrada/salida diversas</i>				
Entrada serie	3	AH = 03	AL = carácter de entrada	DOS1
Salida serie	4	AH = 04 DL = carácter de salida		DOS1
Salida impresora	5	AH = 05 DL = carácter de salida		DOS1
<i>Funciones del disco</i>				
Rearrancar el disco	D	AH = 0D		DOS1
Seleccionar la unidad activa	E	AH = 0E DL = identificador de la unidad	AL = cálculo de la unidad	DOS1
Comunicar la unidad activa	19	AH = 19	AL = código de la unidad activa	DOS1
Fijar el área de transferencia del disco (DTA)	1A	AH = 1A DS:DX = puntero del DTA		DOS1
Obtener información FAT de la unidad activa	1B	AH = 1B	AL = sectores por <i>cluster</i> CX = bytes por sector DX = número de unidades de <i>cluster</i> DS:DX = puntero del byte identificador de la FAT	DOS1

Figura 18-4. Un resumen de los servicios del DOS

Servicio	Función (hex)	Registro		Versión
		Entrada	Salida	
<i>Funciones del disco (continuación)</i>				
Obtener información FAT de cualquier unidad	1C	AL = 1C DL = identificador de la unidad	CX = bytes por sector AL = sectores por <i>cluster</i> DX = número de <i>cluster</i> DS:DX = puntero del byte identificador de la FAT	DOS1
Activar/desactivar la verificación de escritura en disco	2E	AH = 2E AL = conmutador de verificación 00 = desactivado 01 = activado DL = 00		DOS1
Obtener la dirección de la DTA	2F	AH = 2F	AX = código de retorno ES:BX = puntero de la DTA	DOS1
Obtener el espacio libre en el disco	36	AH = 36 DL = código de la unidad	AX = sectores por <i>cluster</i> BX = <i>cluster</i> disponibles CX = bytes por sector DX = total de <i>cluster</i>	DOS2
Obtener el estado de verificación	54	AH = 54	AL = estado de verificación: 00 = desactivado; 01 = activado	DOS2
<i>Funciones de entrada/salida en fichero</i>				
Abrir un fichero	0F	AH = 0F DS:DX = puntero del FCB	AL = código de retorno	DOS1
Cerrar un fichero	10	AH = 10 DS:DX = puntero del FCB	AL = código de retorno	DOS1
Buscar un primer fichero	11	AH = 11 DS:DX = puntero del FCB	AL = código de retorno	DOS1
Buscar el siguiente fichero	12	AH = 12 DS:DX = puntero de FCB	AL = código de retorno	DOS1
Borrar un fichero	13	AH = 13 DS:DX = puntero del FCB	AL = código de retorno	DOS1
Leer un registro de fichero secuencial	14	AH = 14 DS:DX puntero de FCB	AL = código de retorno	DOS1
Escribir un registro de fichero secuencial	15	AH = 15 DS:DX puntero de FCB	AL = código de retorno	DOS1
Crear un fichero	16	AH = 16 DS:DX = puntero del FCB	AL = código de retorno	DOS1

Figura 18-4. Un resumen de los servicios del DOS

Servicio	Función (hex)	Registro		Versión
		Entrada	Salida	
Funciones de entrada/salida en fichero (continuación)				
Renombrar un fichero	17	AH = 17 DS:DX = puntero del FCB	AL = código de retorno	DOS1
Leer un registro de un fichero aleatorio	21	AH = 21 DS:DX = puntero del FCB	AL = código de retorno	DOS1
Escribir un registro en un fichero aleatorio	22	AH = 22 DS:DX = puntero del FCB	AL = código de retorno	DOS1
Obtener el tamaño de un fichero	23	AH = 23 DS:DX = puntero del FCB	AL = código de retorno	DOS1
Fijar el campo del registro aleatorio	24	AH = 24 DS:DX = puntero del FCB		DOS1
Leer registros de un fichero aleatorio	27	AH = 27 CX = número de registros DS:DX = puntero del FCB	AL = código de retorno CX = número de registros leídos realmente	DOS1
Escribir registros de un fichero aleatorio	28	AH = 28 CX = número de registros DS:DX = puntero del FCB	CX = número de registros escritos realmente	DOS1
Analizar el nombre de un fichero		AH = 29 DS:SI = puntero de la línea de comando ES:DI = puntero del FCB AL = bits de control de análisis (véase página 321)	DS:SI = puntero del siguiente lugar en la línea de comando AL = código de retorno ES:DI = puntero del FCB	DOS1
CREAT: crear un fichero	3C	AH = 3C CX = atributo del fichero DS:DX = puntero de la cadena ASCIIZ	AX = número del fichero o código de retorno	DOS2
Abrir un fichero	3D	AH = 3D AL = atributo del fichero DS:DX = puntero de la cadena ASCIIZ	AX = número del fichero o código de retorno	DOS2
Cerrar el número de un fichero	3E	AH = 3E BX = número del fichero	AX = código de retorno	DOS2
Leer desde fichero o dispositivo	3F	AH = 3F BX = número del fichero CX = bytes a leer DS:DX = puntero del <i>buffer</i> de la DTA	AX = número de bytes leídos o códigos de retorno	DOS2

Figura 18-4. Un resumen de los servicios del DOS

Servicio	Función (hex)	Registro		Versión
		Entrada	Salida	
<i>Funciones de entrada/salida en fichero (continuación)</i>				
Escribir en fichero o dispositivo	40	AH = 40 BX = número del fichero CX = bytes a leer DS:DX = puntero del <i>buffer</i> del DTA	AX = número de bytes escritos o código de retorno	DOS2
Borrar un fichero	41	AH = 41 DS:DX = puntero de la cadena ASCIIZ	AX = código de retorno	DOS2
Mover el puntero del fichero	42	AH = 42 AL = código de método CX:DX = desplazamiento	AX = código de retorno si CF está activado DS:AX = nueva posición del puntero si CF no está activado	DOS2
CHMOD: Obtener/modificar los atributos del fichero	43	AH = 43 AL = código de obtener/modificar 00 = obtener de CX 01 = modificar como en CX CX = atributo si se modifica DS:DX = puntero de la cadena ASCIIZ	AX = código de retorno CX = atributo si se usa obtener	DOS2
DUP: Duplicar el número del fichero	45	AH = 45 BX = número del fichero	AX = número del fichero o código de retorno	DOS2
CDUP: Duplicación forzada del número	46	AH = 46 BX = número de fichero existente CX = segundo número del fichero	AX = código de retorno CX = número del fichero	DOS2
FIND FIRST: Comenzar la búsqueda de ficheros	4E	AH = 4E CX = atributo para buscar DS:DX = puntero de la cadena ASCIIZ	AX = código de retorno	DOS2
Continuar la búsqueda de ficheros	4F	AH = 4F DS:DX = información de la llamada FIND FIRST	AX = código de retrno	DOS2
Renombrar un fichero	56	AH = 56 DS:DX = puntero de la cadena ASCIIZ (nombre antiguo) ES:DI = puntero de la cadena ASCIIZ (nuevo nombre)	AX = código de retorno ampliado	DOS2

Figura 18-4. Un resumen de los servicios del DOS

Servicio	Función (hex)	Registro		Versión
		Entrada	Salida	
Funciones de entrada/salida en fichero (continuación)				
Obtener un código de error extendido	59	AH = 59 BX = 0000	AX = código de error BH = clase de error BL = acción CH = localización (véase página 362)	DOS3
Crear un fichero transitorio	5A	AH = 5A DS:DX = puntero del nombre de ruta del fichero CX = atributo del fichero	AX = código de error si CF está activada DS:DX = camino de ruta si CF no está ajustado	DOS3
Crear un nuevo fichero	5B	AH = 5B DS:DX = puntero del nombre de ruta del fichero CX = atributo del fichero	AX = número o código de error	DOS3
Bloquear/desbloquear el acceso a un fichero	5C	AH = 5C AL = 0 (bloqueado) o 1 (desbloqueado) BX = número del fichero CX:DX = dir. relativa de la zona a bloquear SI:DI = cantidad para bloquear	AX = código de error	DOS3
Funciones del directorio				
MKDIR: Crear un directorio	39	AH = 39 DS:DX = puntero de la cadena ASCIIZ	AX = código de retorno	DOS2
RMDIR: Borrar un directorio	3A	AH = 3A DS:DX = puntero de la cadena ASCIIZ	AX = código de retorno	DOS2
CHDIR: Cambiar el directorio activo	3B	AH = 3B DS:DX = puntero de la cadena ASCIIZ	AX = código de retorno	DOS2
Obtener el directorio activo	47	AH = 47 DL = identificador de la unidad DS:SI = puntero del área de datos	AX = código de retorno si CF está activado DS:SI = nombre de ruta si CF no está activado	DOS2
Funciones de fecha/hora				
Obtener la fecha	2A	AH = 2A	AL = día de la semana: 0 = domingo; 6 = sábado CX = año (1980-2099) DH = mes DL = día	

Figura 18-4. Un resumen de los servicios del DOS

Servicio	Función (hex)	Registro		Versión
		Entrada	Salida	
Funciones de fecha/hora (continuación)				
Fijar la fecha	2B	AH = 2B CX = año (1980-2099) DH = mes DL = día	AL = código de retorno	DOS1
Obtener la hora	2C	AH = 2C	CL = minutos CH = horas DL = centésimas de segundo DH = segundos	DOS1
Fijar la hora	2D	AH = 2D CL = minutos CH = horas DL = centésimas de segundo DH = segundos	AL = código de retorno	DOS1
Obtener/ajustar la fecha y la hora de un fichero	57	AH = 57 AL = indicador obtener/modificar 00 = obtener; 01 = modificar BX = título del fichero CX = hora, si AL = 01 DX = fecha, si AL = 01	AX = código de retorno ampliado estándar CX = hora, si AL = 00 DX = fecha si AL = 00	DOS2
Funciones variadas				
Ajustar el vector de interrupción	25	AH = 25 AL = número de interrupción DS:DX = puntero del manipulador de interrupción		DOS1
Obtener el número de versión DOS	30	AH = 30	AL = número principal de versión AH = número secundario de versión BX = 0000 CX = 0000	DS1
Obtener el vector de interrupción	35	AH = 35 AL = número de interrupción	AX = código de retorno ES:BX = vector de interrupción	DOS2
Obtener información dependiente del país	38	AH = 38 DS:DX = puntero del <i>buffer</i> de 32 bytes AL = 00 para información estándar o AL = código de país o AL = FF si el código del país > 255 BX = código del país si AL = FF	AX = código de retorno, si CF está activado DS:DX = información (véase página 344) BX = código del país	DOS2

Figura 18-4. Un resumen de los servicios del DOS

Servicio	Función (hex)	Registro		Versión
		Entrada	Salida	
<i>Funciones variadas (continuación)</i>				
IOCTL: control de E/S de dispositivos	44	AH = 44 AL = código de subfunción (véase página 353) BL = número de la unidad BX = número del fichero CX = número de bytes a leer o a escribir	AX = código de retorno si CF está activado AX = número de bytes leídos o escritos si CF no está activado CF = bits de control de datos	DOS2
<i>Funciones de memoria</i>				
Asignar memoria	48	AH = 48 BX = memoria solicitada, en párrafos	AX = dirección de segmento o código de retorno si CF está activando BX = máximo tamaño de bloque disponible	OS2
Liberar memoria asignada	49	AH = 49 ES = bloque de segmento para devolver	AX = código de retorno	DOS2
SETBLOCK: Modificar un bloque de memoria asignada	4A	AH = 4A BX = tamaño requerido en párrafos ES = dirección de segmento del bloque	AX = código de retorno BX = máximo tamaño posible	DOS2

Figura 18-4. Un resumen de los servicios del DOS

La construcción de un programa

Interfaces de programas 380

Combinación de módulos de programas 383

Paso 1: Escritura del código fuente 384

Paso 2: Traducción del código fuente 384

Paso 3: Montaje de programas 384

Paso 4: Conversión de formatos de ficheros 385

Paso 5: Creación de librerías de código objeto 386

Uso del programa LINK 388

Montaje de un programa simple 388

Montaje de un programa con la librería del compilador 388

Combinación de programas 389

Como hemos indicado a lo largo de este libro, el mejor camino para programar la familia PC es escribir la mayor parte de nuestros programas en lenguaje de alto nivel (como BASIC, el Pascal o el C) y, cuando sea necesario, usar los servicios del DOS, o el BIOS, para cualquier función, para la que los lenguajes de alto nivel no estén preparados. En ocasiones, también es posible querer crear las rutinas en lenguaje ensamblador para realizar tareas especializadas que no sean posibles de realizar desde nuestro lenguaje de programación, o los servicios del sistema.

Cuando creamos programas dentro de los confines de un único lenguaje de programación, realmente no necesitamos saber sobre un lenguaje nada más que lo que podamos encontrar en los manuales que van con él. Sin embargo, si es necesario traspasar los límites del lenguaje para acceder a algunas de las rutinas del sistema, o quizá para enlazar con un programa escrito en un lenguaje diferente, será necesario profundizar en los aspectos técnicos, tanto del DOS como de los lenguajes de programación —del DOS para aprender cómo montar (*link*) programas, y de los lenguajes de programación para averiguar los requerimientos de los programas de interfaz, que permiten a los distintos lenguajes comunicarse los unos con otros.

Este capítulo presenta algunas consideraciones globales que se aplican al uso avanzado de la mayoría de los lenguajes de programación; esto es, construir programas de interfaces y montar (*linking*) programas con la utilidad LINK. El capítulo siguiente, el capítulo 20, cubre cinco tipos específicos de lenguajes de programación, y los traductores de lenguaje. En ese capítulo se señalarán algunas de las características técnicas de los cinco lenguajes que deben ser consideradas siempre que se enlacen con subrutinas en ensamblador.

INTERFACES DE PROGRAMAS

Un interfaz de programa es un nivel de código en ensamblador que hace posible que un programa en lenguaje de alto nivel se comunique con una subrutina en ensamblador. Hay dos partes clave en un interfaz de programas: el interfaz de control y el de datos.

El **interfaz de control** realiza las llamadas y los retornos; es decir, pasa el control de un módulo a otro, y de nuevo al primero. Este interfaz es simple, pero puede generar grandes complicaciones si no se realiza correctamente.

El **interfaz de datos** permite a los dos elementos enlazados tratar y comprender correctamente datos comunes. El éxito se consigue conociendo cómo cada programa unido por el interfaz encuentra y trabaja los datos, y de cómo se formatean los datos para que cada

subprograma pueda interpretarlos de la misma forma. Se tratarán estos temas con más detalle en el siguiente capítulo.

El diseño de interfaces de programas es sólo una parte del proceso de montaje de programas. Los tres elementos del programa —el programa llamador, la subrutina llamada y el interfaz— deben cumplir lo siguiente para trabajar juntos correctamente:

El programa debe ser capaz de encontrar su camino a la subrutina. En el sistema basado en el 8088 de la familia estándar PC, una subrutina puede ser llamada de una de dos maneras: mediante una interrupción, o mediante una instrucción CALL. Como ya se sabe los servicios del DOS y la BIOS, se llaman mediante interrupciones usando la instrucción INT, y las direcciones de las rutinas de servicio están implícitas en el número de interrupción. Los programas y subrutinas en lenguaje ensamblador se llaman desde nuestro lenguaje de programación con la instrucción CALL. Las direcciones están asociadas con los nombres de programa, o subrutina, y se establecen durante el proceso de montaje (● siga leyendo más sobre el montaje).

Hay dos tipos de instrucciones CALL: la NEAR CALL (llamada cercana) y la FAR CALL (llamada lejana). La NEAR CALL localiza una subrutina dentro del segmento de código (CS) de 64K actual y no requiere que se cambie el registro CS. Por el contrario, la FAR CALL localiza una subrutina fuera del CS actual usando una dirección segmentada completa en la instrucción CALL (la cual cambia el valor de CS). Algunos lenguajes usan ambas instrucciones y otros usan sólo una.

La subrutina debe saber qué hacer cuando se termine. A menudo una subrutina volverá al programa principal usando una instrucción NEAR RET, o FAR RET, pero hay otras opciones —por ejemplo, se puede querer terminar el programa y volver al DOS desde la subrutina. La instrucción RET de la subrutina hace algo más que volver al programa principal; también limpia el *stack*, como se verá pronto.

La subrutina debe saber qué sistema de apoyo está obteniendo del programa principal. Este sistema de apoyo implica tales cosas como la forma en que están ajustados los registros de segmento y si hay un *stack* que puede ser usado. En general, los registros de segmento son como deben ser: CS tiene el segmento de código correcto; DS apunta a la situación de los datos del programa principal, y SS y SP están ajustados al *stack* del usuario.

La subrutina llamada puede, generalmente, continuar usando el *stack* del usuario, pero no hay una manera práctica de saber cuánto espacio de trabajo hay disponible. Si su necesidad es razonable

—digamos, menos de 64 bytes—, el espacio de *stack* del usuario debe ser adecuado. Sin embargo, si se necesitase más, la subrutina reserva su propio espacio de datos en la memoria.

Si el programa necesita pasar información (parámetros) a la subrutina, tanto el programa como la subrutina deben saber cuántos parámetros hay, dónde estarán colocados y si necesitan o no ser cambiados y devueltos. Comúnmente, los programas y las subrutinas trabajan con un número fijo de parámetros, aunque hay maneras de manipular un número variable de parámetros en algunos lenguajes. Los parámetros suelen ser pasados a través del *stack*, bien directa o indirectamente. El método directo pasa el valor real del parámetro a través del *stack*; el método indirecto pasa la dirección del valor a través del *stack*. En cada caso, el programa llamado debe saber cuál de los métodos está siendo usado.

El método de pasar el parámetro depende, ante todo, del lenguaje; algunos lenguajes no pueden situar valores en el *stack*, sino sólo direcciones. Con aquellos lenguajes que pueden manipular tanto direcciones como valores, se tiene mucha más libertad para decidir qué método usar, y el método que se utilice determina cómo son tratados los parámetros cuando se pasan de un programa a otro. Por ejemplo, si se quiere evitar que los parámetros del usuario sean cambiados por la subrutina llamada, será conveniente pasar el valor al *stack* para estar seguros de mantener una copia de éste. Pero si se quiere que los parámetros del usuario sean cambiados por la subrutina llamada, se debe enviar la dirección del valor por vía del *stack*, así que la subrutina cambiará el valor del parámetro modificando lo que está almacenado en la dirección especificada.

El paso de parámetros es la parte más complicada de una rutina de interfaz, debido a las diferentes maneras con que los lenguajes de programación tratan los datos y la información del *stack*. A causa de su complejidad y variabilidad de un lenguaje a otro, éste será el tema principal que se tratará en las comparaciones de lenguajes del siguiente capítulo.

La subrutina debe conservar cierta información. Aunque los requerimientos pueden variar en diferentes situaciones, hay unas reglas básicas que indican qué información debe ser conservada y qué puede y qué no puede hacerse cuando se llama a una subrutina. Se han incluido algunas indicaciones aquí, pero encontrará un poco más en el capítulo 3, especialmente en la página 43.

Las interrupciones pueden ser suspendidas, aunque no suele ser una buena idea, excepto brevemente, cuando se cambian los registros de segmento; deben ser reactivadas antes de volver (véase la página 55).

Si se modifica cualquier registro de segmento, el valor original debe ser conservado en el *stack*. Otro registro importante de conservar, en la mayoría de las circunstancias, es el puntero de base (BP), ya que es usado a menudo para controlar la situación de los parámetros en el *stack*. Por convención, un programa principal no espera que los valores de sus registros de trabajo se conserven, así que todos los registros de trabajo, AX, BX, CX, DI y SI, pueden cambiarse libremente, así como los indicadores. El *stack* del usuario tiene que ser también conservado.

El *stack* debe ser limpiado después de que la subrutina haya terminado. Hay cuatro cosas que pueden estar ocupando indebidamente el *stack* cuando termina una subrutina: algunos parámetros, la dirección de regreso de la instrucción CALL, valores de registros salvados antes de CALL y, finalmente, algún almacenamiento de trabajo de la subrutina.

Tres de estos datos sobrantes no son un problema: se espera que las subrutinas retiren sus propios almacenamientos de trabajo del *stack*; los registros salvados son retirados mediante instrucciones POP, y la dirección de regreso se retira mediante la instrucción RET. Son los parámetros los que suele complicar el proceso de borrado, porque el método de retirarlos varía en diferentes lenguajes. Algunos lenguajes esperan que la subrutina retire los parámetros especificando en la instrucción RET el número de bytes que hay que retirar del *stack*. Otros lenguajes esperan que el usuario los retire. Se señalarán estas diferencias cuando se discutan los lenguajes con detalle en el capítulo 20.

Con todos estos elementos de diseño de programas en la mente, retrocedamos un poco y veamos cómo trabaja el proceso completo —desde crear un programa o una subrutina, hasta combinarlo con otros.

COMBINACION DE MODULOS DE UN PROGRAMA

En esta sección se van a cubrir las reglas generales para unir piezas de programa, o para combinar los módulos de un programa. Se estará usando un procedimiento de programación estándar del DOS, que se aplica a todos los lenguajes de programación usados como ejemplos en el siguiente capítulo (excepto por el BASIC interpretado, el cual siempre parece ser un caso especial). Primero, veamos los pasos que están implicados en la creación de un programa.

Paso 1: Escritura del código fuente

Para comenzar, tenemos que escribir nuestro programa usando los comandos y la sintaxis de nuestro lenguaje de programación. Esta forma del programa es conocida como código fuente. Para los lenguajes de programación que usan las convenciones DOS estándar, el código fuente debe estar en la forma de un fichero de texto ASCII (véase apéndice C, página 456). El BASIC interpretado no usa normalmente el formato de ficheros de texto ASCII para sus ficheros fuentes, pero puede hacerlo. (Para crear ficheros de texto ASCII con el intérprete del BASIC, se usa la opción A del comando SAVE.)

Por convención, los ficheros en código fuente tienen una ampliación del nombre del fichero que refleja el nombre del lenguaje de programación usado, como BAS o C.

Paso 2: Traducción del código fuente

Los ficheros de código fuente se entregan a los traductores de lenguaje, llamados compiladores, para los lenguajes de alto nivel o ensambladores para el lenguaje ensamblador. (De nuevo, el intérprete del BASIC es un caso especial que no se considera aquí.) El traductor (compilador o ensamblador) convierte el código fuente en instrucciones de lenguaje máquina, pero no las deja en una forma que estén listas para ser ejecutadas. En vez de ello, los compiladores y ensambladores ponen sus resultados en una forma conocida como código objeto. El formato de código objeto está diseñado con un propósito particular en mente: combinar módulos de código objeto separados en un programa único más grande. Los ficheros en código objeto, por convención, tienen una extensión de nombre de fichero de OBJ.

Paso 3: Montaje de programas

El siguiente paso básico es unir los programas. El montador, conocido como LINK en el DOS, realiza dos tareas principales: combina módulos objeto (como sea necesario), haciendo todas las conexiones necesarias entre ellos, y convierte los módulos de un formato de código objeto en un programa cargable en el formato .EXE.

La combinación real, o montaje, de módulos de programa para crear un fichero .EXE es el aspecto más importante de esta explicación. Se tratará de nuevo en este capítulo, después de tratar otros dos pasos que están implicados en la preparación de programas.

Hasta ahora se han mencionado los tres pasos principales de la preparación de un programa: la escritura del programa para producir el código fuente, la compilación, o el ensamblado, del programa para producir el código objeto, y el montaje para producir un programa ejecutable. Hay otros dos pasos relacionados con el proceso: la conversión del formato .EXE creado por el programa LINK al formato .COM, y el uso de librerías objeto para almacenar un gran número de módulos.

Paso 4: Conversión de formatos de ficheros

Los programas que están almacenados en un disco en formato de fichero .EXE no están completamente listos para ser ejecutados. Cuando son cargados desde el disco en la memoria, el DOS realiza unas pocas operaciones de último momento para preparar el programa para su ejecución. Estas operaciones hacen cosas, como decirle al programa dónde está situado en memoria, calcular su tamaño y reservar un *stack* para su uso. Pero si las convenciones de operación de un programa son suficientemente simples, y si las etapas iniciales son suficientemente inteligentes, esta preparación de carga puede ser hecha de antemano, convirtiendo el fichero a un formato de fichero .COM.

Un fichero .COM es una imagen exacta del programa tal y como aparecerá en la memoria del ordenador. Mientras el DOS debe hacer un gran trabajo para preparar un programa de formato .EXE, sólo tiene que hacer dos cosas para un programa de formato .COM: crear el prefijo de segmento de programa (véase página 292) y dar valores a los registros de segmento.

Se usa el programa EXE2BIN para convertir un fichero .EXE en un formato de fichero .COM. No todos los programas pueden ser convertidos al formato .COM. Aquellos programas que estén cualificados, podemos libremente convertirlos, o dejarlos con el formato .EXE. Ambas formas son funcionalmente lo mismo, pero el formato .COM es más compacto y se carga bastante más deprisa.

Normalmente, nosotros no tenemos elección sobre si un fichero .EXE puede o no ser convertido. Todo depende de las convenciones establecidas por el lenguaje de programación. La mayoría de los lenguajes de programación de alto nivel necesita un apoyo extra que proporciona el formato .EXE y, por tanto, no pueden ser convertidos (véase el «Manual de Referencia Técnica de IBM», una lista de las operaciones de apoyo .EXE). En los programas que estén creados puramente en lenguaje ensamblador, se tiene un control completo y se puede elegir uno u otro formato. Para programas BASIC y Pascal

compilados y para programas en muchos otros lenguajes compilados, el formato .EXE es obligatorio. Los programas escritos en lenguaje C (usando el compilador Lattice/Microsoft C), que es como si estuviera entre el lenguaje ensamblador y un lenguaje de alto nivel, pueden ir de cualquiera de las maneras.

Es posible averiguar de forma muy simple y segura si un programa puede ser convertido del formato .EXE al formato .COM: intentándolo. Si funciona, funciona. Si el EXE2BIN, o el LOCATE, dice que no puede hacerse, entonces no puede hacerse.

Paso 5: Creación de librerías de código objeto

La mayoría de los lenguajes de programación de alto nivel hace uso de docenas de subrutinas preparadas que apoyan la operación de nuestros programas. Naturalmente, estas subrutinas están en forma de código objeto. Sin embargo, no es conveniente tener docenas de estos ficheros objeto en nuestros discos. Tampoco es conveniente tener que determinar nosotros mismos cuáles necesitan ser combinados con nuestros propios ficheros objeto. Para solucionar este problema, hay librerías objeto, que son colecciones de código objeto agrupadas en un fichero. Por convenio, las librerías tienen la extensión de nombre de fichero LIB.

La mayoría de los lenguajes de alto nivel viene con una librería lista para usar, con subrutinas de apoyo estándar. Ocasionalmente, un compilador tendrá varias librerías que proporcionen diferentes versiones de las rutinas estándar. Por ejemplo, pueden venir con rutinas de cálculo en punto flotante que pueden, o no, hacer uso del coprocesador matemático 8087.

El montaje es capaz de buscar en una librería para encontrar y usar las subrutinas que necesita para completar un programa. Sin este mecanismo de librería nos enfrentaríamos con la desagradable tarea de decirle al montador qué ficheros objeto son necesarios. Si olvidáramos alguno, el montaje fallaría, si incluyésemos alguno que no fuese necesario, nuestro programa crecería sin ningún sentido. El uso de una librería nos permite evitar estos problemas.

Las librerías objeto son principalmente usadas para apoyar a los compiladores, de una manera que está completamente fuera de nuestras manos. No creamos ni modificamos las librerías y ni siquiera seleccionamos directamente lo que será usado de una librería. En vez de ello, seleccionamos indirectamente los ficheros de una librería, usando características particulares de nuestro lenguaje de programación, o directamente usando el programa LIB.

□ *NOTA: El programa LIB no es parte de todas las versiones del DOS, aunque debería serlo. Las versiones IBM del DOS desde 1.00 hasta la 3.10 no incluyen el LIB. El LIB está incluido en algunas versiones del DOS que no son de IBM, y acompaña a algunos de los compiladores y ensambladores, pero no a todos. El conseguir una copia de trabajo del LIB es una proposición algo así como «coge lo que puedas».*

Si se tiene una copia del LIB, puede usarse para tres propósitos principales: simplemente para explorar los contenidos de las librerías existentes (que puede ser una experiencia muy instructiva), para reemplazar selectivamente módulos de librerías existentes si se quiere modificar o mejorar la librería que viene con un lenguaje de programación, y para crear librerías propias.

La documentación para el LIB en el manual del DOS explicará completamente su operación. Pero sólo para darle una idea de las formas en que se puede usar el LIB, hemos incluido unos pocos ejemplos para probar. Para crear una nueva librería llamada TESTLIB, introduzca este comando:

LIB TESTLIB;

Para listar los contenidos de una librería existente, dirigiendo la lista a la impresora LPT1: (o cualquier fichero, o a la pantalla), introduzca:

LIB TESTLIB,LPT1;

Para añadir el módulo X.OBJ a una librería, introduzca:

LIB TESTLIB+X;

Para reemplazar el módulo existente por una nueva versión, introduzca:

LIB TESTLIB-X+X;

Con las últimas versiones del LIB, escriba $- + X$ en vez de $- X + X$. Para extraer un módulo para desensamblar, o para otro uso distinto, introduzca:

LIB TESTLIB*X;

Nuestros programas suelen estar compuestos por un número de subrutinas. El encontrar o no apropiado el beneficiarse de los servicios del programa LIB depende de la decisión básica que se tome sobre la manera de organizar estas subrutinas. Si se prefiere combinar el código fuente de las subrutinas en un fichero fuente, lo que significa que quedarán compiladas juntas, entonces se tendrá poca

necesidad del LIB. Por otro lado, si se prefiere compilar las subrutinas por separado, lo que produce ficheros objeto diferentes, entonces el LIB realiza el trabajo necesario: agrupa y organiza los ficheros objeto. Personalmente no doy recomendaciones sobre un estilo de operación u otro; yo he usado ambos en mi programación del PC y he encontrado a cada uno práctico y efectivo. Es principalmente un problema de gustos, aunque su elección puede tener ciertas consecuencias si se programa en Pascal (● véase página 412).

USO DEL PROGRAMA LINK

Estamos ahora preparados para volver sobre el tema de la combinación de módulos de programa y el uso del programa LINK. La documentación para el LINK, en el «Manual de Referencia Técnica del DOS», de IBM, explica totalmente su operación, incluyendo la complejidad de sus conmutadores de control. Aquí se resumirán las operaciones más comunes y útiles, particularmente donde estén relacionadas con los lenguajes de programación, que se tratan en el siguiente capítulo.

Sólo para darle una información de fondo, el programa LINK toma cuatro parámetros, que pueden escribirse así:

`LINK 1,2,3,4;`

El primer parámetro, el 1, representa una lista explícita de módulos objeto (como PROG1+PROG2+PROG3); el 2 representa el nombre del programa terminado; el 3 representa adónde deben ser enviados los mensajes de salida del montador (como la impresora o la pantalla), y el 4 representa una lista de librerías que son usadas (como BASCOM+MILIB).

Montaje de un programa simple

He aquí unos pocos ejemplos prácticos. Para comenzar, considérese un programa completamente autocontenido, como el programa BEEP que se muestra en la página 398. Para acoplarlo, simplemente se pulsa la secuencia de teclas.

`LINK BEEP;`

Montar un único programa como éste crea un fichero .EXE.

Montaje de un programa con la librería del compilador

A continuación se considera la que es seguramente la circunstancia de acoplamiento más común. Se supone que se ha compilado un

programa en lenguaje de alto nivel, como en BASIC compilado, y es necesario montarlo con su librería estándar. En este caso, no se están usando otros interfaces ni otros módulos que los que hemos creado. El nombre de nuestros programas es X y el nombre de la librería del compilador es BASCOM.LIB. Así es como se escribirá el comando LINK:

```
LINK X,,BASCOM;
```

Generalmente, un compilador genera un módulo objeto que va a la librería a buscar todo lo que sea necesario. En el caso del Lattice/Microsoft C (● que se trata en la página 421), hay una rutina estándar de arranque, llamada módulo de prefijo, que debe ser acoplada al frente de nuestro programa. Suponiendo que el módulo de prefijo se llama C, nuestro programa X, y la librería se llama MC, se acoplarían de esta forma:

```
LINK C+X,X,,MC
```

Deben observarse dos nuevos términos en este ejemplo. Primero se pide explícitamente al acoplador que combine dos ficheros objeto, C.OBJ y X.OBJ (que es el programa que nosotros compilamos). Segundo, explícitamente damos un nombre al programa .EXE terminado, llamándolo X. Hay que darle este nombre, o el montaje usará el nombre del primer módulo objeto, que es C. Si dejamos que esto ocurra, todos nuestros programas terminarán con el mismo nombre, C, en vez de con su nombre individual (en este caso X). Es posible darle al programa terminado cualquier nombre que se quiera, incluyendo un nombre sin ninguna relación con los nombres de módulos objeto. En este ejemplo, sin embargo, se ha hecho lo normal, que es darle a nuestro programa terminado X.EXE el mismo nombre que a nuestro fichero objeto X.OBJ.

Combinación de programas

Ahora estamos listos para ilustrar cómo combinar módulos de programas con el montaje. Primero, se considera una situación en la que se haya hecho uso de una librería objeto privada, para conservar nuestros interfaces de lenguaje ensamblador o nuestras subrutinas compiladas independientemente, o ambas cosas. Así es cómo se acoplaría tal librería usando el Pascal. El programa se llama X, y nuestra librería se llama NUESTRA.LIB:

```
LINK X,,PASCAL+NUESTRA.LIB
```

A continuación consideraremos el caso en donde no se tiene librería objeto. En vez de ello, simplemente se quieren combinar dos ficheros objeto: uno de nuestro compilador de lenguaje de alto nivel, y el otro en ensamblador que haga de interfaz con el DOS y la BIOS. Esta es la forma de montar tal programa. En este ejemplo, el lenguaje es Pascal, el programa se llama X y el interfaz en lenguaje ensamblador se llama INTFAZ.

```
LINK X+INTFAZ,,PASCAL
```

Hay, por supuesto, infinidad de variaciones de cómo pueden ser combinados los módulos de programa. Sin embargo, estos ejemplos básicos deben proporcionarle la información necesaria. Armado con la información general de este capítulo, debe estar preparado para seguir adelante con el siguiente, y aprender algo sobre cómo trabajan los lenguajes de programación.

Los lenguajes de programación

Lenguajes específicos 393

Lenguaje ensamblador 394

Organización lógica 395

Aprendizaje sobre convenciones del interfaz 396

Escritura y montaje de programas en ensamblador 398

BASIC interpretado y compilado 399

Formatos de datos en BASIC 399

Interfaces en ensamblador con BASIC interpretado 405

Interfaces en ensamblador con BASIC compilado 408

Pascal 412

Formatos de datos de Pascal 413

Interfaces de ensamblador con Pascal 418

El lenguaje C 421

Formatos de datos en C 422

Interfaces de ensamblador con C 424

Un comentario final 427

En el último capítulo se tratan brevemente los principios generales de la construcción de los módulos de programa y su montaje, usando el programa LINK. En este capítulo se van a tratar los lenguajes de programación que se usan para construir los módulos de programa, particularmente aquellos aspectos de los lenguajes con los que es necesario estar familiarizado si se piensa montar los módulos y subrutinas de lenguaje ensamblador.

El título de este capítulo implica que se van a tratar los lenguajes de programación en general, pero éste no es realmente el caso. Está bien comentar cualquier tema en abstracto, pero, cuando se quiere realmente tener algo hecho, hay que ir a lo específico. Si se quieren cargar programas de ordenador, hay que trabajar con un lenguaje de programación específico, y un lenguaje de programación es mucho más específico que lo que mucha gente cree.

Antes de nada, no existe un lenguaje de programación genérico. Solamente se pueden crear programas de trabajo, con un compilador o con un ensamblador, para un lenguaje de programación que esté diseñado para una máquina particular. Aunque a los expertos académicos de ordenadores les gustaría pretender otra cosa. La verdad práctica es que las definiciones generales de lenguajes de programación carecen de muchas de las características especiales que son necesarias para crear programas que trabajen en ordenadores reales. Así, cuando se crea un compilador, o un intérprete, de un lenguaje de programación particular, como el BASIC, para que funcione en un ordenador particular, como el PC, el lenguaje fundamental es alterado y ampliado para adaptarse a las características reales de la máquina. Las alteraciones son, a menudo, bastante significativas, y, en todos los casos, crean un lenguaje de programación que es afín, pero realmente distinto de todos los demás lenguajes de programación que llevan el mismo nombre.

Esta explicación significa que en este capítulo no se tratan, ni se podrían posiblemente tratar, todos los lenguajes de programación del PC que existen ahora o que pueden ser creados en el futuro, ya que cada compilador, en efecto, crea su propio y único lenguaje de programación. El uso de este capítulo para tratar los aspectos generados de los lenguajes principales no serviría a nuestras necesidades. Es necesario seleccionar solamente unos pocos lenguajes, ya que no es posible tratarlos todos. He elegido cuatro lenguajes populares y representativos: el lenguaje ensamblador, el BASIC, el Pascal y el C. Entonces, dentro de estas categorías, he elegido cinco versiones específicas, o realizaciones de estos lenguajes: el IBM Macro Assembler, el IBM BASIC interpretado, el IBM BASIC compilado, el IBM Pascal y el Lattice/Microsoft C.

Mi decisión, al tomar estos lenguajes en particular, está guiada por el pasado y el futuro. La experiencia pasada me ha enseñado qué versiones son de uso más extendido en la familia PC. La preocupación sobre la conexión de nuevo trabajo y nuevos compiladores me ha llevado a creer que es desaconsejable usar lenguajes de programación que no sean compatibles con el LINK estándar del DOS. Desafortunadamente, una gran cantidad de compiladores y ensambladores o no producen códigos de objeto acoplables (por ejemplo, los admirables *cheap assembler* CHASM, el Turbo Pascal de Borland y el Modula-2 de Logitech) o usan un formato objeto que no es compatible con el LINK del DOS (por ejemplo, la familia de lenguajes de Digital Research y el C-86 de Computer Innovation). Personalmente, yo soy bastante conservador en el tema de la unión de compatibilidad de código objeto, y me temo que al alejarnos del estándar establecido en la práctica por el DOS, corremos un serio riesgo de encontrar problemas en el futuro, mientras la familia de ordenadores PC evoluciona.

LENGUAJES ESPECIFICOS

Los cinco lenguajes de programación que he elegido son realmente familias por sí solos. Hay varias versiones de cada uno, y, en muchos casos, están disponibles desde varias fuentes. Afortunadamente, las diferencias entre las versiones son mínimas —lo bastante reducidas como para que no sea necesario pensar en ellas como lenguajes diferentes en el mismo sentido que el BASIC y el Pascal—, pero son lenguajes diferentes.

Lenguaje ensamblador: Nuestra explicación sobre lenguajes ensambladores estará basada en la versión 1.00 del IBM Macro Assembler, creado por Microsoft. Está disponible un número de versiones de Microsoft, de IBM y de otros fabricantes de ordenadores que tienen autorización para usar el ensamblador básico de Microsoft. Estas versiones varían sólo en sus elementos más sofisticados, que no nos importan aquí.

BASIC interpretado. El BASIC interpretado que se tratará en este capítulo ha tomado miles de caras y variaciones menores. Para los usuarios del IBM PC, la versión que se tratará se conoce simplemente como BASIC, o BASICA, y está definida más detenidamente por nombres de versión como C1.10., J1.00., y A2.10, entre otras. Fuera del mundo de IBM, pueden ser conocidas como BASIC, Microsoft BASIC, o GW-BASIC. Aquí no nos importan las diferencias; lo que nos importa son los elementos comunes.

BASIC compilado. Para nuestra explicación del BASIC compilado, servirá de guía la versión 1.00 del compilador de BASIC de IBM. Los principios que se tratarán también se aplicarán a otros compiladores BASIC de Microsoft, como el Busines BASIC.

Pascal. Para el Pascal se usará como base la versión 1.00 de IBM. Los detalles que se traten se aplicarán igualmente bien a la versión 2.00 de IBM o a varias versiones de Microsoft.

El lenguaje C. Para nuestra explicación del C, se usará la versión 1.04 del compilador C de Lattice/Microsoft, creado por Lattice. Están disponibles otras versiones muy similares de Microsoft, Lattice y Lifeboat, que comparten las características que se van a tratar. Este compilador, a propósito, no debe ser confundido con el compilador C de Microsoft (versión 3 y superiores) que fue creado después de que fuese lanzada la versión C de Lattice/Microsoft.

LENGUAJE ENSAMBLADOR

Hay dos tipos de programa en lenguaje ensamblador: la subrutina en lenguaje ensamblador, que es llamada por otros programas que pueden estar escritos en un lenguaje de alto nivel, y el programa en lenguaje ensamblador autosuficiente. Las subrutinas dependen en gran medida del programa que las llame para proporcionar su estructura y apoyo, mientras que los programas autosuficientes en lenguaje ensamblador deben proporcionarse su propia estructura y apoyo, y deben arreglárselas con los problemas operativos con los que se encuentran los programas que se soportan solos. Las subrutinas en ensamblador son relativamente fáciles de construir, mientras que los programas en ensamblador pueden ser bastante complicados. Las subrutinas tienen un interés más inmediato para aquellos que necesiten construir rutinas de interfaz entre lenguajes de alto nivel y algunos de los servicios BIOS o DOS, mientras que los programas autosuficientes suelen ser utilizados por programadores que deben realizar algo que no proporcionan ni el lenguaje de programación convencional ni los servicios del sistema.

En esta breve exposición del lenguaje ensamblador se mostrarán algunas técnicas que le ayudarán a explicarse las convenciones de conexión del lenguaje de alto nivel para sus subrutinas en lenguaje ensamblador. También se dará una guía a través del proceso de creación de un programa autosuficiente en ensamblador. De cualquier modo, no se intentará enseñar cómo usar el lenguaje ensamblador —eso es demasiado largo y complicado—. Si no se es particularmente hábil en lenguaje ensamblador, una manera de aprender es estudiar alguno de los programas fuente disponibles escrito en

ensamblador. Una fuente elegante son los códigos publicados por fabricantes de ordenadores, como los listados de la BIOS que son parte de los manuales de referencia técnica del DOS. Otra fuente, disponible con muchos compiladores, es el listado en ensamblador que se puede pedir que el ordenador produzca. Esto es útil tanto como para saber cómo manipula el compilador los problemas de codificación particulares (que se pueden controlar seleccionando las respuestas apropiadas en el lenguaje de alto nivel), y también para aprender las convenciones de conexión de subrutinas que usa el compilador. Una manera parecida, pero menos útil de aprender algo sobre el lenguaje ensamblador, es cargar un programa existente usando el programa DEBUG, y después usar el comando U (desensamblar) para estudiar las partes del programa. (La sección que sigue enseña cómo realizar una operación similar para curiosear dentro de librerías de programa.) Cada uso de estos métodos pueden ayudarle a aprender diferentes técnicas de programación y trucos. De hecho, son métodos que yo usé para aprender casi todo lo que conozco sobre programación en el lenguaje ensamblador de la familia PC.

Organización lógica

Los elementos de una subrutina en lenguaje ensamblador son fáciles de entender si están dispuestas en el orden en que ocurren. Como puede recordar, la organización lógica fue totalmente explicada en el capítulo 8, donde se describía una rutina de interfaz como cinco partes anidadas.

- Nivel 1: Ensamblador general elevado
- Nivel 2: Ensamblador de subrutina elevado
 - Nivel 3: Código de entrada
 - Nivel 4: Obtención de los parámetros de los datos del usuario
 - Nivel 5: Llamada al servicio ROM-BIOS o DOS
 - Nivel 4: Devolución de los resultados al usuario
- Nivel 3: Código de salida
- Nivel 2: Finalización del ensamblador de subrutina elevado
- Nivel 1: Finalización del ensamblador general elevado

Esta organización básica puede ser seguida por muchas rutinas de interfaz escritas para los servicios del sistema, o por subrutinas de lenguaje ensamblador convencionales, pero la codificación real varía con cada lenguaje de programación.

La herramienta estándar para la creación de subrutinas en ensamblador en la familia PC es el macro ensamblador de Microsoft, llamado MASM, que está disponible en diferentes versiones. Por ejemplo, el ordenador personal de IBM fue introducido con una

versión numerada 1.00, la cual incluía tanto un pequeño ensamblador (sin capacidad para macros) como un macro ensamblador. Excepto para trabajo con lenguaje ensamblador avanzado —lejos de nuestras intenciones— cualquier versión del ensamblador nos servirá.

Aprendizaje sobre convenciones de interfaz

Una vez que se tenga a mano el lenguaje ensamblador, será necesario examinar las convenciones del lenguaje ensamblador y las costumbres de conexión que se apliquen al lenguaje de programación. El conector (o interfaz) en lenguaje ensamblador tendrá que saber cómo obtener el acceso a los parámetros que pasa el programa principal, cómo interpretar el formato de los datos y cómo devolver los parámetros —entre otras cosas—. Si no hay información adecuada de este tipo en la documentación del lenguaje, hay algunas maneras bastante simples de saber los secretos del lenguaje.

Para aprender las convenciones del programa principal y del programa llamado —esto es, para ver ambos lados del programa interfaz—, se puede estudiar los listados en formato ensamblador del compilador, como se ha mencionado antes. También se pueden estudiar las interioridades de las subrutinas del lenguaje ensamblador proporcionadas con el compilador del lenguaje, que puede dar una perspectiva algo diferente de las que puede ser aprendida al estudiar un listado del compilador. Esta técnica no sólo proporciona los detalles de las convenciones de interfaz para rutinas en lenguaje ensamblador, sino que también ofrece ejemplos de programación específicos, que pueden servir como modelos.

Lo primero que se debe hacer es seleccionar una subrutina adecuada. Las subrutinas que son más accesibles para estudiar son, a menudo, aquellas que son parte de la librería que acompaña a nuestro compilador. Generalmente lo más fácil es elegir simplemente una característica del compilador en la que se esté interesado, tal como entrada/salida, control de pantalla, o aritmética, y después averiguar qué subrutinas son llamadas por esa característica.

A continuación, se tiene que mirar los nombres de los módulos de la librería (que pueden no ser los nombres de las subrutinas dentro de esos módulos). Es posible hacer esto usando el programa LIB (tratado en el capítulo 19) para listar los contenidos de la librería. Suponiendo que hay una librería llamada LANG.LIB en el fichero, es posible dirigir el listado de la librería a otro fichero llamado LISTADO, con la siguiente instrucción (también es posible dirigirla a la pantalla, o la impresora):

```
LIB LANG, LISTADO;
```

Examinando el listado de la librería, se encontrará la subrutina que interesa y el nombre del módulo del que forma parte; se supondrá que se llama XMOD. A continuación, se pide al DOS que separe el XMOD de la librería para poder trabajar con él:

```
LIB LANG*XMOD;
```

El operador * le dice al LIB que cree una copia del módulo como un fichero objeto independiente; en este caso, el fichero se llamará XMOD.OBJ.

En este punto se podría intentar curiosear dentro de XMOD.OBJ, pero este fichero contiene extraña información útil para el programa LINK es mejor convertir el XMOD.OBJ en un juego de instrucciones en código máquina puras, usando dos pasos. Primero se monta para convertirlo en un fichero de programa .EXE:

```
LINK XMOD;
```

Esto nos da un fichero llamado XMOD.EXE. (Debe ignorarse cualquier mensaje de error que no sea del *stack* que pueda dar el montaje.) Después, para librarse del exceso de información del formato .EXE se convierte el XMOD.EXE en un fichero .COM:

```
EXE2BIN XMOD.EXE XMOD.COM
```

En este punto se tiene un fichero llamado XMOD.COM, que debería estar constituido por instrucciones de lenguaje máquina puras. Ahora se está listo para usar el DEBUG para convertir las instrucciones en un formato ensamblador legible. Primero se anota el tamaño de XMOD.COM, y después se activa DEBUG, diciéndole que cargue el XMOD.COM:

```
DEBUG XMOD.COM
```

Entonces se pide al DEBUG que convierta el XMOD en un formato legible con el comando U (desensambla) con esta instrucción:

```
U 100 L XXX
```

(XXX es la longitud del fichero en hexadecimal).

Todos estos pasos pueden parecer muy elaborados e incómodos, pero una vez que se hayan aprendido, se pueden realizar rápida y fácilmente, y darán una visión interior de cómo el propio lenguaje de programación usa las rutinas de conexión.

La siguiente sección repetirá los pasos claves de este ejercicio mientras se muestran los mecanismos de creación de un pequeño, pero completo, programa en lenguaje ensamblador.

Escritura y montaje de programas en ensamblador

Para ilustrar el proceso implicado en la escritura y montaje de un programa en ensamblador, se creará un programa increíblemente simple y, sin embargo, útil, que hace sonar un tono en el altavoz del ordenador. Para hacer esto con cualquier ordenador de la familia PC, o con cualquier ordenador DOS, sólo hay que enviar a la pantalla el carácter que activa el pitido, CHR\$(7). En este ejemplo se hace usando el servicio 2 del DOS, que se llama con la interrupción 33. Después se finalizará el programa y se devolverá el control al DOS usando la interrupción 32. Siguiendo este ejemplo, se aprenderá bastante sobre la creación de programas autocontenidos en lenguaje ensamblador. Aquí está el código fuente de este pequeño programa:

```
; programa del DOS para generar un pitido
BEEPSEG  SEGMENT BYTE PUBLIC 'PROG'
          ASSUME  CS:BEEPSEG
BEEP      PROC
          MOV     DL,7           ; carácter de campana
          MOV     AH,2           ; función de salida de carácter
          INT     33             ; interrupción de función DOS
          INT     32             ; interrupción de vuelta al DOS
BEEP      ENDP
BEEPSEG   ENDS
          END
```

Como se ve, el programa sólo tiene cuatro instrucciones; ocupando sólo ocho bytes, se puede ensamblar el programa con esta instrucción:

MASM BEEP;

El comando MASM crea un fichero objeto que está listo para montar. En este caso se montará el programa sin subrutinas, librerías ni otros ficheros objeto; de este modo:

LINK BEEP;

El programa montador generalmente espera encontrar un segmento de *stack* en el programa que monta, pero este programa es muy simple y no lo tiene —una característica clave que hace posible convertirlo en un fichero .COM, como se verá pronto. El montador se quejará, pero se puede ignorar su protesta.

El montaje nos dará un programa ejecutable llamado BEEP.EXE. Es muy común escribir programas en lenguaje ensamblador de tal modo que puedan ser convertidos al formato más compacto .COM. Este programa se convierte usando el comando EXE2BIN:

EXE2BIN BEEP.EXE BEEP.COM

Ahora se tiene el programa de pitidos terminado, y puede ser usado en cualquier ordenador que use el sistema DOS.

Merece la pena parar y anotar lo que le ocurre al tamaño del programa cuando se transforma de una idea a un fichero .COM ejecutable. El código fuente de este programa tiene aproximadamente 378 bytes (dependiendo del espacio usado para los comentarios, etc.). Cuando se ensambla, se puede ver que sólo se habían creado 8 bytes de instrucciones de trabajo en lenguaje máquina. Sin embargo, el fichero objeto, que incluye alguna información para el montador estándar, es de 54 bytes —mucho más pequeño que nuestro fichero fuente, pero mucho más grande que los 8 bytes de instrucción real—. Después del montaje, el fichero objeto de 54 bytes aumenta al cambiar en un fichero .EXE de 520 bytes. Esto se debe a que el formato .EXE contiene un prefijo que describe cómo van a ser cargados los programas, y este prefijo se crea en registros de 512 bytes —en este caso los 512 bytes de cabecera, seguidos de nuestros 8 bytes de instrucciones—. La conversión del programa al formato .COM elimina los 512 bytes de cabecera y se termina con un fichero .COM, que es sólo de 8 bytes de código máquina puro.

BASIC INTERPRETADO Y COMPILADO

Para ser sincero y directo, debo reconocer que no podremos tratar aquí todo lo necesario. El trabajo con el BASIC y la conexión con el BASIC son temas muy, muy complicados —lo bastante complejos para escribir varios libros sólo con ellos—. Francamente, la conexión con el BASIC es un área particularmente confusa, más confusa aún por el número de versiones de BASIC usadas con los distintos modelos de la amplia familia PC. Incluso dentro del tronco hecho por IBM de este árbol familiar hay más versiones del BASIC que dedos tenemos en las manos y en los pies para contarlos.

En esta exposición se resaltarán los asuntos relacionados con la conexión de programas BASIC con rutinas externas. Los dos temas en los que nos centraremos son los formatos de datos del BASIC (que son importantes para las rutinas externas, porque necesitan ser capaces de cambiar datos con el BASIC) y las convenciones de conexión que especifican cómo se comunican el BASIC y el lenguaje ensamblador.

Formatos de datos del BASIC

El BASIC usa cuatro formatos de datos: enteros, cadenas de longitud variable y números en coma flotante, en las formas larga y

corta, conocidas en la terminología del BASIC como números de simple precisión y de doble precisión. Las variables del BASIC pueden darse explícitamente en uno de estos cuatro tipos de formatos, añadiendo un sufijo de identificación al nombre de la variable % para las enteras, ! para las de simple precisión (punto flotante corto), # para las de doble precisión (punto flotante largo) y \$ para las cadenas. Las constantes numéricas pueden ser clasificadas similarmente. Los tipos no explícitos pueden controlarse con la sentencia DEF, y se toma por defecto la simple precisión. Como referencia, aquí hay ejemplos sencillos:

A%	Variable entera
A!	Variable de simple precisión
A#	Variable de doble precisión
A\$	Variable de cadena
1%	Constante entera
1!	Constante de simple precisión
1#	Constante de doble precisión
"1"	Cadena constante

❑ **NOTA IMPORTANTE:** *Mientras los tres formatos de datos numéricos son los mismos para el BASIC interpretado que para el compilado, los formatos de cadena son diferentes en el BASIC compilado.*

Formatos de datos enteros

El formato de datos enteros es el de palabras de 2 bytes (de 16 bits con signo) estándar usado universalmente para la familia PC. El rango de los valores es desde -32 768 hasta +32 767, pasando por el 0. Véase página 25 para obtener una explicación general de este formato de datos. El BASIC no admite la variación típica de este formato: enteros de 16 bits sin signo con un rango de valores desde 0 hasta 65 535. Ya que el formato entero sin signo es fundamental para el direccionamiento de memoria y cálculo de direcciones en el PC, se debe llevar algo de cuidado cuando se manipulan direcciones. Es costumbre el usar el formato entero con signo del BASIC para almacenar direcciones enteras sin signo; de hecho, ésta es la manera preferida de pasar direcciones a interfaces en ensamblador. Sin embargo, se debe proceder con cuidado cuando se escriben o calculan direcciones sin signo, que están almacenadas en formato entero con signo. Para evitar cálculos erróneos, es más prudente realizar los cálculos de direcciones en doble precisión, usando el formato largo de coma flotante.

Si se está usando el formato entero del BASIC para conservar y pasar direcciones de palabras sin signo, hay algunos puntos intere-

santes que recordar. En teoría, un entero del BASIC no puede aceptar ningún número sobre su límite de rango de 32767; por ejemplo, no es posible asignar a un entero decimal una constante de 50 000 o una variable en coma flotante con un valor equivalente. Sin embargo, el formato de constante hexadecimal puede ser usado para asignar valores desde 0 hasta 65535 (o desde &H0 hasta &HFFFF, en la notación de constante hexadecimal del BASIC). Esto significa que el BASIC no permite $I\% = 50\,000$, pero sí permite el equivalente en hexadecimal: $I\% = \&HC350$. Además, el BASIC proporciona la función de representación hexadecimal HEX\$, haciendo fácil convertir valores decimales en sus equivalentes hexadecimales. Estas dos características hacen razonablemente conveniente trabajar con direcciones en formato hexadecimal.

Pueden convertirse con seguridad valores de direcciones de formato entero a formato de coma flotante, para realizar operaciones aritméticas, usando este método:

```
IF I% < 0 THEN D# = I% + 65536# ELSE D# = I%
```

Donde $I\%$ es un entero y $D\#$ es su equivalente en doble precisión. Para convertir direcciones de doble precisión a enteros utilizaremos el método:

```
IF D# > 32767 THEN I% = D# - 65536 ELSE I% = D#
```

La función BASIC VARPTR proporciona directamente las direcciones relativas de las variables enteras (como se verá brevemente, también las proporciona para variables en coma flotante; para valores de cadena, VARPTR es una conexión indirecta con el valor de la variable). VARPTR nos da la dirección relativa dentro del segmento de datos del BASIC, que puede entonces ser usada para hacer PEEK, o POKE, en la zona de las variables. Se puede demostrar esto con el siguiente ejemplo:

```
I% = 999 ' o cualquier otro valor
PUNTERO.I = VARPTR (I%)
J% = PEEK (PUNTERO.I) * 256 + PEEK (PUNTERO.I + 1)
PRINT I%, J%
```

Este proceso de usar VARPTR para capturar la dirección de posicionamiento de una variable tiene un valor práctico pequeño, pero demuestra cómo encontrar y usar direcciones. Ejemplos como éste se diseñan para dar seguridad y ayudar a comprender el uso de las direcciones de memoria en BASIC —ambas cosas son importantes, como se podrá apreciar cuando se comiencen a crear interfaces en lenguaje ensamblador.

Formatos de datos en coma flotante

Los valores en coma flotante, tanto los de simple como los de doble precisión, se almacenan en un formato común peculiar del BASIC. No sólo es diferente el formato en coma flotante del BASIC del usado por muchos otros lenguajes de programación para el PC, sino que también es incompatible con los formatos usados por los coprocesadores matemáticos 8087 y 80287.

Para ayudarle a hacer uso del formato de datos en coma flotante, se describirán sus elementos clave. Pero esté prevenido de que el tema de los formatos en coma flotante es complejo. La siguiente explicación supone que se tiene un conocimiento general elevado sobre cómo almacenan y usan los números coma flotante los ordenadores.

En BASIC, los formatos de datos de simple y doble precisión solamente difieren en el número de dígitos de la mantisa. El resto del formato, incluyendo el rango de la mantisa, es el mismo. Los datos de simple precisión ocupan cuatro bytes, y los de doble precisión ocupan ocho bytes. La mantisa se almacena en los tres (o siete) primeros bytes, con los primeros bytes menos significativos delante (siguiendo la costumbre del microprocesador de PC). El exponente se almacena al final, ocupando el último byte. Es posible esquematizar los cuatro u ocho bytes de la siguiente manera:

M7 M6 M5 M4 M3 M2 M1 E

El exponente (E) se almacena como una potencia de 2, desplazada 128. Esto significa que un exponente de 0 sería almacenado como 128 (80 hex) y un exponente de -3 sería almacenado como 125 (7D hex).

La mantisa se almacena como una fracción binaria normalizada, con el primer bit, o bit de mayor peso implícito. El bit de mayor peso del byte de mayor peso (byte M1) se usa para almacenar el signo: 0 para valores positivos y 1 para valores negativos. El bit de signo ocupa el lugar que pertenece (pero no es usado) al bit implícito de mayor peso de la mantisa.

El programa siguiente muestra cómo decodificar un número en coma flotante. Este ejemplo ilustra la explicación anterior y debe ayudarle a comprender el formato si está teniendo problemas.

```

100 INPUT "Introduzca cualquier valor ",SIMPLE
110 DIRECCION = VARPTR (SIMPLE)
120 PRINT "Los bytes son "
130 FOR I = 0 TO 3
140   H = PEEK (DIRECCION + I)
150   IF H < 16 THEN PRINT "0";
160   PRINT HEX$ (H);" ";
170 NEXT

```

```
180 PRINT
190 E# = PEEK (DIRECCION + 3)
200 M1# = PEEK (DIRECCION + 2)
210 M2# = PEEK (DIRECCION + 1)
220 M3# = PEEK (DIRECCION + 0)
230 EXPONENTE# = E# - 128
240 SIGNO = M1# / 128
250 M1# = 128 + M1# MOD 128
260 MANTISA# = M1# / 256 + M2# / (256 * 256)
    + M3# / (256 * 256 * 256)
270 VALOR# = MANTISA# * 2 ^ EXPONENTE#
280 IF SIGNO THEN VALOR# = - VALOR#
290 PRINT "El valor decodificado es "; VALOR#
300 PRINT
310 GOTO 100
```

Advierta que las líneas desde la 190 hasta la 220 aíslan los cuatro bytes con el formato de doble precisión, usando nombres variables que corresponden a la notación que se utilizaba para representar los bytes de la mantisa. Antes se estableció que el exponente se almacena en un formato desplazado; la línea 230 retira el desplazamiento, dando un rango de exponente desde -128 hasta $+127$. Las líneas 240 y 250 analizan del bit de mayor peso del byte de mantisa más significativo: la 240 registra el valor del bit como signo, y la línea 250 pone el bit 1 implícito en su lugar. Después, la línea 260 pone los tres bytes de mantisa juntos en un solo valor, al que pone el punto decimal (o binario, si procede), justo antes del lugar del primer bit. La línea 270 opera el valor del exponente; la línea 280 aplica el signo, y —¡voilà!— hemos decodificado con éxito el valor del formato en punto flotante del BASIC.

Con los formatos en punto flotante conocidos, ya se puede echar un vistazo a cómo almacena las cadenas el BASIC.

Formatos de datos de cadena del BASIC interpretado

Los valores de cadena se almacenan en dos partes: un identificador de cadena que contiene la longitud y situación de la posición de la cadena; y el valor de la cadena, que es una serie de caracteres ASCII.

El identificador de cadena son tres bytes de largo. El primer byte contiene la longitud de la cadena, que limita el máximo tamaño de una cadena a 255 bytes. Los siguientes dos bytes proporcionan posición del segmento de datos del valor real de la cadena. El valor de la cadena no tiene un formato especial; se almacena simplemente como una serie de bytes en la dirección indicada.

Cuando se aplica la función `VARPTR` a una cadena, devuelve la posición del identificador de cadena. Desde el identificador de cadena

se puede obtener la dirección de la cadena. El siguiente programa demuestra el proceso del encuentro y decodificación de esta información.

```

100 INPUT "Introduzca cualquier cadena ",CADENAS$
110 DIR.ID = VARPTR (CADENAS$)
120 PRINT "El puntero de la cadena está en el hex ";
130 PRINT HEX$ (DIR.ID)
140 LONG.CADENA = PEEK (DIR.ID)
150 PRINT "La longitud de la cadena es";
160 PRINT LONG.CADENA
170 DIR.CADENA = PEEK (DIR.ID + 1)
    + 256 * PEEK (DIR.ID + 2)
180 PRINT "El valor de la cadena está en el hex ";
190 PRINT HEX$ (DIR.CADENA)
200 PRINT "El valor de la cadena es ";
210 FOR I = 0 TO LONG.CADENA - 1
220   PRINT CHR$ (PEEK (I + DIR.CADENA));
230 NEXT I
240 PRINT : PRINT
250 GOTO 100

```

Formatos de datos de cadena del BASIC compilado

El formato de datos de cadena es bastante diferente para el BASIC compilado. Mientras que no hay diferencia en el tipo de datos de cadena permitidos, hay una diferencia en la longitud de los datos de cadena permitidos. En el BASIC interpretado, la longitud de la cadena se registraba como un entero de un byte sin signo, que permitía una longitud de 0 a 255 caracteres. En el BASIC compilado, la longitud se registra como un entero de 2 bytes con signo, que permite una longitud de 0 a 32 767 caracteres (los posibles valores negativos se ignoran).

Como en el BASIC interpretado, a las cadenas en el BASIC compilado se accede indirectamente mediante un identificador de cadena. La dirección de memoria asociada con el nombre de la variable —la dirección dada por VARPTR (NOMBRE\$)— señala hacia el identificador de la cadena, en vez de a la cadena en sí. El identificador de cadena consta de dos campos que contienen la longitud de la cadena y la dirección relativa de datos del valor real de la cadena. La única diferencia entre el BASIC compilado y el interpretado es que la longitud del campo es de dos bytes en el BASIC compilado y sólo de un byte en el interpretado.

Es posible examinar cadenas con el siguiente programa. Para aclarar lo que debería ser obvio, adviértase que los detalles de este programa son característicos del BASIC compilado, y así pues no pueden ser probados con el BASIC interpretado. Este programa

difiere del programa del BASIC interpretado dado en la sección previa, solamente en la línea 140 (donde se toma la longitud de la cadena, y en la línea 170 en las direcciones del PEEK, usadas para coger la posición de la cadena).

```
100 INPUT "Introduzca cualquier cadena ",CADENA$
110 DIR.ID = VARPTR (CADENA$)
120 PRINT "El puntero de la cadena está en el hex ";
130 PRINT HEX$ (DIR.ID)
140 LONG.CADENA = PEEK (DIR.ID)
150 PRINT "La longitud de la cadena es";
160 PRINT LONG.CADENA
170 DIR.CADENA = PEEK (DIR.ID + 1)
    + 256 * PEEK (DIR.ID + 2)
180 PRINT "El valor de la cadena está en el hex ";
190 PRINT HEX$ (DIR.CADENA)
200 PRINT "El valor de la cadena es ";
210 FOR I = 0 TO LONG.CADENA - 1
220   PRINT CHR$ (PEEK (I + DIR.CADENA));
230 NEXT I
240 PRINT : PRINT
250 GOTO 100
```

Interfaces en ensamblador con BASIC interpretado

En esta sección se tratarán las reglas de conexión que se aplican a los interfaces en lenguaje ensamblador, llamados desde los programas en BASIC interpretados. Solamente se tratarán subrutinas llamadas con CALL, no con funciones USR. En mi opinión, las funciones USR implican complicaciones molestas e innecesarias, y poseen intrínsecamente problemas especiales que pueden variar enormemente entre los miembros IBM y los miembros no IBM de la familia PC ampliada. En general, no aconsejo el uso de las funciones USR.

El BASIC interpretado hace uso de una convención de conexión estándar que se ajusta muy bien a la norma acostumbrada para las conexiones de lenguaje ensamblador. Aquí está la esencia de la convención de conexión del BASIC:

- Todos los parámetros se pasan situando sus direcciones relativas en el *stack*, un formato conocido como llamada por referencia o nombre. Esto significa que las subrutinas en lenguaje ensamblador pueden acceder y cambiar los valores de los parámetros; también significa que en el *stack* aparecen las direcciones y no los valores, así que el acceso a los valores requiere varios pasos.

- Los parámetros se pasan en el orden en que son escritos, como en muchos otros lenguajes (excepto en el C; véase página 421). Esto significa que el primer parámetro está más próximo al fondo del *stack*, y el último parámetro está más próximo a la cima.
- Las subrutinas se llaman mediante FAR CALL y deben, por tanto, terminar con una instrucción FAR RET.
- La subrutina es responsable de retirar los parámetros del *stack*, como en muchos otros lenguajes (excepto el C, véase página 421). Esto significa, entre otras cosas, que se debe pasar a la subrutina un número fijo de parámetros, que es conocido de antemano.
- El registro AX no se usa para devolver valores ni códigos de error; a diferencia de muchos otros lenguajes, cualquier información devuelta por la subrutina debe ser pasada por cambio de los valores de los parámetros.
- El BASIC ajusta y mantiene los registros de segmento de código y de *stack*. Los otros registros pueden ser cambiados si se necesita.

Se describirán los detalles de una rutina de interfaz en lenguaje ensamblador en términos de los cinco niveles lógicos que se trataron antes en este capítulo y en el capítulo 8 (véase página 185). Los niveles 1 y 2, la cabecera del ensamblador, pueden ser codificados así:

```
MI_SEG  SEGMENT
        ASSUME  CS:MI_SEG
MI_PROC  PROC    FAR
        ; aquí aparecen los niveles del 3 al 5
MI_PROC  ENDP
MI_SEG  ENDS
END
```

Los nombres MI-SEG y MI-PROC son arbitrarios. En el BASIC interpretado se puede elegir cualquier nombre factible para identificar la subrutina, pero en el BASIC compilado hay restricciones.

Si se compara esta rutina con la rutina de interfaz general explicada en el capítulo 8, se notarán dos diferencias. Primero, en este ejemplo, la sentencia SEGMENT no está seguida por la clasificación «CODE»; es más, no tiene una sentencia PUBLIC MI-PROC. Los dos cambios, y el hecho de que el nombre del procedimiento es irrelevante para el BASIC, tienen que ver con el hecho de que los interfaces en lenguaje ensamblador no son monta-

dos con el BASIC interpretado. En los programas que estén montados, como en el resto de los lenguajes tratados en este capítulo, estos asuntos son importantes.

El siguiente nivel a considerar, el nivel 3, proporciona el código de entrada y salida de la subrutina. La forma del código es como sigue:

```
PUSH    BP
MOV     BP,SP
; aquí aparecen los niveles 4 y 5
POP     BP
RET     XXX
```

Estas cuatro instrucciones son estándar y deberían de ser las mismas para todos los interfaces en lenguaje ensamblador. El único elemento variable, mostrado como XXX en este ejemplo, representa el número de bytes de parámetros que hay en el *stack*. El valor del parámetro XXX debe ser el doble del número de parámetros que se pasaron cuando el BASIC llamó a la subrutina, debido a que cada parámetro ocupa dos bytes en el *stack*. Así, por ejemplo, si se pasan tres parámetros, se reemplazaría XXX por un 6 para limpiar el *stack*; si no nubiera parámetros, entonces se usaría un cero. A propósito, la instrucción RET usada en el ejemplo es ensamblada en una instrucción FAR RET, basada en el comando PROC FAR que apareció al principio de la rutina.

El siguiente nivel, el nivel 4, tiene que ver con el acceso y el intercambio de parámetros. Como se trató en la página 188, la dirección del último parámetro estará en el *stack* en la situación a que se refiere [BP + 6], el parámetro anterior en [BP + 8], y así sucesivamente.

A manera de ilustración, supóngase que hay tres parámetros, todos enteros, y que se desea cargarlos en los registros AX, BX y CX (por cualquier razón). Aquí está el código del nivel 4, que obtendría esos parámetros:

```
MOV     SI,[BP+10]    ; obtener la dirección del primer parámetro
MOV     AX,[SI]        ; obtener el primer valor
MOV     SI,[BP+8]      ; obtener la dirección del segundo parámetro
MOV     BX,[SI]        ; obtener el segundo valor
MOV     SI,[BP+6]      ; obtener la dirección del tercer parámetro
MOV     CX,[SI]        ; obtener el tercer parámetro
```

Adviértase que se obtienen los valores de los parámetros en dos pasos: primero se obtiene la dirección (que se carga en el registro SI, sólo porque es un sitio conveniente), y entonces se usa la dirección para obtener el valor (que se pone donde realmente interesa).

Eso ilustra sólo la primera mitad del nivel 4; la segunda mitad se

refiere al movimiento de valores hacia el programa BASIC, almacenándolos en los parámetros. Esto se consigue obteniendo la dirección del parámetro (de nuevo), y usando esa dirección para almacenar el nuevo valor. Por ejemplo, supóngase que se ha calculado un nuevo valor y se tiene en el registro DX. Es necesario pasarlo hacia BASIC como un nuevo valor del primer parámetro. Es posible hacerlo usando dos instrucciones:

```
MOV     SI,[BP+10]           ; obtener la dirección del primer parámetro (otra vez)
MOV     [SI],DX              ; pasar el valor desde DX
```

Para proporcionar un ejemplo completo, aquí hay una subrutina en lenguaje ensamblador que pasa al BASIC los valores de los registros SS y SP, una técnica que hace posible que un programa BASIC investigue en su propio *stack*. Normalmente no habría ninguna razón para hacerlo, pero se puede jugar con esto por la simple razón de que puede ser educativo. La subrutina supone que es llamada con dos parámetros enteros; sitúa los valores del segmento del *stack* (SS) y del puntero de *stack* (SP) en estos parámetros. Aquí está la rutina:

```
INFOSTACK SEGMENT
          ASSUME  CS:INFOSTACK
OBTENSTK  PROC   FAR
          PUSH   BP
          MOV    BP,SP
          MOV    SI,[BP+8]
          MOV    [SI],SS
          MOV    SI,[BP+6]
          MOV    [SI],SP
          POP    BP
          RET    4
OBTENSTK  ENDP
INFOSTACK ENDS
          END
```

Interfaces en ensamblador con BASIC compilado

Cualquiera que haya trabajado extensamente con el BASIC para la familia PC está, sin duda, familiarizado con las desesperantes diferencias e incompatibilidades entre el BASIC compilado y el interpretado. Aquí se señalará solamente la diferencia significativa que interesa para la conexión con lenguaje ensamblador, y después se trabajará con un ejemplo.

En el BASIC interpretado, la manera normal de trabajar con interfaces en lenguaje ensamblador es meterlos en alguna posición de

memoria (una de las tareas más molestas y propensas a error del BASIC), ajustar una variable con la dirección de memoria del interfaz y entonces llamar (CALL) al nombre de la variable.

Por contraste, el BASIC compilado proporciona dos métodos primarios para trabajar con interfaces en lenguaje ensamblador —uno que comparte prácticamente el método del BASIC interpretado, y otro que sigue las convenciones estándar para lenguajes compilados.

El primer método sigue prácticamente el método del BASIC interpretado en sustancia, aunque la forma difiere. Para ilustrarlo, se hará que POSICION% represente al nombre variable del BASIC que contiene con la posición de memoria del interfaz, y PARAMETROS representará cualquier parámetro que pase al interfaz. Con este valor, en el BASIC interpretado se llamaría al interfaz de la siguiente manera:

```
CALL POSICION% (PARAMETROS)
```

En el BASIC compilado el formato usado para llamar al interfaz sería bastante diferente, aunque el resultado neto es exactamente el mismo. Se hace así:

```
CALL ABSOLUTE (PARAMETROS, POSICION%)
```

Una diferencia crítica en la versión de BASIC compilado es que la variable POSICION% debe ser entera. En el BASIC interpretado, cualquier variable numérica es aceptable.

Algunos de los mecanismos de esta operación CALL ABSOLUTE son dignos de ser tratados. En el BASIC compilado todas las llamadas son realmente llamadas convencionales a rutinas externas que están montadas a los programas compilados. Este es el mecanismo estándar que se usa para todos los lenguajes compilados. El BASIC compilado realmente simula el equivalente del CALL POSICION% del BASIC interpretado, con una subrutina de librería llamada ABSOLUTE. En otras palabras, ABSOLUTE no es parte del lenguaje BASIC, en el sentido en que ON ERROR, o CHAIN, sí es parte del mismo. En vez de ello, ABSOLUTE es simplemente el nombre de una subrutina externa residente en la librería de montaje del BASIC. La subrutina ABSOLUTE realiza una operación simple: usa sus parámetros (incluyendo el parámetro POSICION%) para imitar la operación CALL POSICION% que realiza el BASIC interpretado.

Antes de continuar con el otro método de llamada de interfaces desde el BASIC compilado, se deberían comentar los problemas relacionados con la colocación de un interfaz en lenguaje ensambla-

dor en la memoria. Los mecanismos usados para situar estos interfaces en memoria son iguales para el BASIC interpretado y el compilado: se suele hacer o con la sentencia BLOAD, para cargar el conector desde un fichero, o haciendo POKE en la memoria byte a byte. Aunque los mecanismos sean los mismos para el BASIC interpretado y para el compilado, los problemas para encontrar una posición de memoria adecuada pueden ser bastante diferentes, dependiendo de las circunstancias y de las técnicas de programación. Advertirá que hasta ahora he evitado con éxito el tratar este tema, y voy a seguir haciéndolo. Este tema particular está poco claro y es un área realmente especial para los libros de BASIC.

El otro método para usar los interfaces en lenguaje ensamblador con el BASIC compilado es el método estándar utilizado por todos los lenguajes de programación DOS normales. Los módulos, tal como interfaces en lenguaje ensamblador, se preparan separadamente y se almacenan en forma de ficheros objeto diferentes (con una extensión del nombre de fichero .OBJ) o dentro de una librería de ficheros objeto (con una ampliación del nombre de fichero .LIB). En cada caso, los módulos independientes, tales como programas en BASIC compilado e interfaces en lenguaje ensamblador ensamblados, son combinados por el programa LINK, como se describió en el capítulo anterior.

Se va a analizar cómo se usa esta rutina de interfaz desde ambos lados —desde el lado BASIC y desde el lado del lenguaje ensamblador—. Desde el lado BASIC, un conector de lenguaje ensamblador se llama de este modo:

```
CALL NOMBRE (PARAMETROS)
```

Los PARAMETROS son los mismos que se usaron en el método anterior, o en el BASIC interpretado. El NOMBRE es el mismo nombre que aparece en el interfaz en lenguaje ensamblador y que identifica la rutina deseada.

Los pasos seguidos por el código del BASIC compilado al llamar a una subrutina son los mismos que los seguidos por el BASIC interpretado; las mismas reglas se aplican al uso del *stack*, a la situación de parámetros en el *stack* y al modo de hacer la llamada (una FAR CALL, requiriendo un FAR RET). Debido a esto, se usará la rutina de la sección anterior sobre el BASIC interpretado (la que pasaba los ajustes actuales del segmento de *stack* y del puntero de *stack*) y se señalarán las diferencias.

```
INFOSTACK SEGMENT
```

```
    ASSUME  CS:INFOSTACK
```

```
    PUBLIC  OBTENSTK
```

```

OBTENSTK  PROC      FAR
          PUSH      BP
          MOV       BP,SP
          MOV       SI,[BP+8]
          MOV       [SI],SS
          MOV       SI,[BP+6]
          MOV       [SI],SP
          POP       BP
          RET       4
OBTENSTK  ENDP
INFOSTACK ENDS
          END

```

El único requerimiento nuevo para acoplar este tipo de interfaz con un programa en BASIC compilado es que el nombre de la subrutina debe hacerse «público»; esto es, debe ser declarado como nombre oficial para ser usado por el ensamblador. Esto se hace en la tercera línea de esta rutina, que lee PUBLIC OBTENSTK. Esta es la diferencia entre esta rutina y la del capítulo 10. (Realmente, la sentencia PUBLIC no haría daño en la rutina anterior, pero no serviría para ningún propósito, ya que el BASIC interpretado no la usa.)

Para mostrar el uso de este tipo de interfaz, se da un programa en BASIC compilable que lo usa y muestra los resultados:

```

100 ' demostración del uso de interfaces montados
110 '
120 I% = 0
130 J% = 0
140 CALL OBTENSTK (I%,J%)
150 PRINT "En el programa"
160 PRINT "  el segmento del stack es ";HEX$(I%)
170 PRINT "  el puntero del stack es ";HEX$(J%)
180 GOSUB 200
190 GOTO 270
200 ' nuestra subrutina go-sub
210 '
220 CALL OBTENSTK (I%,J%)
230 PRINT "En la subrutina go-sub"
240 PRINT "  el segmento del stack es ";HEX$(I%)
250 PRINT "  el puntero del stack es ";HEX$(J%)
260 RETURN
270 END

```

Hay varias cosas que merece la pena anotar en este programa. Cuando se llama a la subrutina en lenguaje ensamblador (en las líneas 140 y 220), se llama por su nombre, OBTENSTK, que es el nombre que aparece en la sentencia PUBLIC del código en lenguaje ensamblador. Los nombres deben coincidir para trabajar en el proceso

de acoplamiento. (Por otro lado, el nombre del segmento en el programa en lenguaje ensamblador, INFOSTACK, es arbitrario y no importa, mientras sea utilizado consistentemente dentro del ensamblador.) Vea también que tanto el programa principal en BASIC como el programa llamado en lenguaje ensamblador deberán coincidir en el número de parámetros que están siendo usados. La sentencia RET 4, que en la última instrucción de trabajo en ensamblador, extrae cuatro bytes de parámetros del *stack*, que corresponde a dos parámetros con dos bytes por parámetro. (Una rutina en lenguaje ensamblador sofisticada y trucada puede aceptar un número variable de parámetros, pero los métodos son demasiado avanzados para nosotros para tratarlos debidamente aquí.)

Este programa de muestra informa del estado del *stack* dos veces (dado que ejecuta dos veces la rutina en lenguaje ensamblador): una vez en el movimiento lineal de sentencias, y otra, en una rutina GOSUB. La diferencia entre los dos valores del puntero del *stack* nos permite descubrir cómo afecta el uso de GOSUB al *stack* del BASIC. Ejecute este ejemplo para conocer la respuesta.

Aquí están los pasos necesarios para preparar y combinar programas en BASIC compilado e interfaces, usando los programas precedentes como un ejemplo. Primero, suponiendo que se tiene el código fuente del lenguaje ensamblador en un fichero llamado OBTENSTK.ASM, se ensamblaría con este comando:

```
MASM OBTENSTK;
```

Esto producirá la creación de un fichero objeto llamado OBTENSTK.OBJ. Después, suponiendo que se tiene el código fuente en BASIC en un fichero llamado TEST.BAS —que debe ser un fichero ASCII, no un fichero en BASIC comprimido y codificado—, se compilaría con este comando:

```
BASCOM TEST;
```

Esto originaría la creación de un fichero objeto llamado TEST.OBJ. Después se acoplarían juntos con este comando:

```
LINK TEST+OBTENSTK
```

que causaría la creación de un fichero de programa llamado TEST.EXE.

PASCAL

En esta sección se tratará el compilador de Pascal de IBM y su primo genérico, Microsoft Pascal. Cuando se discutan los formatos de datos del Pascal, se indicará de forma general qué elementos son

característicos de estos compiladores y cuáles del Pascal estándar. Si se está usando cualquier versión de estos compiladores, la información que aquí se da se aplicará en detalle; de otro modo, deberá ser capaz de usar la explicación como una base para determinar las características de su compilador particular.

Deberá tenerse en cuenta que hay algunas diferencias significativas entre la versión 1 y la versión 2 del compilador de Pascal del IBM PC. Se señalarán cuando se expliquen los formatos de datos.

Formatos de datos de Pascal

Hay tres formatos de datos usados por el Pascal: entero, en coma flotante (conocido como REAL en la terminología Pascal) y de cadena. Hay también un tipo especial conocido como set o conjunto. Dentro de los tipos, hay unas cuantas variaciones, particularmente dentro del tipo entero.

Formatos de datos enteros

Se comenzará con el tipo entero, con todas sus variaciones. Los enteros se almacenan como números binarios y se sitúan en memoria con el byte menos significativo primero. Los enteros pueden ser de uno, dos o cuatro bytes de longitud, con signo o sin signo. De los seis formatos que sugiere esta descripción, cinco se usan realmente (el de 4 bytes sin signo no), y varios de los formatos hacen una doble tarea desde el punto de vista de Pascal.

Los enteros de un solo byte están disponibles en formato con signo y sin signo. La forma con signo tiene un rango desde -128 hasta $+127$, y se llaman SINT (enteros cortos) en Pascal; la forma sin signo tiene un rango de 0 a 255, y se llama BYTE. Ni SINT ni BYTE son parte del Pascal estándar, y ya que no se habla mucho de ellas en la documentación del compilador, es fácil pasarlas por alto.

Los enteros de dos bytes también están disponibles en forma con y sin signo. La forma con signo tiene un rango desde $-32\,768$ hasta $+32\,767$, y es llamada INTEGER; es un formato estándar del Pascal. La forma sin signo tiene el rango desde 0 hasta 65\,535, y es llamada WORD; no es un formato estándar del Pascal. El formato INTEGER es, por supuesto, el más universal de los formatos para los lenguajes de programación de la familia PC; p. ej., comparte exactamente el formato usado por el BASIC. El formato de 16 bits sin signo WORD también tiene una importancia especial. Todas las facilidades de dirección en Pascal, como ADR y ADS, están basadas en el formato WORD. (● Véanse más notas sobre direcciones en Pascal en la

página 418.) El formato WORD del Pascal es el mismo que el de los enteros de 16 bits sin signo estándar usado en la familia PC y el mismo que el formato UNSIGNED INT del C.

Los enteros de cuatro bytes están disponibles sólo en formato con signo y sólo en versiones del compilador numeradas 2.0 y en adelante. Este formato tiene el rango de $-2\,147\,483\,647$ hasta $+2\,147\,483\,647$, y es llamado INTEGER4. (Advierta que debería haber —y probablemente haya— un valor negativo adicional, aunque no hay forma de que el compilador defina este formato. Si piensa que lo necesita de todos modos, tendrá un problema.) Este formato entero de 32 bits no es parte del Pascal estándar ni es usado por muchos otros lenguajes del PC. Sin embargo, es parte del C, donde se le llama LONG INT (● Véase página 422). El formato INTEGER4 no está integrado totalmente con los otros formatos enteros del Pascal; en muchas circunstancias en donde cualquier otro formato entero puede ser usado, el INTEGER4 no puede.

El lenguaje estándar Pascal proporciona una generalización de los enteros en un tipo de datos conocido como datos de enumeración. Para todas las aplicaciones prácticas, cada tipo de datos de enumeración está compuesto de enteros sin signo, desde 0 en adelante, a los que se les dan nuevos nombres. El formato de datos real para todos los tipos de datos de enumeración está compuesto de enteros sin signo en el formato de 1 byte (BYTE) o en el de 2 bytes (WORD), dependiendo de si el número de valores cabe en un byte o no. BOOLEAN, el tipo especial más común, es simplemente un tipo de enumeración predeclarado con dos valores. Como tal, ocupa un byte, y solamente toma los valores 0 ó 1.

Formatos de datos de cadena

El Pascal estándar tiene un tipo de carácter llamado CHAR, que consiste en un único carácter ASCII almacenado en un único byte. Dependiendo del punto de vista, es posible considerar el tipo de datos CHAR como un caso especial del tipo BYTE, o como un elemento de los tipos de datos de cadena. Sin embargo, el Pascal trata al CHAR como su propio formato distinto.

Hay dos formatos de cadena: Uno, parte del Pascal estándar, es el llamado STRING, y contiene cadenas de longitud fija de caracteres ASCII; el otro es una ampliación del Pascal, se denomina LSTRING, y contiene cadenas de caracteres de longitud variable. La mayoría de los compiladores de Pascal ha añadido cadenas de longitud variable al lenguaje estándar, pero lo han hecho de varias formas diferentes. Se tratará el formato usado por los compiladores IBM/Microsoft.

Las cadenas de longitud fija se almacenan simplemente como cadenas de bytes, sin un formato ni una delimitación especial. Veamos si *S* es una cadena de longitud fija; entonces en la notación de Pascal el primer carácter de esa cadena será *S*[1]. La dirección de una cadena de longitud fija es la misma que la dirección de su primer carácter.

Las cadenas de longitud variable se almacenan como cadenas de bytes de caracteres, precedidas por un código de 1 byte de longitud en forma de entero sin signo (un BYTE en la notación de este compilador); por tanto, las cadenas de longitud variable pueden contener desde 0 hasta 255 caracteres. Para una cadena de longitud variable, como para una de longitud fija, *S*[1] es el primer carácter de la cadena, y *S*[0] se refiere al byte que contiene la longitud de la cadena. La dirección de una cadena de longitud variable es la dirección de su byte de longitud *S*[0], en vez de la de su primer byte de datos, *S*[1].

□ **NOTA:** *En Pascal, la longitud de cualquier cadena debe ser declarada. Para una cadena de longitud fija, la longitud es igual al número de bytes de la cadena; las cadenas de longitud variable usa un byte adicional.*

El formato de datos de tipo SET (conjunto)

El tipo SET es un elemento especial del Pascal, algo no compartido con muchos otros lenguajes. Los SET deben estar basados en un tipo de datos de enumeración, y el tipo de enumeración no debe tener más de 256 elementos. El formato de datos para el SET lo constituye un bit distinto por cada elemento del tipo de datos de enumeración fundamental. Si el elemento del tipo de enumeración está en el SET, entonces el bit correspondiente se ajusta a 1. El tamaño de los datos del SET depende cuántos bits se necesitan; esto es, de cuántos elementos hay en el tipo de dato de enumeración. Sin embargo, a diferencia de lo que se podría esperar, los datos del SET se miden en unidades de 2 bytes; así pues, un SET de ocho elementos ocupa dos bytes, en vez del único byte en el que podría estar almacenados. El mínimo tamaño de datos de SET es de dos bytes (usado por SET de uno hasta 16 elementos); el máximo es de 32 bytes (usado por SET basados en un tipo de datos de enumeración con 241 hasta 256 elementos).

Los bits en los datos de tipo SET se asignan de izquierda a derecha, y el código de bits corresponde al orden de los elementos declarado en el tipo de datos de enumeración. Por ejemplo, el primer elemento en el tipo de numeración corresponde al bit de mayor peso del primer byte (en hexadecimal 80 hex); el siguiente elemento en el

tipo de enumeración en el siguiente bit (40 hex), y así en adelante. Los bits no usados —que redondean el tamaño de los datos a dos bytes, o 16 bits— se ajustan a cero, como ya se supondrá.

Formatos de datos en punto flotante

Los formatos de datos en coma flotante, llamados REAL en la terminología Pascal, presentan algunas complicaciones interesantes. Hay tres formatos que se deben considerar. El primer formato se aplica a la versión 1 del compilador e imita exactamente al formato usado por el BASIC para números de simple precisión; solamente para esta explicación a este formato se le denominará «real viejo». Los otros dos formatos se aplican a la versión 2 (y versiones posteriores) del compilador; estos formatos corresponden a una forma estándar usada también por el coprocesador matemático 8087. Estos dos formatos son conocidos, en la terminología de este compilador de Pascal, como REAL4 y REAL8; en esta explicación se les llamará «reales nuevos».

□ *NOTA: No hay ninguna forma buena, uniforme, de convertir los datos entre el formato de real viejo usado por el compilador viejo y los formatos de reales nuevos usados por el nuevo compilador.*

Primero se trata brevemente el formato real viejo para que se pueda comparar con los otros dos formatos. Para obtener una información más detallada, véase la explicación del formato de simple precisión del BASIC, página 402. Ya que los dos formatos son iguales, se supondrá que se está familiarizando con ambas aritméticas de punto flotante y los formatos de datos generales usados para almacenarlas; se tratarán las particularidades de estos formatos de datos en punto flotante, no los principios básicos de los números en punto flotante.

El formato real viejo se almacena en cuatro bytes, que pueden ser resumidos así:

M3 M2 M1 E

donde los bytes M son la mantisa, almacenada con los bytes menos significativos delante. El bit de mayor peso, M1, contiene el valor del signo (el valor 1 indica que es negativo); en lugar del bit implícito de mayor peso de la mantisa está el bit de signo. El byte E contiene el exponente binario en forma exceso-128 (esto es, E-128 da el verdadero exponente). La mantisa se trata como una fracción; esto es, el punto «decimal» (en realidad binario) está a la izquierda del bit de mayor peso implícito.

Los formatos reales nuevos difieren en todos los detalles importantes del formato real viejo. Entre ellos siguen el mismo diseño, y sólo difieren en el número de bits dado a cada parte.

Cada formato consta de tres campos: un signo, un exponente y una mantisa. Se almacenan en ese orden, con los bits menos significativos primero. En otras palabras, el bit de mayor peso del último byte, el situado a la derecha, contiene el signo, seguido por el bit de mayor peso del exponente, y así sucesivamente. Siguiendo esta muestra, los bits menos significativos de la mantisa se encontrarán en el primer byte, o byte situado más a la izquierda.

El campo del signo en ambos formatos es de un bit, con un 1 indicando los números negativos y un 0 indicando los positivos.

El campo del exponente está en la forma de un entero con notación por exceso. En el caso del REAL4, el exponente ocupa 8 bits y está en la forma exceso-127 (esto es, al restar 127 del exponente aparente, se obtendrá verdadero valor del exponente). En el caso del REAL8, el exponente ocupa 11 bits y es de la forma exceso - 1 023. Por tanto, el rango de exponentes para el REAL4 va desde -127 hasta +128, y para el REAL8 va desde -1 023 hasta +1 024. Adviértase, a modo de comparación, que el formato real viejo (que también es usado por el BASIC) es de la forma exceso-128, en vez de 127 como se ve aquí. Los formatos reales nuevos dan un rango ligeramente más dinámico para números muy grandes, mientras que el formato real viejo da un rango más dinámico para números muy pequeños. Véase que a diferencia de los formatos de simple y doble precisión del BASIC, el REAL4 y el REAL8 difieren en el rango de sus exponentes, así como en la precisión de la mantisa.

El campo de la mantisa tiene la forma de una fracción binaria, con el bit 1 de mayor peso implícito (como en el formato real viejo y en el BASIC). El punto lógico decimal (o binario) se sitúa a la derecha del bit de mayor peso implícito de la mantisa. Para el REAL4, hay 23 bits de mantisa (más el bit de mayor peso implícito), y para el REAL8 hay 52 bits de mantisa (más el bit implícito).

Las partes de un REAL4 suman hasta 32 bits, o cuatro bytes; las de un REAL8 suman hasta 64 bits, u ocho bytes. Sin embargo, a diferencia de los formatos real viejo y del BASIC, las partes del REAL4 y del REAL8 no encajan pulcramente en bytes. Aquí se da un tosco diagrama de estos dos formatos:

REAL4
M3 M2 E2M1 SE1

REAL8
M7 M6 M5 M4 M3 M2 E2M1 SE1

Los bytes de mantisa desde M2 hasta M7 son puros bytes de mantisa. Los otros bytes contienen una mezcla de campos y partes. Los bytes

etiquetados SE1 en ambos formatos contienen el bit de signo y los primeros siete bits del exponente; así:

Bit								Descripción
7	6	5	4	3	2	1	0	
S	.	.	.	.	.	.	.	Bit de signo
.	E	E	E	E	E	E	E	Bits de mayor peso del exponente

Los bytes etiquetados E2M1 contienen el resto de los bits del exponente (un bit para el REAL4, cuatro bits para el REAL8) y los primeros siete o cuatro bits de la mantisa; así:

REAL4								REAL8								Descripción
Bit								Bit								
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	
E	.	.	.	.	.	.	.	E	E	E	E	.	.	.	.	Bits de menor peso del exponente
.	M	M	M	M	M	M	M	.	.	.	.	M	M	M	M	Bits de mantisa

Aunque los dos formatos reales nuevos y el formato real viejo funcionan bajo los mismos principios, difieren casi en cada detalle: orden y tamaño de las partes, notación en exceso del exponente y límite de bytes.

Anteriormente en este capítulo, como una ayuda para comprender el formato en punto flotante de simple precisión del BASIC (que es el mismo que el formato real viejo de la versión 1 del Pascal), se mostró un programa que decodificaba los bits de un número y construía el valor que representaban. El mismo ejercicio puede ser realizado en la versión 1 del Pascal para interpretar el formato antiguo, o en la versión 2 para los formatos REAL4 y REAL8. Los principios son los mismos para los tres formatos, aunque los detalles son más complicados para el REAL4 y para el REAL8, ya que los campos no encajan tan perfectamente en los límites de los bytes.

Por razones de compatibilidad, la versión 1 del compilador tiene un formato en punto flotante compatible con el BASIC y la versión 2 tiene formatos compatibles con muchos otros lenguajes, incluyendo el Fortran, de IBM/Microsoft, y el C de Lattice/Microsoft, y con los coprocesadores matemáticos 8087 y 80287.

Interfaces de ensamblador con Pascal

Aquí se tratarán detalles importantes, necesarios para el uso de parámetros en Pascal. Se tomará prestado el ejemplo específico de Pascal del capítulo 10, dejándolo en sus componentes clave:

```

1  INTERFAZ SEGMENT 'CODE'
2              PUBLIC LEESEG
3  LEESEG      PROC    FAR
4              PUSH    BP
5              MOV     BP,SP
6              ...
7              POP     BP
8              RET     10
9  LEESEG      ENDP
10 INTERFAZ    ENDS
11             END

```

En la línea 1, el nombre INTERFAZ es arbitrario; sin embargo, la clasificación «CODE», con comillas simples, es necesaria para montar con éxito esta rutina con el compilador de Pascal.

En la línea 2, el nombre de cualquier rutina, tal como LEESEG, debe ser declarada con PUBLIC para que el montaje la conecte a las rutinas que la usan. En la línea 3 el PROC debe ser declarado con FAR, debido a que el Pascal hace llamadas de tipo lejano a las rutinas externas.

Las líneas 4 y 5, y 7 y 8, representan el código estándar de entrada-salida, que mantiene el *stack*. El valor 10 en la línea 8 representa la cantidad de espacio del *stack* que usan los parámetros (un tema que se tratará después).

En las convenciones de llamada estándar para la familia PC, los parámetros de subrutina, en una u otra forma, se sitúan en el *stack*. Hay dos caminos básicos que los programas pueden tomar para situar los parámetros en el *stack*. Pueden situarse en el *stack* o bien el valor del parámetro o bien un puntero del valor en la memoria. Hay ventajas en ambos sistemas: el primero permite un acceso más fácil a los valores, el segundo permite a una subrutina devolver datos modificando los parámetros variables. El Pascal usa ambos métodos, y dentro de unos límites permite elegir cuál usar. (Diferentemente, el BASIC solamente usa el segundo método, situando la dirección del parámetro en el *stack*.)

Si cuando se define una subrutina, se especifica un parámetro como VAR, o VARS, se está instruyendo al Pascal para que sitúe la dirección de ese parámetro en el *stack*. (Desde el punto de vista de Pascal, usar VAR, o VARS, es darle permiso a una subrutina para que cambie el valor de una variable; en la práctica, significa que se le está dando a una subrutina la dirección de una variable.) Si se especifica VAR, la dirección se pasa como un desplazamiento dentro del segmento de datos, una sola palabra de 2 bytes. Si se especifica VARS, entonces la dirección se pasa como una dirección segmentada, ocupando dos palabras de 2 bytes. La diferencia es de

una importancia considerable, porque afecta al número de bytes de parámetro que se deben retirar del *stack* en una sentencia RET.

Por otro lado, si no se especifica VAR, el Pascal evitará que el valor del parámetro sea modificado permanentemente por la subrutina. Esto puede hacerse de dos maneras, y es de gran importancia cuál usar. Si el valor es del tipo adecuado y puede ser situado en el *stack* (enteros, por ejemplo), entonces el valor en sí es situado en el *stack*; de otro modo, se hace en la memoria una copia del valor, y la dirección de la copia se sitúa en el *stack*. La diferencia es muy importante por dos razones: la primera es que la rutina llamada accederá al parámetro de modo distinto, u obteniendo el valor del *stack* u obteniendo el valor mediante una dirección en el *stack*, la otra es que el obtener el valor a través de una dirección es mucho menos eficiente para el programa principal.

Cuando se realiza una rutina de interfaz es importante saber qué método de transferencia de parámetros es utilizado. Esto se puede averiguar usando el mismo método que yo he usado: tomar la información disponible y usar el sentido común para imaginar lo que está ocurriendo con cualquier juego de parámetros. También se pueden realizar tests, o ejecutar un par de programas de prueba, o inspeccionando el listado del compilador en pseudoensamblador.

Si se está indeciso, es posible obligar a todos los parámetros a ser pasados a través de direcciones declarándolos como VAR. El uso del VAR uniformemente tiene la ventaja de producir una simple y consistente forma de manipulación de parámetros. De hecho, la única ventaja de no usar VAR para variables simples es que VARS puede ahorrar dos instrucciones en lenguaje máquina, un beneficio mucho menor.

Una subrutina puede devolver un valor si es declarada como FUNCTION en Pascal. Si la subrutina devuelve un valor suficientemente simple, entonces la rutina del programa principal en Pascal espera que el valor de retorno sea situado en el registro AX. De otro modo, el valor se devuelve almacenándolo en la memoria en una dirección relativa que se sitúa en el *stack*, siguiendo a los parámetros de la función. En efecto, el Pascal crea un parámetro de tipo VAR adicional para la subrutina de la función, y el valor de la función de la subrutina se devuelve en ese parámetro.

Para mostrar cómo se obtienen los parámetros a través del *stack*, creemos un ejemplo. Suponga que una rutina de interfaz se declara en Pascal así:

```
PROCEDURE EJEMPLO (I : INTEGER; VAR J : INTEGER);  
EXTERNAL;
```

La subrutina de lenguaje ensamblador encontrará que el primer parámetro, I, está en el *stack* en la posición [BP + 8]. Para poner ese valor en el registro AX, se codificaría esta instrucción en lenguaje ensamblador.

```
MOV    AX,[BP+6]
```

Por otro lado, el segundo parámetro, J, tiene su dirección situada en el *stack* en la posición [BP + 6], ya que es el siguiente parámetro. Así, para poner ese valor en el registro AX hay que seguir dos pasos: uno para extraer del *stack* la dirección, y otro para obtener el valor de J, de este modo:

```
MOV    BX,[BP+6]
MOV    AX,[BX]
```

Para modificar el valor de J, se invertiría el proceso:

```
MOV    BX,[BP+6]
MOV    [BX],AX
```

El valor de I no puede ser modificado por esta subrutina, ya que no se sabe dónde está: sólo se sabe que una copia de su valor estaba situada en el *stack*.

Estos principios, usados con inteligencia y alguna investigación sobre los particulares de su situación, deberán proporcionarle todo lo que necesite para crear con éxito conectores de lenguaje ensamblador para usarlos con Pascal.

EL LENGUAJE C

Esta explicación del lenguaje C se centrará en la versión 1.04 del compilador C de Lattice/Microsoft y otros compiladores relacionados (algunos de los cuales están disponibles bajo los nombres de Lattice y Lifeboat). Como en la explicación de otros lenguajes, la información específica que se da aquí está basada en este compilador en particular, pero también se aplica en varios grados a otras versiones y a compiladores totalmente diferentes.

Merece la pena detenerse para señalar que el compilador C de Lattice/Microsoft usa lo que se llama el modelo de memoria pequeño, lo que significa que los programas y los datos están limitados a segmentos no más grandes de 64K cada uno.

Como contraste, el modelo de memoria media usa direcciones de código segmentadas (lo que permite a cualquier programa crecer hasta cualquier tamaño, mientras deja los datos limitados a 64K), y el modelo de memoria grande usa direcciones segmentadas, tanto para

el código como para los datos (lo que les permite a ambos crecer hasta cualquier tamaño). El BASIC compilado y el Pascal tratados antes usan el modelo medio (aunque el Pascal hace un uso limitado de direcciones segmentadas para datos, así que tiene algunos aspectos del modelo grande). Las últimas versiones del compilador C de Lattice (un pariente próximo del compilador que se está tratando) pueden usar cualquiera de los tres modelos de memoria, según se desee. La disponibilidad de tres modelos de memoria en un compilador complica en gran manera la explicación de interfaces (una complicación que se evitará aquí). En vez de ello, la explicación se centrará en el modelo pequeño.

El uso del modelo pequeño limita el tamaño del código de programa y el de los datos, pero es el formato más eficiente. Se verá el efecto del modelo pequeño en el formato de direcciones de datos, mencionado en la página 423, en la entrada y salida de una subrutina, y en el acceso a los parámetros, mencionado en la página 426.

Formatos de datos en C

Hay tres principales clases de datos en el C: enteros, en punto flotante y de cadena. Para el compilador C de Lattice/Microsoff, hay cuatro formatos enteros distintos y dos formatos de punto flotante.

Formatos de datos enteros

Se comenzará con los cuatro formatos enteros CHAR, INT, UNSIGNED y LONG. El formato CHAR ocupa un byte y es tratado como un entero sin signo, cuyos valores pueden oscilar entre 0 y 255. El formato INT ocupa una palabra de 2 bytes y es tratado como un valor con signo, así que puede oscilar entre -32 768 y +32 767. El formato UNSIGNED (también conocido como UNSIGNED INT en una terminología C más formal) ocupa una palabra de 2 bytes, y es tratado como un valor sin signo y puede oscilar entre 0 y 65,535. El formato LONG ocupa 4 bytes y es tratado como un valor con signo, así que puede oscilar entre -2 147 483 648 y +2 147 483 547.

Aunque estos cuatro formatos enteros parecen ser bastante sencillos, hay varias cosas sobre ellos que merecen ser discutidas. Primero, debería notarse que la definición general del lenguaje C permite versiones LONG y SHORT, y con signo y UNSIGNED de muchos de los tipos de datos fundamentales. Con este compilador, SHORT INT es un valor de 2 bytes con signo (el mismo que INT), y no hay ni una versión UNSIGNED del LONG de 4 bytes, ni una

versión con signo del CHAR de 1 byte. Los valores de direcciones aparecen en la forma de desplazamientos sobre el segmento de datos. Como se podría esperar, se almacenan en el formato UNSIGNED INT de 16 bits.

El lenguaje C, en general, es un poco confuso con respecto a la relación entre caracteres y números, y particularmente con las similitudes y diferencias entre CHAR e INT. Para este compilador, los CHAR son estrictamente números que ocupan un byte, con un rango de valor de 0 a 255. Sin embargo, los CHAR también pueden ser expresados en la forma de caracteres. Por ejemplo, la constante de carácter Q se traduce en el valor numérico 81; pero si se fuese a escribir 81 en un programa C, sería considerado de tipo INT y no CHAR de 1 byte —a no ser que se haga algo que lo convierta al formato CHAR—. Debido a que el C no hace una firme distinción entre caracteres y números (a diferencia del Pascal), es posible convertir valores entre formatos enteros (como INT, o UNSIGNED) y formatos de caracteres (CHAR, o elementos de cadenas).

Formatos de datos de cadena

Las cadenas se manipulan en el C de una manera que refleja el trato ambiguo de los caracteres como caracteres o números. En el C, una cadena de caracteres es un grupo de CHAR seguido por un carácter cero, que marca el final de la cadena. El uso de un carácter cero para terminar una cadena significa que las cadenas pueden tener cualquier longitud, ya que no hay un indicador de longitud explícito (como lo hay en Pascal, o en BASIC) que pueda ponerle un límite al máximo tamaño. Pero debido a que no hay ningún indicador de longitud, la cadena debe ser inspeccionada desde el principio hasta el fin para conocer la longitud de la cadena, y ya que las cadenas terminan con un carácter cero, el carácter cero, CHR\$(0) en la notación del BASIC, no puede ser parte de una cadena. El uso de un carácter cero para terminar una cadena también significa que no hay distinción en la forma entre cadenas de longitud fija y variable.

El compilador C se cuida de la tarea de añadir el byte cero final de cualquier cadena que se le muestre como valor literal; el programador no tiene que preocuparse de ponerlo allí. Por ejemplo, si se escribe *abc* en un programa, el compilador C crea una cadena de 4 bytes, que se compone de los tres bytes que se han escrito, seguidos del byte cero. Debe saber que, aunque el compilador añadirá el byte cero a cualquier constante de cadena, y las subrutinas de manipulación de cadena estándar del C se cuidarán del byte cero cuando sea necesario, el byte cero no aparece por arte de magia. Si se escribe cualquier

programa de manipulación de cadenas —una cosa común—, es necesario tener este byte en cuenta, cuidándolo y creándolo cuando se necesite.

Formatos de datos en punto flotante

Hay dos formatos de punto flotante usados por este compilador: FLOAT y LONG FLOAT (o DOUBLE). El formato FLOAT es de cuatro bytes de longitud y el formato LONG FLOAT es de ocho bytes de longitud. Siguen el formato estándar usado por los coprocesadores matemáticos 8087 y 80287, y por muchos lenguajes de programación. En esta explicación se tratarán por encima los dos formatos. Están descritos con mayor detalle en la página 416, bajo los nombres de REAL4 y REAL8.

Los dos formatos en coma flotante son de lo más fácil de entender si se analizan como cadenas de bits (32 bits para el FLOAT, 64 bits para el LONG FLOAT), almacenados al revés con los bits más significativos en los últimos bytes. Se describirán los formatos en coma flotante a nivel de bits, comenzando con el bit más significativo.

El primero, el más significativo de los bits, es el bit de signo: 0 para positivo, 1 para negativo. El siguiente grupo de bits (ocho para el FLOAT, once para el LONG FLOAT) especifica el exponente binario en notación de exceso: exceso 127 para el FLOAT, exceso 1 023 para el LONG FLOAT. Si se forman los bits del exponente entero binario sin signo y se les resta el valor de exceso (127 ó 1 023), se obtiene el verdadero valor del exponente. Los bits restantes (23 o 52) son la mantisa, o parte fraccionaria. El bit implícito de mayor peso tiene su lugar delante del resto de los bits de mantisa. El punto decimal (realmente binario) tiene su sitio entre el bit implícito de mayor peso y los restantes bits de la mantisa. En la página 418 hay una tabla que muestra la distribución de los bits de signo, exponente y mantista en los cuatro u ocho bytes de estos dos formatos.

Interfaces de ensamblador con C

Las reglas generales para el acceso de los cinco niveles a la programación de interfaces se aplican al compilador de lenguaje C, tal como ocurre con los otros compiladores que se han explicado. Para demostrar la estructura básica de una rutina de interfaz que será usada con el compilador C, se tomará prestado un programa y se reducirá a sus elementos clave. El programa original, listado en la

página 330, fue diseñado para calcular el día de la semana, usando los servicios del DOS de fecha y hora.

```

1          PGROUP  GROUP  PROG
2  PROG    SEGMENT BYTE PUBLIC 'PROG'
3          PUBLIC  DIASEM
4          ASSUME  CS:PROG
5  DIASEM  PROC    NEAR
6          PUSH    BP
7          MOV     BP,SP
8          ...
9          POP     BP
10         RET
11  DIASEM  ENDP
12  PROG    ENDS
13         END

```

La línea 1 se necesita para ayudar a coordinar el interfaz de lenguaje ensamblador con las convenciones de montaje usadas por el compilador C. En la línea 2, el nombre PROG y la clasificación 'CODE', entre comillas simples, son también necesarias para satisfacer las convenciones de montaje.

En la línea 3, el nombre de cualquier rutina, tal como DIASEM, se debe declarar con PUBLIC, a fin de que el montaje la conecte a las rutinas que la usan. En la línea 4, el ASSUME es necesario para ensamblar este proceso como NEAR, incluso si no se está haciendo ninguna referencia a direcciones de programa. En las líneas 5, el PROC debe ser declarado con NEAR, debido a que este compilador C hace llamadas de tipo cercano a rutinas externas. Esta característica es una diferencia con muchos otros lenguajes, y es la primera de tres importantes desviaciones que hace este compilador C de las reglas de conexión más comunes.

Las líneas 6, 7, 9 y 10 representan el código estándar de entrada y de salida, que mantienen el *stack*.

Advertirá que en la línea 10 no hay valor a continuación de la instrucción RET. Los parámetros de subrutina son situados en el *stack* por el usuario y pueden ser retirados del *stack* o por la subrutina llamada o por el usuario. Como se ha visto en muchos lenguajes de programación, esto lo realiza la subrutina llamada, especificando el número de bytes de parámetro a ser retirados con la instrucción RET. Sin embargo, en este compilador C la subrutina llamadora retira parámetros del *stack*, así que la instrucción RET no tiene ningún número siguiéndola. Esta es la segunda de las tres desviaciones del C respecto a las reglas de conexión convencionales.

Advertirá que el C también permite llamar a una subrutina con un número de parámetros variable; esto es, una parte estándar del

lenguaje C, y es necesario, por ejemplo, para la rutina `PRINTF`, que se usa a menudo. Una rutina llamada puede determinar cuántos parámetros fueron introducidos en el *stack*, comprobando los valores de los punteros del *stack* y del BP.

Las líneas 11, 12 y 13 son los elementos estándar para finalizar una rutina de conexión.

Transferencia de parámetros en C

Los parámetros se sitúan en el *stack* directamente o indirectamente. El compilador C sitúa los parámetros en el *stack* en orden inverso en que fueron escritos; así que el primer parámetro se introduce en el *stack* el último, y es, por tanto, más próximo; esto es, tiene el mínimo desplazamiento del registro BP. Esto es contrario a la convención usada por muchos lenguajes, y es la última de las tres cosas inusuales de los interfaces con C.

Como se ha explicado en algún sitio, los parámetros pueden ser pasados o bien situando en el *stack* su valor o bien su dirección. A menos que se le ordene otra cosa, el compilador C intentará situar el valor en el *stack* en vez de en la dirección. Esta afirmación puede ser un poco engañosa, porque el C considera como direcciones muchas cosas que nosotros no podemos considerar como tales. Por ejemplo, si se tiene una variable de cadena llamada S, el C considera el valor del nombre S como una dirección de una cadena —no como una cadena—. Si S es una variable de cadena e I una variable entera, el llamar a una subrutina con esta instrucción:

```
SUBR (S,I)
```

causará que los valores de S y de I sean situados en el *stack*; el valor de I es un entero, pero el valor de S es una dirección (de un valor de cadena).

En el C es posible obligar a pasar una dirección prefijando el nombre de la variable con `&`; de este modo:

```
SUBR (S,&I)
```

pasaría la dirección de la variable I. En este ejemplo, la subrutina SUBR recibiría dos direcciones, cada una señalando a un valor. Si se cometiese el fallo de escribir `SUBR (&S, &I)`, se estaría pasando la dirección de la dirección de una cadena —que probablemente no es algo que se deba hacer.

Los novatos con el lenguaje C pueden quedar confundidos fácilmente por el paso de parámetros. En mi experiencia, es una de las mayores fuentes de errores en el uso de interfaces de lenguaje ensamblador con el C. Use el sentido común, y pruebe sus rutinas a

fondo para estar seguro de que lo que está ocurriendo realmente con los parámetros de conexión es lo que piensa que está ocurriendo.

En comparación con el Pascal, el uso de & en el C antes de un parámetro de subrutina es equivalente a declarar el parámetro VAR en Pascal: provoca que se pase al *stack* una dirección y no un valor.

A los parámetros se accede desde el *stack*, y los valores de los parámetros se modifican en la manera acostumbrada mediante las direcciones de parámetros situadas en el *stack*. La principal peculiaridad que se debe explicar en cuanto a los interfaces específicos con C es que las direcciones en el *stack* serán diferentes de las que se encuentran normalmente. Ya que se accede a las subrutinas con una llamada NEAR en vez de con una llamada FAR, el parámetro más próximo en el *stack* estará en la dirección [BP + 4] en vez de la más acostumbrada [BP + 6]; también, como se ha mencionado antes, el parámetro más próximo será el último escrito, en vez del primero.

Para proporcionar un ejemplo concreto del acceso y modificación de parámetros, considérese una subrutina que es llamada con dos enteros; de este modo:

SUBR (&I,J)

y cuya labor es situar el valor del segundo parámetro en el primero, haciendo la subrutina el equivalente a la sentencia de asignación $I = J$. El código de trabajo para hacer esto sería como sigue:

MOV	AX,[BP+6]	; obtener el valor de J
MOV	BX,[BP+4]	; obtener la dirección de I
MOV	[BX],AX	; poner el valor de J en I

En el C, todas las subrutinas normalmente devuelven un valor entero, incluso cuando un valor de retorno no tiene significado para el usuario. El valor puede ser usado o puede ser ignorado (de nuevo el C es peculiar con respecto a esto —muchos otros lenguajes hacen una clara distinción entre subrutinas que devuelven o no devuelven un valor—). Siguiendo la convención usual, el C espera este valor de retorno en el registro AX. Si se tiene que devolver un valor, se sitúa en AX: si no, y si quiere ser meticuloso, podría poner AX a cero antes de regresar, para asegurar que se devuelve un valor consistente, no accidental.

UN COMENTARIO FINAL

Aunque éste ha sido un breve análisis de un número muy pequeño de lenguajes de programación, esperamos que haya sido de ayuda el

exponer una perspectiva de sus características. Comparar los méritos de cada lenguaje usado en la familia PC sería ideal, pero imposible en un libro de esta naturaleza. Sin embargo, puede encontrar que los criterios que usamos para comparar estos cinco lenguajes pueden ayudarle al examinar otros lenguajes.



Controladores de los dispositivos periféricos

Perspectiva general 430

El controlador ANSI 431

El control de pantalla del ANSI 433

El control de teclado del ANSI 433

Los pros y los contras del controlador ANSI 434

Dos características que fueron introducidas con la versión DOS-2 necesitan una explicación especial: el tema de los controladores de los dispositivos periféricos instalables en general, y del controlador ANSI (también llamado ANSI.SYS) en particular. Estos temas están relacionados por su introducción común en la versión 2.00 del DOS y por el hecho de que el controlador ANSI es en sí un controlador de dispositivos instalables, pero son temas radicalmente diferentes desde la perspectiva de la programación. Comenzaré tratando los controladores de dispositivos en general y después pasaré a explicar el controlador ANSI con más detalle.

PERSPECTIVA GENERAL

El DOS tiene la capacidad interna de trabajar con los dispositivos más comunes, como unidades de disco ordinarias, líneas de comunicación, impresoras y, por supuesto, el teclado y la pantalla. Sin embargo, se pueden conectar al ordenador muchas otras clases de dispositivos. Todo lo que estas conexiones necesitan es algún apoyo de *software* adicional —llamado controlador de dispositivos— que les conecte con el DOS, o con programas del DOS.

Desde el lanzamiento de la versión 2.00, el DOS ha sido capaz de incorporar en sus propias operaciones cualquier controlador de dispositivos que siga un juego de reglas estándar de integración. El programa del controlador de dispositivos se lee y se integra al DOS durante el proceso arranque del DOS. Un fichero de disco llamado CONFIG.SYS le dice al DOS dónde hay un controlador de dispositivos que cargar. El nombre y la posición del fichero del controlador de dispositivos es identificada por la línea de comando `DEVICE = designación del fichero` en el CONFIG.SYS. Por cada uno de estos comandos, el DOS localiza el fichero del programa, lo carga en la memoria, y sigue los pasos necesarios para dar la bienvenida al controlador de dispositivos en el redil del DOS.

Típicamente, un controlador de dispositivos apoya a un nuevo tipo de dispositivo en una manera antigua. Por ejemplo, un controlador de dispositivos que soporte a una unidad de disco cuyos comandos de control sean nuevos para el DOS, pero cuyas características generales sean similares a otros tipos de unidad de disco, seguirá probablemente la forma de programa asentada por sus predecesores más comunes. Asimismo, un controlador de dispositivos que soporte un ratón, o un *joy-stick*, puede tratarlos como dispositivos de tipo teclado.

Por otro lado, los controladores de dispositivos pueden realizar

funciones que tienen poco o nada que ver con la adición de nuevos dispositivos de *hardware* al ordenador; tome como ejemplo el controlador de dispositivos ANSI, del que hablaré en la siguiente sección. El controlador de dispositivos ANSI no añade al ordenador nuevo *hardware*; en vez de esto, modifica la operación del *hardware* estándar del ordenador (el teclado y la pantalla).

Todos los detalles técnicos de la escritura de un controlador de dispositivos realmente pertenecen a un libro especializado en programación de sistemas del DOS, pero se pueden proporcionar aquí los puntos principales.

El fichero del controlador de dispositivos en sí tiene casi un formato de programa estándar con alguna información identificada específica del controlador. Hay dos tipos de controladores de dispositivos: aquellos para dispositivos de caracteres, los cuales, como el teclado, la pantalla, la impresora o el puerto de comunicaciones, trabajan con una corriente en serie de caracteres, y aquéllos para dispositivos de bloques, los cuales, como la unidad de disco, leen y escriben bloques de datos aleatorios identificados por alguna forma de direccionamiento de bloque. Los dispositivos de caracteres se identifican por sus propios nombres (similares a los nombres LPT1: o COM1:) y pueden ser tratados como ficheros. Los dispositivos de bloques se identifican por una letra de la unidad que el DOS les asigna y que son idénticas a las letras identificadoras de las unidades de disco A, B, C, etc.

El fichero del programa del controlador de dispositivos debe proporcionar al DOS varias direcciones de puntos de entrada desde donde el controlador será llamado para varios propósitos: para inicialización, para atender las peticiones de comando y para realizar trabajos de control. El programa del controlador debe estar preparado para manipular un juego de comandos estándar que el DOS les pide a todos los controladores que realicen, y para informar del estado del dispositivo al DOS.

La escritura de un controlador de dispositivos es similar a la escritura de los programas del BIOS que están en el corazón del DOS y que son el corazón de la ROM-BIOS incorporados del ordenador. Está entre la programación más sofisticada e intrincada que se puede realizar.

EL CONTROLADOR ANSI

Un ejemplo de un controlador de dispositivos periféricos que viene como una parte opcional del DOS es el controlador ANSI, un programa que mejora la manipulación de la entrada desde el teclado

y de la salida por la pantalla. Para nuestras versiones IBM del DOS, el controlador ANSI está activo solamente cuando se introduce deliberadamente en el DOS a través del fichero CONFIG.SYS, que el DOS carga durante la operación de arranque. El comando específico del fichero CONFIG.SYS, que se usa para activar el controlador ANSI, es:

`DEVICE = ANSI.SYS`

Merece la pena señalar que, mientras que el controlador ANSI es una parte opcional de las versiones IBM del DOS, el controlador es una parte integral del DOS en algunos ordenadores que son similares (pero no totalmente compatibles) con la familia del IBM PC. Por ejemplo, el ordenador profesional de Texas Instruments (comúnmente llamado TI Pro) y el ordenador personal avanzado NEC III (el NEC APC III) incluyen automáticamente el controlador ANSI en sus versiones DOS-2.

El controlador ANSI controla las salidas por la pantalla y la entrada desde el teclado que pasan los servicios estándar del DOS de la pantalla y del teclado. (Es importante advertir que cualquier dato del teclado o de la pantalla que evita el contacto con el DOS no es visto ni procesado nunca por el controlador ANSI.)

En el control de la salida por la pantalla, el controlador ANSI busca códigos especiales que identifiquen los comandos del controlador. El controlador toma nota en todo lo que reconoce como comando y entonces lo borra para que los códigos de comandos especiales no aparezcan en la pantalla. En efecto, este aspecto del controlador actúa como un conmutador de dos estados: inspecciona la salida que se dirige a la pantalla y pasa cualquier cosa que no sea un comando del controlador, a la vez que pasa todo lo que sea un comando del controlador a su propia parte de procesamiento de comandos.

Los comandos del controlador ANSI se identifican por un código especial de 2 bytes: el primer byte es el carácter «escape», 1B hex o CHR\$(27), y el segundo es el carácter de corchete izquierdo [, 5B hex o CHR\$(91). Siguiendo a estos bytes de identificación, están los parámetros de comando y, finalmente, el código de comando en sí. Los parámetros de comando son o números (en la forma de caracteres numéricos ASCII interpretados como dígitos decimales) o cadenas de caracteres ASCII encerradas entre comillas; de esta forma, «un parámetro de cadena». Si hay más de un parámetro, están separados por puntos y comas. El código de comando en sí, que completa el comando del controlador ANSI, es siempre un único carácter alfabético. Importa si la letra es mayúscula o minúscula; por

ejemplo, la h minúscula es un comando, y la H mayúscula es otro completamente diferente.

Para mostrar a qué se parecen estos comandos, aquí hay dos ejemplos, uno simple y otro complejo (el asterisco representa al carácter escape, 1B hex):

```
*[1C
```

```
*[65;32;66;"Remapeado B"p
```

El controlador ANSI reconoce un gran número de comandos, pero están todos divididos en dos amplias categorías: comandos de control de pantalla y comandos de traducción del teclado. Veamos primero al control de pantalla.

El control de pantalla del ANSI

Aunque los servicios BIOS del PC permiten mover el cursor a cualquier lugar de la pantalla y hacer otras cosas que nos dan un completo control de la pantalla, los servicios estándar del DOS no nos lo permiten. De hecho, los servicios de salida de la pantalla del DOS están completamente orientados a la salida TTY (o de teletipo) —salida que solamente abarca la clase de cosas que se puede hacer con una impresora—. Esto, por supuesto, ignora el potencial más rico de una pantalla. Esta escasez de salida de pantalla en el DOS es la que obliga a muchos programas a evitar los servicios del DOS y usar los servicios de más bajo nivel, como los servicios de la BIOS.

Los comandos de control de la pantalla del controlador ANSI remedian esta situación, proporcionando un juego de comandos para la pantalla, que pueden usarse para hacer casi todo lo que la pantalla puede hacer. Los comandos incluyen movimiento del cursor, borrado de la pantalla, ajuste de atributos de la pantalla (color, subrayado, parpadeo, etc.) y el cambio de modo de texto a gráficos, y viceversa. Como máximo nivel de sofisticación, hay comandos que pueden salvar la posición actual del cursor para que el cursor pueda moverse para mostrar información y que luego regrese a su posición inicial.

El control de teclado del ANSI

El otro tipo de comando aceptado por el controlador ANSI es un comando de traducción del teclado. Cuando se le da uno de estos comandos al controlador, el controlador vigila la entrada del teclado y reemplaza un carácter de tecla por otro carácter simple, o incluso una cadena de caracteres. Esto permite al controlador ANSI actuar

como un sustituto tosco pero efectivo de los programas de mejora del teclado populares, como ProKey (☛ véase página 145).

Advierta que estos dos tipos de comandos del controlador ANSI son muy diferentes en su propósito y su uso, pero los dos se pasan al controlador del mismo modo mediante una corriente de caracteres de salida a la pantalla.

Los pros y los contras del controlador ANSI

Como ve, hay dos maneras de mirar a los comandos del controlador ANSI: desde la perspectiva del usuario, que puede usar el controlador ANSI para realizar unos pocos trucos provechosos, y desde la perspectiva del programador, que puede usarlo como una ayuda para el desarrollo de programas. Este es un libro de programadores, pero permítanme comentar brevemente los trucos comunes del usuario con el controlador ANSI.

Todo lo que sé es que los usuarios, muy a menudo, consideran el controlador ANSI como un ProKey de pobres y como un mejorador del *prompt* del DOS. Usando los comandos de traducción del teclado, como mencioné antes, es posible imitar toscamente al programa ProKey. Generalmente, los comandos del teclado se activan situándolos en un fichero de texto y enviándolos a la pantalla (y, por tanto, al controlador ANSI) con el comando TYPE. Insertando los comandos del controlador ANSI en la cadena de *prompt* es posible lanzar el *prompt* del DOS a la cuarta dimensión. Podría mover el cursor a lo alto de la pantalla, mostrar la fecha y la hora en inverso y después devolver el cursor a su posición normal, o —para colmo de lujos— incluso borrar la pantalla y generar un menú completo. Las posibilidades son innumerables (y también, en mi opinión, un poco tontas).

Desde el punto de vista del programador, el controlador ANSI parece algo bastante diferente. El uso del controlador proporciona al programador dos beneficios principales, los cuales pueden ser bastante importantes. Para los programadores que no tengan las habilidades y las herramientas necesarias para construir interfaces en lenguaje ensamblador con los servicios de la BIOS, el controlador ANSI hace disponible a cualquier lenguaje de programación los servicios tipo BIOS más cruciales. Además puede ser de gran beneficio para los programadores que quieran escribir programas que no estén vinculados a la familia de los PC y que funcionen con cualquier ordenador bajo DOS que use el controlador ANSI.

A pesar de estas ventajas aparentes, yo creo que usar generalmente comandos del controlador ANSI en los programas no es buena idea. Por una razón concreta, requiere que el controlador ANSI esté

instalado en cualquier ordenador en el que se usen los programas, lo cual aumenta considerablemente las instrucciones que se tienen que preparar para acompañar a los programas. Es bastante difícil el intentar explicar la organización y el uso de los programas tanto a los novatos como a los expertos, sin añadir bases extras de complejidad, tal como la explicación de cómo instalar el controlador ANSI.

Un argumento más contra el uso del controlador ANSI es que no está disponible en todas las circunstancias. Por ejemplo, el sistema de ventanas de IBM Topview no se apoya en las características del controlador ANSI, así que los programas que requieren el controlador no pueden ser usados con el Topview. Esto también puede resultar cierto con otros entornos.

Pero lo más importante de todo es el hecho de que, comparado con otros métodos que están disponibles, el controlador ANSI es patéticamente lento al generar una salida de pantalla completa. Usted puede ver la comparación de la velocidad relativa del controlador ANSI, los servicios BIOS del PC y la salida de pantalla directa desde la memoria que se obtiene con el programa NU, que está en la versión 3 de las *Norton Utilities*. El programa NU contiene tres controladores de pantalla que usan estos tres métodos de salida. Si los prueba todos, verá rápidamente lo lento que es el controlador ANSI. A no ser que haya que mostrar pocas salidas de pantalla, el controlador ANSI es demasiado lento para ser satisfactorio.

B

Aritmética hexadecimal



Los bits y la notación hexadecimal 439

Direcciones segmentadas y notación hexadecimal 440

La conversión decimal-hexadecimal 441

Uso del BASIC para la aritmética hexadecimal 443

La suma hexadecimal 445

La multiplicación hexadecimal 446

Los números hexadecimales surgen en el trabajo de los ordenadores por la sencilla razón de que todo lo que hace un ordenador está basado en los números binarios, y la notación hexadecimal es una manera conveniente de representar los números binarios.

Los números hexadecimales se construyen sobre una base de 16, igual que los números decimales ordinarios se construyen sobre una base de 10; la diferencia estriba en que los números hexadecimales se escriben con 16 símbolos, mientras que los números decimales se escriben con diez (del 0 al 9). (De ahora en adelante usaré los términos «hexadecimal» y «hex» indistintamente). En la notación hex, los diez símbolos del 0 al 9 se usan para representar diez valores, del cero al nueve, y los seis valores restantes, del once al quince, se representan con los símbolos desde el A hasta el F. (Véase figura B-1.)

Los dígitos hex desde el A hasta el F se suelen escribir con letras mayúsculas, pero también se pueden ver con las letras minúsculas desde la a hasta la f; el significado es el mismo.

Los números hex se construyen con dígitos hex del mismo modo en que se construyen los números en base 10. Por ejemplo, cuando se escribe el número en base 10 123, queremos decir:

$$\begin{array}{rcl} 1 \text{ vez} & 100 & (10 \text{ veces } 10) \\ + 2 \text{ veces} & 10 & \\ + 3 \text{ veces} & 1 & \end{array}$$

Si se utilizan los símbolos 123 como un número hex, se quiere decir:

$$\begin{array}{rcl} 1 \text{ vez} & 256 & (16 \text{ veces } 16) \\ + 2 \text{ veces} & 16 & \\ + 3 \text{ veces} & 1 & \end{array}$$

No parece haber una forma estándar de escribir números hex, y se pueden encontrar expresados de forma diferente en sitios diferen-

Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec
0	Cero	4	Cuatro	8	Ocho	C	Doce
1	Uno	5	Cinco	9	Nueve	D	Trece
2	Dos	6	Seis	A	Diez	E	Catorce
3	Tres	7	Siete	B	Once	F	Quince

Figura B-1. El valor decimal de los dieciséis dígitos hexadecimales

tes. El BASIC usa el prefijo &H para identificar a los números hex, y esta notación se usa a veces en otros sitios también. Ocasionalmente se usa el prefijo # o 16#, pero un número hex es más a menudo seguido simplemente por una H. Sin embargo, la forma más común de expresar números hex, especialmente en la información de referencia, es sin ninguna notación especial. Se espera que usted comprenda en el contexto cuando se escribe un número en decimal y cuando en hexadecimal. Cuando vea un número en cualquier información de referencia técnica que parezca ser un número decimal, compruébelo cuidadosamente: puede estar realmente en hex. En este libro hemos identificado generalmente los números hexadecimales, añadiéndoles detrás la palabra hex.

Cuando necesite trabajar con números hex, puede usar el BASIC como ayuda (☛ véase página 443) o puede trabajar con ellos a mano. Sea cual sea el método que elija, encontrará las tablas de conversión y aritméticas situadas al final de este apéndice. Pero antes de que llegue a las tablas, explicaré primero por qué los números hex y los números binarios son tan compatibles. Después describiremos uno de los usos más comunes de los números hex en la programación: las direcciones segmentadas.

LOS BITS Y LA NOTACION HEXADECIMAL

Los números hex se usan primariamente como abreviatura de los números binarios con los que trabaja nuestro ordenador. Cada dígito hex representa cuatro bits de información binaria (☛ véase figura B-2). En el sistema de numeración binario (en base 2), un número de 4 bits puede tener 16 combinaciones diferentes, así que la única manera de representar cada número de 4 bits con un único dígito es usar un sistema de numeración en base 16, que es por lo que la aritmética hex es usada con nuestros ordenadores (☛ véase figura B-3).

Hex	Bits	Hex	Bits	Hex	Bits	Hex	Bits
0	0 0 0 0	4	0 1 0 0	8	1 0 0 0	C	1 1 0 0
1	0 0 0 1	5	0 1 0 1	9	1 0 0 1	D	1 1 0 1
2	0 0 1 0	6	0 1 1 0	A	1 0 1 0	E	1 1 1 0
3	0 0 1 1	7	0 1 1 1	B	1 0 1 1	F	1 1 1 1

Figura B-2. La combinación de los bits para cada uno de los dieciséis dígitos hex

Bit	Palabra	Byte	Valor	
			Dec	Hex
0	1	1	1	1
1	1.	1.	2	2
2	1..	1..	4	4
3	1...	1...	8	8
4	1....	1....	16	10
5	1.....	1.....	32	20
6	1.....	1.....	64	40
7	1.....	1.....	128	80
8	1.....		256	100
9	1.....		512	200
10	1.....		1 024	400
11	1.....		2 048	800
12	1.....		4 096	1000
13	1.....		8 192	2000
14	1.....		16 384	4000
15	1.....		32 768	8000

Figura B-3. Las equivalencias entre decimal y hexadecimal, de cada bit en un byte y de cada bit en una palabra de 2 bytes

Cuando esté usando palabras de 2 bytes, recuerde el orden inverso o «palabras inversas» en el que se almacenan en memoria. Véase capítulo 3, página 31.

DIRECCIONES SEGMENTADAS Y NOTACION HEXADECIMAL

Uno de los usos más comunes de los números hex es el direccionamiento de la memoria. Puede recordar de los capítulos 2 y 3 que una dirección completa es de 20 bits o de 5 dígitos hex de anchura, ya que el microprocesador 8088 del PC solamente puede trabajar con números de 16 bits, las direcciones están separadas en dos palabras de 16 bits, llamadas segmento y dirección relativa. Las dos partes se escriben juntas, como 1234:ABCD. La parte de segmento siempre se escribe primero, y ambas partes se dan en hex.

La parte de segmento de una dirección es tratada como si estuviera multiplicada por 16, lo que es lo mismo que si estuviera un hex 0 extra escrito después de ella. Las dos partes, sumadas juntas, producen la dirección de 20 bits real que ellas representan. Por

ejemplo, la dirección segmentada 1234:ABCD se convierte en una dirección completa de este modo.

$$\begin{array}{rcccccc} & 1 & 2 & 3 & 4 & 0 & \text{(advierta el cero añadido a la derecha)} \\ + & & A & B & C & D & \\ \hline & 1 & C & F & 0 & D & \end{array}$$

Si se necesita calcular la dirección real a la que se refiere una dirección segmentada, siga esta fórmula. Las tablas de suma de la página 445 también pueden ayudarle.

□ **NOTA:** *Sepa que la misma dirección real puede estar representada por muchas direcciones segmentadas diferentes.*

No hay ninguna forma buena de separar una dirección real en su formato segmentado. Una manera simple es tomar el primer dígito de la dirección real, seguida de tres ceros como la parte de párrafo de segmento, y los cuatro dígitos restantes como la parte relativa. Siguiendo esta regla, la dirección de arriba, 1CF0D, sería separada como 1000:CF0D. El listado de la ROM-BIOS de IBM en el «Manual de Referencia Técnica» sigue esta convención, así que todas las direcciones relativas que aparecen allí tienen la parte oculta de segmento de F000.

Cuando esté trabajando con direcciones segmentadas reales, la parte de segmento representará los contenidos reales de los registros segmento y podrá señalar a cualquier parte de la memoria. Las direcciones relativas varían con el uso. Las direcciones relativas del código, que indican las posiciones de programa, generalmente comienzan con hex 100, ya que se reservan 256 (100 hex) bytes antes de cada programa para su prefijo de segmento de programa. Las direcciones relativas de datos, usados para los datos, generalmente empiezan desde 0. Las direcciones relativas de la pila son números grandes generalmente, ya que la pila trabaja hacia atrás desde el final del segmento de la pila.

Para ver un ejemplo vivo del tipo de direcciones segmentadas que están en uso cuando se ejecuta un programa, ejecute el programa DEBUG. Cuando comienza DEBUG, generará una petición de comando. Cuando introduzca como comando una letra D, el DEBUG mostrará parte de la memoria. Las direcciones que se muestran a la izquierda son el modo en que aparecen típicamente las direcciones segmentadas.

LA CONVERSION DECIMAL-HEXADECIMAL

Las tablas de la página siguiente muestran el equivalente decimal de cada dígito hex en las primeras cinco posiciones de un número, lo

que cubre el espacio de dirección completa aritmética usado en el PC. Como demostraré, estas tablas pueden ser usadas para la conversión entre números hexadecimales y decimales.

Primera posición				Segunda posición			
Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec
.... 0	0	 8	8	... 0 .	0	... 8 .	128
.... 1	1	 9	9	... 1 .	16	... 9 .	144
.... 2	2	 A	10	... 2 .	32	... A .	160
.... 3	3	 B	11	... 3 .	48	... B .	176
.... 4	4	 C	12	... 4 .	64	... C .	192
.... 5	5	 D	13	... 5 .	80	... D .	208
.... 6	7	 E	14	... 6 .	96	... E .	224
.... 7	7	 F	15	... 7 .	112	... F .	240

Tercera posición				Cuarta posición			
Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec
.. 0 ..	0	.. 8 ..	2 048	. 0 ...	0	. 8 ...	32 768
.. 1 ..	256	.. 9 ..	2 304	. 1 ...	4 096	. 9 ...	36 864
.. 2 ..	512	.. A ..	2 560	. 2 ...	8 192	. A ...	40 960
.. 3 ..	768	.. B ..	2 816	. 3 ...	12 288	. B ...	45 056
.. 4 ..	1 024	.. C ..	3 072	. 4 ...	16 384	. C ...	49 152
.. 5 ..	1 280	.. D ..	3 328	. 5 ...	20 480	. D ...	53 248
.. 6 ..	1 536	.. E ..	3 584	. 6 ...	24 576	. E ...	57 344
.. 7 ..	1 792	.. F ..	3 840	. 7 ...	28 672	. F ...	61 440

Quinta posición			
Hex	Dec	Hex	Dec
0	0	8	524 288
1	65 536	9	589 824
2	131 072	A	655 360
3	196 608	B	720 896
4	262 144	C	786 432
5	327 680	D	851 968
6	393 216	E	917 504
7	458 752	F	983 040

Figura B-4. El equivalente decimal de cada posición del dígito hexadecimal

Estas tablas pueden ser usadas para convertir un número hex en un número decimal del siguiente modo. Usaré el número A1B2 hex como ejemplo. Mire cada dígito hex en la tabla correspondiente a su posición y después sume los valores decimales de este modo:

2 en la primera posición es	2
B en la segunda posición es	176
1 en la tercera posición es	256
A en la cuarta posición es	40 960
El total es:	41 394

Para convertir un número decimal en *hex* usando estas tablas es igual de fácil de hacer, pero un poco más complicado de describir. Otra vez, las cosas se aclararán si trabajo con un ejemplo. Usaré el número decimal 1 492.

Trabajo con la tabla desde la quinta posición hasta la primera de la tabla. En la tabla de la quinta posición encuentro el dígito hex mayor con un valor que no sea superior a 1 492, escribo el dígito hex, resto su valor decimal de 1 492, y continúo con la siguiente tabla y el nuevo valor (esto es, la diferencia después de la resta). Voy de tabla en tabla hasta que el número que quede sea cero. El proceso se muestra en la figura B-5. El resultado es 005D4, o 5D4 sin los ceros a la izquierda.

Posición	Máximo dígito hex	Valor decimal	Número decimal restante
Comienzo			1 492
5	0	0	1 492
4	0	0	1 492
3	5	1280	212
2	D	208	4
1	4	4	0
Resultado	005D4		

Figura B-5. Conversión del número decimal 1,492 en un número hexadecimal

USO DEL BASIC PARA LA ARITMETICA HEXADECIMAL

Una manera sencilla de manipular números hex es dejar que el BASIC haga el trabajo. Para hacer esto, active el intérprete del BASIC e introduzca cualquier operación que quiera realizar usando el modo de comando (sin números de línea).

Para mostrar el equivalente decimal de un número hex, tal como 1234 hex, puede hacer simplemente lo siguiente:

```
PRINT &H1234
```

Esté seguro de prefijar cualquier número hex con &H para que el BASIC sepa que es un número hex. Para obtener una exposición mejor de los números decimales, particularmente los números grandes, use el formato PRINT USING; así:

```
PRINT USING "###,###,###"; &H1234
```

Para mostrar el equivalente de un número decimal, tal como 1,234, puede hacer simplemente lo siguiente:

```
PRINT HEX$( 1234 )
```

Los ejemplos hasta ahora han usado solamente constantes decimales y hex. Con la misma facilidad podemos hacer que el BASIC realice algunas operaciones aritméticas y muestre el resultado en decimal, o hexadecimal. Aquí hay dos ejemplos:

```
PRINT USING "###,###,###"; &H1000 - &H3A2 + 16 * 3
PRINT HEX$(1776 - 1492 + &H100)
```

El uso de variables para conservar los resultados calculados puede evitarnos el tener que retectar una expresión, o un número complicado. Las variables que conservan números hexadecimales deberán estar siempre escritas como variables de doble precisión (con un # al final del nombre variable) para que obtengamos la máxima exactitud. Por ejemplo:

```
X# = 1776 - 1492 + &H100
PRINT USING "###,###,###"; X#, 2 * X#, 3 * X#
```

LA SUMA HEXADECIMAL

Para sumar números hexadecimales, trabajamos dígito a dígito, como hacemos con números decimales. Para hacerlo más fácil, use la figura B-6, que calcula la suma de dos dígitos hex cualesquiera. Para usar esta tabla, encuentre la fila de un dígito hexadecimal y la columna del otro. El número hex situado en la intersección de la fila y la columna es la suma de los dos dígitos.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1		2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
2			4	5	6	7	8	9	A	B	C	D	E	F	10	11
3				6	7	8	9	A	B	C	D	E	F	10	11	12
4					8	9	A	B	C	D	E	F	10	11	12	13
5						A	B	C	D	E	F	10	11	12	13	14
6							C	D	E	F	10	11	12	13	14	15
7								E	F	10	11	12	13	14	15	16
8									10	11	12	13	14	15	16	17
9										12	13	14	15	16	17	18
A											14	15	16	17	18	19
B												16	17	18	19	1A
C													18	19	1A	1B
D														1A	1B	1C
E															1C	1D
F																1E

Figura B-6. Adición de dos números hexadecimales

LA MULTIPLICACION HEXADECIMAL

Para multiplicar números hex, trabajamos dígito a dígito, como hacemos con números decimales. Para hacerlo más fácil, use la figura B-7, que calcula la suma de dos dígitos hex cualesquiera. Para usar la tabla, encuentre la fila de un número hex y la columna del otro. El número hex situado en la intersección de la fila y la columna es el producto de dos dígitos.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1		1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2			4	6	8	A	C	E	10	12	14	16	18	1A	1C	1E
3				9	C	F	12	15	18	1B	1E	21	24	27	2A	2D
4					10	14	18	1C	20	24	28	2C	30	34	38	3C
5						19	1E	23	28	2D	32	37	3C	41	46	4B
6							24	2A	30	36	3C	42	48	4E	54	5A
7								31	38	3F	46	4D	54	5B	62	69
8									40	48	50	58	60	68	70	78
9										51	5A	63	6C	75	7E	87
A											64	6E	78	82	8C	96
B												79	84	8F	9A	A5
C													90	9C	A8	B4
D														A9	B6	C3
E															C4	D2
F																E1

Figura B-7. Multiplicación de dos números hexadecimales

C

Sobre los caracteres



El juego de caracteres estándar y ampliado 448

El formato de los caracteres 451

Los primeros treinta y dos caracteres ASCII 453

Los caracteres de dibujo de cajas 454

Los caracteres gráficos y de bloques 454

Convenciones de formateo de los ficheros de texto 456

Formatos de los ficheros de texto ordinarios 456

Formatos de texto de los procesadores de texto 457

Hay 256 caracteres distintos usados por la familia de ordenadores personales de IBM. Tienen códigos de bytes numéricos con valores variables entre 0 y 255. A veces se refiere a estos caracteres por su valor numérico usando la notación BASIC de CHR\$(0) hasta CHR\$(255). Los primeros 128 caracteres desde CHR\$(0) hasta CHR\$(127) son los verdaderos caracteres estándar ASCII. Los últimos 128 caracteres, desde CHR\$(128) hasta CHR\$(255), son caracteres especiales que componen un juego de caracteres ampliado.

Generalmente, los ordenadores tratan los caracteres ASCII verdaderos de una manera similar, aunque hay alguna variedad en la forma en que son usados los primeros 32 caracteres. (● Véase página 453 para una explicación de estos caracteres.) El fabricante de ordenadores decide cómo usar los 128 caracteres especiales.

Afortunadamente, todos los modelos de ordenadores personales de IBM usan el mismo juego de caracteres ASCII ampliado. Los ordenadores que imitan prácticamente a los ordenadores personales de IBM usan también este juego, pero otros ordenadores suelen tener sus propios caracteres especiales. Esto es importante para tener en cuenta cuándo se convierten programas de otros ordenadores, o cuándo se escriben programas del PC que usted planea convertir para usar en otros ordenadores.

EL JUEGO DE CARACTERES ESTANDAR Y AMPLIADO

Aquí hay un programa BASIC que mostrará 256 caracteres con sus códigos numéricos en notación decimal y hexadecimal. Los caracteres también se listan en la figura C-1, en el apéndice C.

```

1000 ' mostrar todos los caracteres del PC
1010 '
1020 MONOCROMO = 1
1030 IF MONOCROMO THEN WW = 80 : HH = &HB000
      ELSE WW = 40 : HH = &HB800
1040 GOSUB 2000
1050 FOR I = 0 TO 255
1060 GOSUB 3000
1070 NEXT I
1080 PRINT "Hecho."
1090 GOSUB 6000
1095 SYSTEM
1999 '
2000 ' inicializar
2010 '
2020 DEF SEG = HH
2030 KEY OFF : CLS
2040 WIDTH WW : COLOR 14,1,1

```

' inicializa el registro DOS
' para todos los códigos de carácter
' muestra la información
' ajusta el registro DS para poke
' ajusta la pantalla

```

2050 FOR I = 1 TO 25 : PRINT : NEXT I
2060 PRINT "Mostrando todos los caracteres"
2070 GOSUB 5000 ' subencabezamiento periódico
2080 RETURN
2099 '
3000 ' mostrar la información del carácter
3010 '
3020 PRINT USING " ###      ";I;
3030 IF I < 16 THEN PRINT "0";
3040 PRINT HEX$(I);" ";
3050 POKE WW * 2 * 23 + 34, I ' insertar el carácter
3060 GOSUB 4000 ' imprimir cualquier comentario
3070 IF (I MOD 16) < 15 THEN RETURN ' pausa después de cada 16 caracteres
3080 GOSUB 6000
3090 IF I < 255 THEN GOSUB 5000
3100 RETURN
3997 '
3998 ' comentarios de los caracteres
3999 '
4000 IF I = 0 THEN PRINT "espacio en blanco";
4007 IF I = 7 THEN PRINT "pitido (campana)";
4008 IF I = 8 THEN PRINT "retroceso";
4009 IF I = 9 THEN PRINT "tabulador";
4010 IF I = 10 THEN PRINT "salto de línea";
4012 IF I = 12 THEN PRINT "salto de página";
4013 IF I = 13 THEN PRINT "retorno de carro";
4026 IF I = 26 THEN PRINT "fin de fichero de texto";
4032 IF I = 32 THEN PRINT "espacio en blanco verdadero";
4255 IF I = 255 THEN PRINT "muestra espacio en blanco";
4997 PRINT ' finaliza la línea
4998 RETURN
4999 '
5000 ' subencabezamiento periódico
5010 '
5020 COLOR 15
5030 PRINT
5040 PRINT
5050 PRINT "Decimal - Hex - Char - Comentarios"
5060 PRINT
5070 COLOR 14
5080 RETURN
5999 '
6000 ' pausa
6010 '
6020 IF INKEY$ <> "" THEN GOTO 6020
6030 PRINT
6040 COLOR 2
6050 PRINT "Pulse cualquier tecla para continuar..."
6060 COLOR 14
6070 IF INKEY$ = "" THEN GOTO 6070
6080 PRINT
6090 RETURN

```

Hay unas cuantas cosas que subrayar sobre este programa y los caracteres que muestra.

El programa está diseñado para ajustarse automáticamente al adaptador monocromo o color/gráficos, basado en el valor mostrado en la línea 1020; use el 1 (como está) para un adaptador monocromo, cambie el 1 por un 0 para un adaptador color/gráficos, o su

NUMERO, dec			NUMERO, hex			PANTALLA			NUMERO, dec			NUMERO, hex			PANTALLA			NUMERO, dec			NUMERO, hex			PANTALLA			NUMERO, dec			NUMERO, hex			PANTALLA		
0	00		32	20		64	40		96	60		128	80		160	A0		192	C0		224	E0													
1	01	␣	33	21	!	65	41	A	97	61	a	129	81	ü	161	A1	i	193	C1	L	225	E1	␣												
2	02	␣	34	22	"	66	42	B	98	62	b	130	82	ü	162	A2	ö	194	C2	T	226	E2	␣												
3	03	␣	35	23	#	67	43	C	99	63	c	131	83	ü	163	A3	ü	195	C3	t	227	E3	␣												
4	04	␣	36	24	\$	68	44	D	100	64	d	132	84	ü	164	A4	n	196	C4	-	228	E4	␣												
5	05	␣	37	25	%	69	45	E	101	65	e	133	85	ü	165	A5	n	197	C5	+	229	E5	␣												
6	06	␣	38	26	&	70	46	F	102	66	f	134	86	ü	166	A6	e	198	C6	f	230	E6	␣												
7	07	␣	39	27	'	71	47	G	103	67	g	135	87	ü	167	A7	e	199	C7	H	232	E7	␣												
8	08	␣	40	28	(	72	48	H	104	68	h	136	88	ü	168	A8	i	200	C8	E	232	E8	␣												
9	09	␣	41	29	)	73	49	I	105	69	i	137	89	ü	169	A9	r	201	C9	F	233	E9	␣												
10	0A	␣	42	2A	*	74	4A	J	106	6A	j	138	8A	ü	170	AA	1/2	202	CA	=	234	EA	␣												
11	0B	␣	43	2B	+	75	4B	K	107	6B	k	139	8B	ü	171	AB	1/2	203	CB	H	235	EB	␣												
12	0C	␣	44	2C	,	76	4C	L	108	6C	l	140	8C	ü	172	AC	1/2	204	CC	=	236	EC	␣												
13	0D	␣	45	2D	-	77	4D	M	109	6D	m	141	8D	ü	173	AD	1/2	205	CD	=	237	ED	␣												
14	0E	␣	46	2E	.	78	4E	N	110	6E	n	142	8E	ü	174	AE	1/2	206	CE	=	238	EE	␣												
15	0F	␣	47	2F	/	79	4F	O	111	6F	o	143	8F	ü	175	AF	1/2	207	CF	=	239	EF	␣												
16	10	␣	48	30	0	80	50	P	112	70	p	144	90	ü	176	B0	1/2	208	D0	=	240	F0	␣												
17	11	␣	49	31	1	81	51	Q	113	71	q	145	91	ü	177	B1	1/2	209	D1	=	241	F1	␣												
18	12	␣	50	32	2	82	52	R	114	72	r	146	92	ü	178	B2	1/2	210	D2	=	242	F2	␣												
19	13	␣	51	33	3	83	53	S	115	73	s	147	93	ü	179	B3	1/2	211	D3	=	243	F3	␣												
20	14	␣	52	34	4	84	54	T	116	74	t	148	94	ü	180	B4	1/2	212	D4	=	244	F4	␣												
21	15	␣	53	35	5	85	55	U	117	75	u	149	95	ü	181	B5	1/2	213	D5	=	245	F5	␣												
22	16	␣	54	36	6	86	56	V	118	76	v	150	96	ü	182	B6	1/2	214	D6	=	246	F6	␣												
23	17	␣	55	37	7	87	57	W	119	77	w	151	97	ü	183	B7	1/2	215	D7	=	247	F7	␣												
24	18	␣	56	38	8	88	58	X	120	78	x	152	98	ü	184	B8	1/2	216	D8	=	248	F8	␣												
25	19	␣	57	39	9	89	59	Y	121	79	y	153	99	ü	185	B9	1/2	217	D9	=	249	F9	␣												
26	1A	␣	58	3A	:	90	5A	Z	122	7A	z	154	9A	ü	186	BA	1/2	218	DA	=	250	FA	␣												
27	1B	␣	59	3B	;	91	5B	[	123	7B	{	155	9B	ü	187	BB	1/2	219	DB	=	251	FB	␣												
28	1C	␣	60	3C	=	92	5C	\	124	7C		156	9C	ü	188	BC	1/2	220	DC	=	252	FD	␣												
29	1D	␣	61	3D	>	93	5D	]	125	7D	}	157	9D	ü	189	BD	1/2	221	DD	=	253	FD	␣												
30	1E	␣	62	3E	>	94	5E	^	126	7E	~	158	9E	ü	190	BE	1/2	222	DE	=	254	FE	␣												
31	1F	␣	63	3F	?	95	5F	_	127	7F	Δ	159	9F	ü	191	BF	1/2	223	DF	=	255	FF	␣												

Figura C-1. El juego de caracteres de la familia IBM PC

equivalente. Dependiendo del valor de la línea 1020, el programa hace dos ajustes: uno que cambia donde la información del display es buscada (POKE) en la memoria de la pantalla, y otra que cambia la anchura de la pantalla en 40 y 80 columnas. Para un adaptador color/gráficos, se usa el modo de 40 columnas para mayor claridad. La sentencia POKE en la línea 3050 es la que provoca que los caracteres sean representados. Este paso extra es necesario, debido a que unos pocos caracteres no pueden ser expuestos mediante la sentencia ordinaria PRINT. ● Véase una explicación de esto en «Los primeros treinta y dos caracteres ASCII», en la página 453.

Cada uno de los 256 caracteres diferentes tiene su propio y único aspecto, excepto los dos caracteres CHR\$(0) y CHR\$(255), que aparecen igual, como el carácter espacio en blanco, CHR\$(32). (● Véase figura C-1.)

El formato de los caracteres

Todos los caracteres que aparecen en la pantalla están compuestos por puntos dibujados dentro de una red llamada caja del carácter. Hay dos redes estándar, una para el adaptador monocromo (y sus equivalentes) y otra para el adaptador color/gráficos (y sus equivalentes). En cada caso, los caracteres se crean llenando (o iluminando) los puntos apropiados en la red.

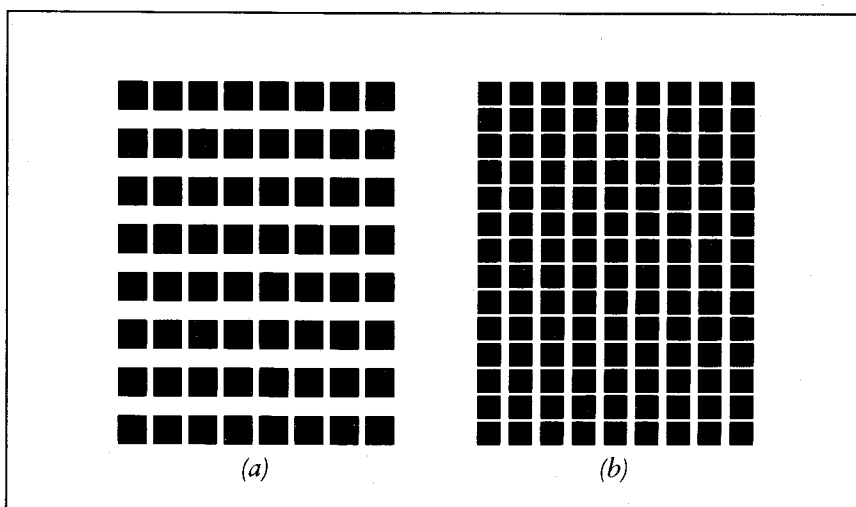


Figura C-2. Las muestras de la matriz de puntos producidas para (a) el adaptador color/gráficos y (b) para el adaptador monocromo

El adaptador color/gráficos usa una caja de carácter de 8 por 8 puntos; el adaptador monocromo utiliza una caja de 9×14 , que permite dibujar caracteres más definidos. La mayoría de los miembros de la familia PC no-IBM utilizan el formato color/gráficos, igual que el PCjr. Una excepción notable es el Compaq PC-compatible, que utiliza con éxito ambos formatos a la vez.

Las impresoras de matriz de puntos, tales como la impresora IBM Compact, también escriben los caracteres con una rejilla de puntos. De todas formas cada impresora puede tener su propia forma de dibujar los caracteres y no coincidir exactamente con los puntos de la pantalla. Pero por el contrario cuando se vuelca una pantalla a la impresora existe una correspondencia punto a punto.

Para ver cómo aparecen los caracteres, las tres matrices de puntos de la figura C-3 ilustran una Y, una y, y un punto y coma, usando la caja de carácter de 8 por 8.

Hay varias reglas que se aplican al dibujo de caracteres. Para caracteres ordinarios, las dos columnas de la derecha se dejan sin usar para proporcionar la separación entre caracteres. Estas dos columnas sólo se usan con caracteres que, se supone, llenan la caja del carácter entera, como el carácter de bloque sólido, CHR\$(219). Las dos filas superiores se usan sólo para los ascendentes (la parte de un carácter que está sobre la altura normal de un carácter). El espacio ascendente se usa para las letras mayúsculas y para las letras minúsculas, como *b*, *d* y *k*. La fila inferior se usa para descendentes (la parte de un carácter que cae bajo la línea), como en las letras minúsculas *g*, *e*, y. Estas líneas de guía generales quedan a veces comprometidas por un mejor resultado general. Por ejemplo, el punto y coma, nuestro tercer ejemplo en la figura C-3, se eleva una línea, de .o que cabría esperar y no usa la línea descendente.

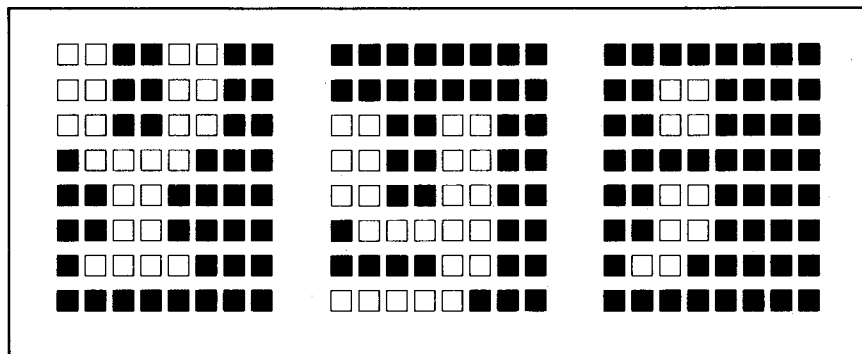


Figura C-3. El diseño de puntos de tres caracteres en una caja de carácter de 8 por 8

Bit								Valor (hex)
7	6	5	4	3	2	1	0	
1	1	0	0	1	1	0	0	CC
1	1	0	0	1	1	0	0	CC
1	1	0	0	1	1	0	0	CC
0	1	1	1	1	0	0	0	78
0	0	1	1	0	0	0	0	30
0	0	1	1	0	0	0	0	30
0	1	1	1	1	0	0	0	78
0	0	0	0	0	0	0	0	00

Figura C-4. La codificación de los 8 bytes de carácter del carácter Y

En los modos de exposición de gráficos desde 4 hasta el 6 y desde el 8 al 10 (modos de pantalla del 1 al 6 en BASIC) podemos crear nuestras propias tablas de dibujo de caracteres, usando las líneas de guía anteriores. Las tablas de dibujo de caracteres están codificadas con ocho bytes para cada carácter —un byte por cada columna en el dibujo—. Los ocho bits de cada byte indica qué puntos de la columna van a ser mostrados. Por ejemplo, el carácter Y se codifica en hex como CC CC CC 78 30 30 78 00. Los bits individuales de cada byte se muestran en la figura C-4 (mire de cerca y verá de nuevo el diseño de la Y).

Los primeros treinta y dos caracteres ASCII

Los primeros 32 caracteres ASCII, de CHR\$(0) a CHR\$(31), tienen dos usos importantes que parecen enfrentarse el uno con el otro. Por un lado, estos caracteres tienen significados ASCII estándar; se usan tanto como para el control de la impresora (p. ej., CHR\$(12) es el carácter de salto de línea), y para el control de comunicaciones. Por otro lado, IBM también los usa para algunos de los caracteres más interesantes y útiles, como los caracteres de los palos de la baraja (corazones, diamantes, tréboles y picas), desde CHR\$(3) hasta CHR\$(6).

Generalmente, todos los dispositivos de ordenador, incluyendo la impresora y la pantalla, actúan sobre el significado ASCII de los caracteres en vez de mostrar el dibujo del carácter. Una simple forma de demostrar esto es con el carácter zumbido/campana, CHR\$(7), que tiene un punto por dibujo. Si escribimos este carácter en la pantalla en BASIC, de este modo:

```
PRINT CHR$(7)
```

Carácter	Resultado	
	En BASIC	En muchos otros lenguajes
CHR\$(7)	Pitidos	Pitidos
CHR\$(8)	Aparece el carácter	Acción de retroceso
CHR\$(9)	Acción de tabulador	Acción de tabulador
CHR\$(10)	Acción de salto de línea y de retorno de carro	Acción de salto de línea
CHR\$(11)	Cursor a la esquina superior izquierda	Aparece el carácter
CHR\$(12)	Borrado de pantalla	Aparece el carácter
CHR\$(13)	Acción de retorno de carro	Acción de retorno de carro
CHR\$(28)	El cursor se mueve a la derecha	Aparece el carácter
CHR\$(29)	El cursor se mueve a la izquierda	Aparece el carácter
CHR\$(30)	El cursor se mueve hacia arriba	Aparece el carácter
CHR\$(32)	El cursor se mueve hacia abajo	Aparece el carácter

Figura C-5. Los resultados cuando se escriben ciertos caracteres usando diferentes lenguajes

el altavoz del PC pitará. Pero si ponemos directamente el carácter sobre la pantalla mediante un POKE en el *buffer* de la pantalla, de este modo:

```
DEF SEG = &HB800 : POKE 0, 7
```

aparecerá el dibujo del carácter. Siempre podemos hacer que los caracteres aparezcan en la pantalla con un POKE en el *buffer* de la pantalla. Sin embargo, el hacer POKE debe ser evitado siempre que sea posible, ya que hace que nuestro programa se endurezca para adaptarse a cambios. Es siempre una práctica mejor el mostrar la información en la pantalla usando métodos ordinarios, como la sentencia PRINT del BASIC.

Muchos de los 32 primeros caracteres pueden ser escritos en la pantalla, pero los caracteres representados pueden variar, dependiendo del lenguaje que se use. La figura C-5 muestra algunas de estas diferencias. Los caracteres que no se muestran, de CHR\$(0) a CHR\$(6) y de CHR\$(14) a CHR\$(27), siempre se pueden escribir en la pantalla con resultados pronosticables.

Los caracteres de dibujo de cajas

Entre los caracteres ASCII ampliados especiales más útiles están los caracteres que se diseñan para el dibujo de cajas de una línea y de dos líneas. Estos caracteres tienen los códigos desde CHR\$(179) hasta

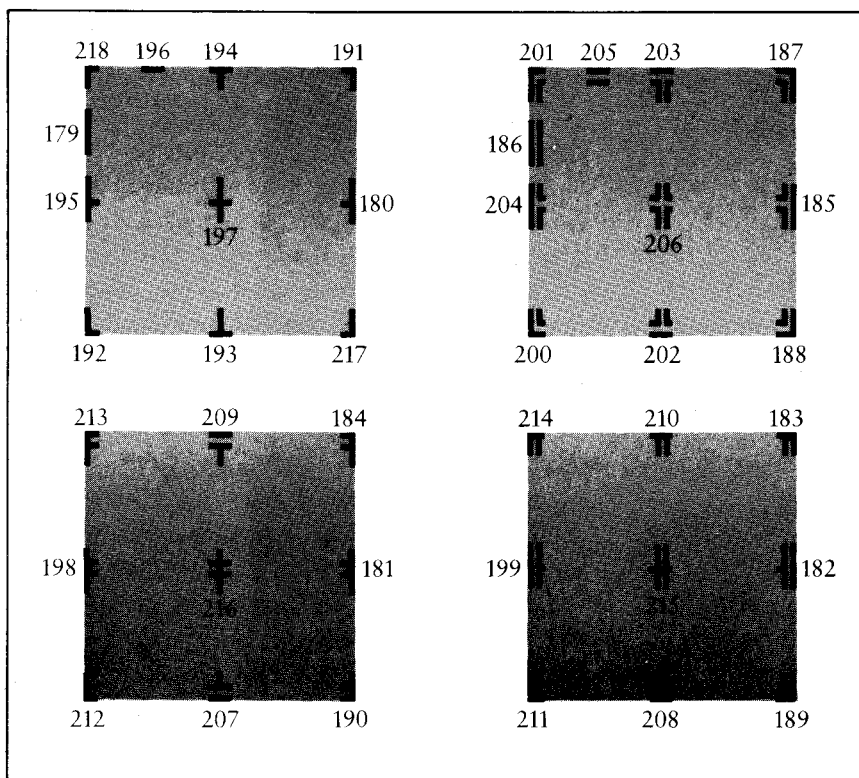


Figura C-6. Los caracteres de dibujo de cajas

CHR\$(218). Ya que son difíciles de combinar correctamente, usted puede encontrar información de ayuda en la figura C-6.

Los caracteres gráficos y de bloques

Además de los caracteres de diseño de cajas, hay dos series de caracteres diseñados para dibujos de gráficos y bloques. Una serie consta de cuatro caracteres que llena la caja de carácter, pero que

CHR\$(176)		CHR\$(220)	
CHR\$(177)		CHR\$(221)	
CHR\$(178)		CHR\$(222)	
CHR\$(219)		CHR\$(223)	

Figura C-7. Los dos juegos de caracteres de gráfica y de bloque

están oscurecidas en diferentes densidades. En los tres que no son sólidos, parte de los puntos del carácter están encendidos, o ajustados, al color del primer plano, mientras los restantes puntos están apagados, o ajustados, al color de fondo. La otra serie consta de caracteres de bloque que proporcionan un color sólido que cubre la mitad de la caja de carácter. El carácter sólido CHR\$(219) también se usa con estos medios caracteres.

CONVENCIONES DE FORMATEO DE LOS FICHEROS DE TEXTO

Muchos programas trabajan con ficheros de texto. Como resultado, muchos programadores han adoptado algunas convenciones de formato de ficheros de texto que hacen más fácil que los ficheros de datos sean usados por diferentes programas. Los formatos están definidos por caracteres insertados en el texto, que realizan funciones tales como retornos de carro, saltos de línea, o retrocesos, sólo por nombrar algunos.

Generalmente es buena idea que los programas sean muy tolerantes con los distintos formatos de texto. Es también importante diseñar programas que creen datos de texto con formatos simples, usando solamente unos pocos caracteres de formateo. Esto no es siempre posible con los programas de tratamiento de texto, que deben ser capaces de formatear el texto en la pantalla de diversas maneras. En esta sección discutiremos primero el tipo más ordinario de fichero de texto, y después seguiremos explicando ficheros de tratamiento de texto.

Formatos de los ficheros de texto ordinarios

Los ficheros de texto ordinarios están compuestos de caracteres ASCII estándar y no usan ninguno de los caracteres ASCII ampliados. En el esquema de codificaciones ASCII los primeros 32 caracteres, desde CHR\$(0) hasta CHR\$(31), tienen significados especiales: algunos de éstos se usan para el formateo de datos de texto y los otros se usan generalmente para el control de comunicaciones. Ya que estos caracteres se usan para controlar el formateo del texto, no necesitan aparecer en la pantalla; de hecho, es mejor si no aparecen, ya que suelen tener asociados dibujos bastante insólitos.

Sólo hay un puñado de caracteres de formateo que se usen extensamente en ficheros de texto ordinarios. Fueron desarrollados

originalmente como comandos para formatear una página de impresión. Ahora su uso se extiende a todos los dispositivos de salida. Trataremos cada uno de los caracteres de formateo por turno.

CHR\$(26) se usa para señalar el verdadero final de un fichero de texto. Este carácter puede ir antes del final del fichero, indicado por el tamaño del fichero en el directorio. Esto es porque muchos programas de tratamiento de textos típicamente leen y escriben ficheros, no byte a byte, sino en grandes trozos —de 128 bytes a la vez—. Cuando transfieren los datos a este ritmo, el DOS ve solamente el final del bloque de 128 bytes, y no reconoce el fin de fichero real delimitado por el carácter **CHR\$(26)**.

CHR\$(13) y **CHR\$(10)**. Normalmente dividen al fichero de texto en líneas marcando el final de cada línea con un retorno de carro (**CHR\$(13)**) y un salto de línea (**CHR\$(10)**), normalmente en este orden. Muchos programas de tratamiento de textos tienen dificultades con las líneas de más de 255 caracteres de longitud, y algunos están limitados a líneas de 80 caracteres.

Un retorno de carro puede ser usado por sí mismo. Desafortunadamente, esto puede ser interpretado como dos cosas: el fin de una línea con un salto de línea que está implícito y proporcionado automáticamente por algunas impresoras, o un retorno al principio de la línea de impresión actual, lo que provoca que la línea entera sea reimpressa. (El carácter de retroceso, **CHR\$(8)**, se usa también algunas veces para hacer que una impresora remarque un carácter.)

CHR\$(4), el carácter tab, se usa algunas veces para representar uno o más espacios, hasta la siguiente posición de ajuste del tabulador. Desafortunadamente, todavía no hay una convención universal sobre ajustes de tabuladores, lo que hace que el uso del carácter tab sea incierto. Sin embargo, uno de los ajustes de tabulador más comunes es cada ocho espacios.

CHR\$(12), el salto de página, es otro carácter de formato. Este carácter se interpreta como un comando para una impresora, que le dice que salte al comienzo de la siguiente página.

Hay también otros caracteres de formateo disponibles, tal como el tabulador vertical, **CHR\$(11)**, pero no son de uso muy extenso en los ordenadores personales.

Es posible evitar muchas dificultades habiendo creado los programas datos de texto con formatos simples. Los formatos más simples permiten líneas no más grandes de 255 caracteres y usan solamente los caracteres de formateo de retorno de carro (**CHR\$(13)**), salto de línea (**CHR\$(10)**) y fin de fichero (**CHR\$(26)**). Muchos lenguajes de programación, incluyendo al BASIC y al Pascal, generan automáticamente estos caracteres de formato cuando se crean datos de texto.

Normalmente también procesan los caracteres, así que sólo tenemos que tratar con el formato de texto cuando hemos evitado entrar en el control de datos usual de nuestro lenguaje.

Formatos de texto de los procesadores de texto

Los programas procesadores de texto tienen necesidades especiales para el formato de datos de texto. Los ficheros que crean estos programas son extrañamente simples y típicamente tienen muchas adiciones exóticas al formato ASCII más sencillo. Generalmente, cada procesador de texto tiene sus propias reglas de formateo; por suerte, hay algunas características comunes.

Muchos de los códigos de formato especiales usados por los procesadores de texto se crean usando un código ASCII ampliado, que es 128 unidades más grande que un código ASCII normal. Esto es equivalente a ajustar el bit de mayor peso de un byte ordinario de otra manera. Por ejemplo, un retorno de carro `CHR$(141)` «suave» se codifica añadiendo 128 a un retorno de carro ordinario `CHR$(13)`. Los retornos de carro suaves se usan para indicar un final de línea provisional, que puede ser cambiado cuando un párrafo es reformateado. Por otro lado, un retorno de carro ordinario, `CHR$(13)`, puede ser usado para señalar el final de un párrafo que no esté cambiado por reformateos. Este tipo de codificación de textos puede causar que algunos programas traten un párrafo entero como una sola línea.

Los guiones «suaves», `CHR$(173)`, que son 128 unidades mayores que los guiones ordinarios, `CHR$(45)`, son usados algunas veces para indicar dónde puede ser dividida en sílabas una palabra al final de una línea. Los guiones «duros» ordinarios, `CHR$(45)`, son tratados como caracteres regulares y no pueden ser usados o retirados por el programa de procesamiento de textos, del mismo modo que se puede con los guiones suaves.

Incluso un texto alfabético ordinario puede ser sumado 128 a su código de caracteres. Esto puede ser hecho por algunos programas para marcar la última letra de una palabra. Por ejemplo, una a minúscula es `CHR$(97)`; pero cuando aparece al final de una palabra, como América, puede ser almacenada como `CHR$(225)`, ya que 225 es la suma de $97 + 128$.

Los programas que se pretende que trabajen con una variedad de textos y datos de tratamiento de textos deberán estar preparados, todo lo que sea posible, para arreglárselas con la variedad de codificaciones que estos ejemplos sugieren.

INDICE ALFABETICO

A

Altavoz, 164.
 activación, 168.
 control directo, 169.
 volumen, 171.
ANSI, 272.
ASCIIZ, 336.
Atenuación, 173.
Atributo de fichero, 130.
 escritura, 200.
 lectura, 199.
Atributos, 92.

B

BASIC, carga, 246.
Boot, 123.
Boot-strap, 52, 120, 246.
Buffer, 144, 308.
Bus de datos, 13-14.
Bus de direcciones, 13-14.
Bytes de estado, 153.

C

CALL, 381.
Cartuchos, 72.

CDUP, 354.
CGA, 82.
CGA, 193.
Cinta, escritura de bloques, 240.
 lectura de bloques, 239.
 motor, 239.
Clusters, 122, 136.
Código de error, 361.
 retorno, 358.
Códigos de error, 335.
 exploración, 145-147.
Cola de impresión, 284.
Color, 86, 89, 93.
Compatibilidad, 108.
Compilador, 384.
Controlador de CRT, 78.
 de CRT 6845, 12.
 disco PD765, 12.
 DMA 8237A, 3, 11.
 interrupciones 8259, 10.
 interrupciones 8259A, 3.
 teclado 8042, 145.
 video 6845, 108.
 CRT 6845, 106.
 DMA, 24.
Controladores de dispositivos, 337.
Coprocesador matemático 80287, 8.
 8087, 3, 8.
CREATE, 347.
Ctrl-Break, estado, 341.
Cursor, 104.

CH

CHDIR, 346.
Chip, 8255, 2, 4.
 controlador 6845, 97.
 TI SN76469N, 4.
CHMOD, 351.

D

Dirección de manipulador de
 errores críticos, 289.
 ruptura, 288.
 terminación, 287.
PSP, 365.
Directorio, 126.
 activo, 354.
 borrado, 346.
 cambiar, 346.
 creación, 346.
Disco duro, 117, 122, 124, 217.
 búsqueda de cilindro, 218.
 cambio de estado, 219.
 diagnóstico del controlador, 219.
 inicialización de parámetros, 218.
 obtención de parámetros, 217.
 recalibrar, 218.
 reinicialización, 218.
 sectores largos, 218.
 tipo, 219.
 tipo, 220.
Disco, área de transferencia, 316.
DOS, 114.
 espacio libre, 342
 estructura física, 112.
 estructura lógica, 117.
 formato, 274.
 lectura y escritura absoluta, 280.
 reinicialización, 310.
 unidad activa, 310.
 unidad activa, 316.
 verificación de escritura, 324.
Diskette, 113, 210.
 escritura de sectores, 214.
 formateo de pistas, 215.

Diskette (continuación)
 lectura de sectores, 212.
 obtención del estado, 211.
 reinicialización, 211.
 verificación de sectores, 214.
DOS, 52, 269.
 3.0, funciones añadidas, 361.
 funciones nuevas, 333.
 funciones universales, 301.
 interrupciones, 278.
 interrupciones, 286.
 resumen de servicios, 367.
 versiones, 273.
 versiones, 340.
DTA, dirección, 340.
DUP, 353.

E

ECD, 87.
EGA, 17, 79, 82.
Equipo, listado, 244.
Estado de bloqueo por teclado,
 155.
 inserción, 153.
 toggle, 156.
 verificación, 359.
Caps-Lock, 154.
E/S de dispositivos, 351.
E/S directa por teclado/pantalla,
 306.

F

FAR CALL, 381.
FAR RET, 381.
FAT, 135, 137, 317.
Fecha, 323.
Fichero aleatorio, escribir registro,
 318, 321.
 inicializar registro, 319.
 leer registro, 318, 320.
Fichero secuencial, escribir registro,
 315.
 leer registro, 314.

Fichero transitorio, creación, 364.
 abrir, 311, 347.
 atributos, 351.
 bloqueo de acceso, 365.
 borrado, 314, 350.
 búsqueda, 312, 314, 358, 359.
 cerrar, 312.
 cerrar el número, 349.
 creación, 315, 347, 364.
 duplicación forzada, 354.
 duplicar, 353.
 escribir, 349.
 fecha y hora, 360.
 leer, 349.
 nombre, 321.
 puntero, 350.
 renombrar, 316.
 renombrar, 360.
 tamaño, 319.
 FIND FIRST, 358.

G

Generador de reloj 8284A, 3, 11.
 Generador de sonido TI
 SN76496A, 171.

H

Hora, 323.
 Horarios, 247.

I

ICA, 66.
 ID, 69.
 Imagen mapeada, 80.
 Impresora, envío de un byte, 242.
 estado, 242.
 inicialización, 242.
 salida, 306.
 IN, 41.
 Información dependiente del país,
 342.
 INT, 381.

Interfaces de programas, 380.
 Interfaz de periféricos programable
 8255, 12.
 Intérprete, 384.
 Interrupciones, 2, 25, 42, 51, 54,
 146, 277.
 IRET, 44.

M

Mapeado, 99.
 Memoria, 15, 26, 31, 38, 58, 79.
 virtual, 7.
 asignación, 355.
 liberar, 355.
 modificar asignación, 355.
 tamaño, 245.
 Microprocesador 80286, 2, 6.
 Microprocesador 8088, 2.
 MKDIR, 346.
 Modo de video, 193, 205.
 Modo gráfico, 78, 89.
 Modo texto, 78, 89.
 Monitor, 83.
 Montador, 384.
 Multitarea, 7.

N

NEAR CALL, 381.
 NEAR RET, 381.

O

OUT, 41.

P

Página de visualización activa, 197.
 Paleta de color, 202.
 Pantalla, impresión, 243.
 salida, 305.
 salida de una cadena, 308.

PIC, 10.
 Pila, 35, 183, 381.
Pixel, 80, 203, 204.
 Posición del cursor, 196.
 lápiz óptico, 196.
 Posicionamiento del cursor, 195.
 PPI, 2, 169.
 Programa, carga y ejecución, 356.
 residente, 283, 340.
 terminación, 279.
 Protección, 141, 216.
 PSP, 292, 294.
 Puerto, 105, 146, 234.
 control del CRT, 105.
 estado del CRT, 106.
 E/S, 106.
 Puertos, 40.
 lápiz óptico, 107.

R

Registros, 28, 32, 39.
 Reloj, 248-249.
 programable 8253, 3, 12.
 Resolución, 84.
 RET, 44.
 RET, 381.
 RMDIR, 346.
 ROM, 48.
 ROM, 67.
 ROM-BASIC, 70.
 ROM-BIOS, 49, 50, 71, 85, 97,
 180.
 ROM-BIOS, interrupciones, 181.
 resumen de servicios, 251.
 servicios de comunicación serie,
 234.
 impresora, 241.
 la cinta de cassette, 238.
 video, 192.
 diversos, 243.
 para disco, 209.
 para el teclado, 227.
 Ruido, 174.
 Rutina de interfaz, 184.

S

Scan code, 145.
 Scroll ascendente, 198.
 Scroll descendente, 199.
 Segmento de programa, 320.
 Señal de sincronismo vertical, 81.
 Serie, entrada, 306.
 enviar un carácter, 236.
 estado del puerto, 237.
 inicializa puerto, 235.
 recibir un carácter, 237.
 salida, 306.
 SETBLOCK, 355.
Spool, 284.
 Subdirectotios
 Subdirectorios, 128.

T

Tamaño del cursor, 194.
 Teclado, borrar el buffer, 310.
 buffer, 229.
 diferencias, 157, 161.
 entrada con buffer, 308.
 entrada con eco, 304.
 entrada directa sin eco, 307.
 entrada sin eco, 307.
 estado de la entrada, 310.
 estado de shift, 229.
 siguiente carácter, 228.
 Teletipo (TTY), 204.
 Temporizador, 166.
 Temporizador programable 8253-5,
 166.
 Terminación, 358.
 Terminar, 304.
 Toggle, 147.
 Tono, 172.
 Traductor, 384.
 Typematic, 149.

V

Vector de interrupción, 319, 341.
 Ventanas, 272.
 VGA, 17.
 VGA, 80.

PETER NORTON

Peter Norton se crió en Seattle, Washington, y se educó en el Reed College en Portland, Oregón. Antes de descubrir los microordenadores pasó doce años trabajando con mainframes y miniordenadores para compañías, entre las que se incluyen la Boeing y los Jet Propulsion Laboratories. Después del debut del PC de IBM, Peter fue casi el primero en comprar uno. Ahora, reconocido como primera autoridad en la tecnología de los ordenadores personales de IBM, Peter es el autor del libro *El IBM PC a fondo*, publicado también por Anaya Multimedia, y el creador de uno de los programas más vendidos, las *Utilidades Norton*. También es un columnista popular en las revistas *PC* y *PC Week*.

CD ROM: El nuevo papiro

Recopilado por Steve Lambert
y Suzanne Ropiequet

La tecnología CD ROM va a cambiar el mundo de la Informática.

«CD ROM: El nuevo papiro» es una recopilación ordenada de artículos, estudios y análisis sobre esta terminología, escritos por las mayores autoridades en la materia a nivel mundial.

Cubre paso a paso todos los aspectos de la tecnología, desde el *hardware* y *software* básico del sistema hasta las aplicaciones futuras de este revolucionario medio.

Se trata de un libro exhaustivamente documentado, riguroso y polifacético en el tratamiento, que constituye la primera obra importante sobre este tema que se publica en el mundo.

Representa una fuente insustituible de información sobre esta tecnología de interés general por su futura repercusión en todas las áreas de la Informática, tanto profesional como de consumo.

El libro del Apple Macintosh

Cary Lu

Es una guía práctica y completa sobre el uso del Apple Macintosh y sus aplicaciones, que describe el funcionamiento básico del ordenador y de los programas y aplicaciones (procesos de textos, hoja electrónica, base de datos, gráficos, etcétera).

Cary Lu, famoso experto en microordenadores, ha logrado recopilar en este libro toda la información que necesita un usuario de Macintosh para sacarle el máximo partido al ordenador, proporcionándole una visión completa y comprensible de la tecnología del Mac, de su filosofía, del *hardware* disponible en la actualidad y del *software* desarrollado más recientemente.

Cualquiera que tenga un Macintosh descubrirá en este libro cómo aumentar la productividad o posibilidades de uso del Mac, cómo ampliarlo o cómo hacer una selección apropiada del *software* a utilizar.

El libro del MS DOS

Van Wolverton

Se trata de un manual completo y directo, que da una visión profunda del MS DOS, sistema operativo del IBM PC y más de 50 ordenadores compatibles. Ha sido escrito con la autoridad que otorga el hecho de haber sido redactado en colaboración directa con el equipo de programadores de MICROSOFT que desarrolló el MS DOS.

Estructurado como manual de aprendizaje y referencia de MS DOS, está dividido en tres partes: Introducción y ejemplos de uso de un sistema operativo, tratamiento completo sobre manejo del MS DOS (desde las operaciones rutinarias hasta la creación de comandos propios) y sección de referencia: descripción detallada de cada comando, características avanzadas, etcétera.

Es una obra fundamental para cualquiera que use MS DOS.

Contempla todas las versiones del MS DOS, desde la 1.00 hasta la 3.0.

