# LENGUAJE ENSAMBLADOR Y PROGRAMACIÓN Y PARA IBM® PC Y COMPATIBLES

**Tercera edición**


**Peter Abel**
British Columbia
Institute of Technology


**TRADUCCIÓN:**

**Lic. Víctor Hugo Ibarra Mercado**
Lic. en Física y Matemáticas
Coordinador Matemáticas Aplicadas
Escuela de Actuaría - Universidad Anáhuac

**REVISIÓN TÉCNICA:**

**Prof. Raymundo Hugo Rangel Gutiérrez**
UNAM


PEARSON Educación®

**México • Argentina • Brasil • Colombia • Costa Rica • Chile • Ecuador**
**España • Guatemala • Panamá • Perú • Puerto Rico • Uruguay «Venezuela**

*EDICIÓN EN INGLÉS*

| | |
|---|---|
| **PRE-PRESS/MANUFACTURING BUYER:** | **BILL SCAZZERO** |
| **ACQUISITIONS EDITOR:** | **MARCIA HORTON** |
| **EDITORIAL7PRODUCTION SUPERVISIÓN AND INTERIOR DESIGN:** | **RICHARD DeLORENZO** |
| **COPY EDITOR;** | **BRIAN BAKER** |
| **EDITORIAL ASSISTANT:** | **DOLORES MARS** |
| **SUPLEMENT EDITOR:** | **ALICE DWORKIN** |

*ABEL: LENGUAJE ENSAMBLADOR Y PROGRAMACIÓN PARA IBM PC Y COMPATIBLES (3a. ed.)*

Traducido del inglés de la obra: IBM®-PC ASSEMBLY LANGUAGE AND PROGRAMMING.

All Rights Reserved. Authorized translation from English language edition published by Prentice Hall Inc, A Simón & Shuster Company.

Todos los derechos reservados. Traducción autorizada de la edición en inglés publicada por Prentice Hall Inc.

All Rights Reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage and retrieval system, without permission in writing from the publisher.

Prohibida la reproducción total o parcial de esta obra, por cualquier medio o método, sin la autorización escrita del editor.

Derechos reservados © 1996 respecto a la primera edición en español publicada por PRENTICE-HALL HISPANOAMERICANA, S.A.

**Atlacomulco Núm. 500-5° Piso**
**Col. Industrial Atoto**

**53519, Naucalpan de Juárez, Edo. de México**

ISBN 968-880-708-7
Miembro de la Cámara Nacional de la Industria Editorial, Reg. Núm. 1524
Original English Language Edition Published by Prentice Hall Inc.
Copyrigth © MCMXCV
ISBN 0-13-124603-8
IMPRESO EN MÉXICO/PRINTED IN MÉXICO

# Contenido

***PREFACIO***

**Parte** A **Fundamentos de hardware** y **software de la PC**

/ ***INTRODUCCIÓN AL HARDWARE DE LA PC***

Introducción 1
Bits y bytes 2
Números binarios 3
Representación hexadecimal 6
Código ASCII 7
El procesador 7
Memoria interna 9
Segmentos y direccionamiento 10
Registros 13
Puntos clave 17
Preguntas 18

2 ***REQUERIMIENTOS DE SOFTWARE DE LA PC***

Introducción 19
Características del sistema operativo 19
El proceso de arranque 20
Interfaz DOS-BIOS 21
Programa cargador del sistema 21

La pila (stack) 22
Direccionamiento de programas 24
Referencias a memoria y a registros 26
Puntos clave 26
Preguntas 27

*3 EJECUCIÓN DE INSTRUCCIONES*

Introducción 28
El programa DEBUG 29
Visualización de las localidades de memoria 30
Ejemplo I de lenguaje de máquina: Datos inmediatos 32
Ejemplo II de lenguaje de máquina: Datos definidos 37
Cómo introducir un programa simbólico en ensamblador 40
Uso de la instrucción INT 41
Cómo guardar un programa desde DEBUG 43
Ejemplo de lenguaje ensamblador: El operador PTR 44
Puntos clave 45
Preguntas 45

**Parte B Fundamentos de lenguaje ensamblador**

*4 REQUERIMIENTOS DE LENGUAJE ENSAMBLADOR*

Introducción 48
Ensambladores y compiladores 49
Comentarios en lenguaje ensamblador 49
Palabras reservadas 50
Identificadores 50
Instrucciones 51
Directivas 52
Cómo inicializar un programa para su ejecución 55
Cómo terminar la ejecución de un programa 57
Ejemplo de un programa fuente 58
Cómo inicializar el modo protegido 59
Directivas simplificadas de segmentos 59
Definición de datos 61
Directivas para la definición de datos 63
La directiva EQU 68
Puntos clave 69
Preguntas 70

*5 CÓMO ENSAMBLAR, ENLAZAR Y EJECUTAR UN PROGRAMA*

Introducción 72
Cómo preparar un programa para su ejecución 73
Cómo ensamblar un programa fuente 73

Listado del ensamblador de las definiciones convencionales de segmentos 75
Listado del ensamblador de las directivas simplificadas de segmentos 79
Ensamblador de dos pasadas 79
Cómo enlazar un programa objeto 81
Cómo ejecutar un programa 83
Listado de referencias cruzadas 84
Diagnóstico de errores 85
Puntos clave 86
Preguntas 86

*6 INSTRUCCIONES Y DIRECCIONAMIENTO DEL PROCESADOR 88*

. Introducción 88
El conjunto de instrucciones del procesador 88
Operandos 92
La instrucción MOV 95
Instrucciones para mover y llenar 96
Operandos inmediatos 97
La instrucción XCHG 98
La instrucción LEA 99
Las instrucciones INC y DEC 99
Instrucciones de movimiento extendido 99
La instrucción INT 101
Alineación de direcciones 101
Direcciones cercana y lejana 102
Prefijo que invalida el segmento 102
Puntos clave 103
Preguntas 104

*7 ESCRITURADEPROGRAMAS.COM 106*

Introducción 106
Diferencias entre programas .COM y .EXE 106
Conversión a formato .COM 107
Ejemplo de un programa .COM 108
La pila de .COM 109
Sugerencias para la depuración 110
Puntos clave 110
Preguntas 111

*8 LÓGICA Y CONTROL DE PROGRAMAS 112*

Introducción 112
Direcciones corta, cercana y lejana 113
Etiquetas de instrucciones 113
La instrucción JMP 114
La instrucción LOOP 116
El registro de banderas 117

La instrucción CMP 118
Instrucciones de salto condicional 118
Llamada a procedimientos 121
Efectos en la pila de la ejecución de programas 123
Operaciones booleanas 125
Cambio de minúsculas a mayúsculas 126
Corrimiento de bits 127
Rotación de bits (desplazamiento circular) 129
Tablas de bifurcación 131
Organización de un programa 132
Puntos clave 134
Preguntas 135

**Parte** C **Operaciones para la pantalla y el teclado** **136**

*9 INTRODUCCIÓN AL PROCESAMIENTO EN PANTALLA Y DEL TECLADO* *136*

Introducción 136
La pantalla 137
Colocación del cursor 138
Limpiar la pantalla 138
Función 09H del DOS para despliegue en pantalla 139
Función OAH del DOS para entrada del teclado 141
Cómo aceptar y desplegar nombres 142
Uso de caracteres de control para desplegar 146
Función 02H del DOS para despliegue en pantalla 147
Manejadores de archivos 148
Manejadores de archivo para despliegue en pantalla 148
Manejadores de archivo para entrada desde el teclado 149
Puntos clave 151
Preguntas 152

*10 PROCESAMIENTO AVANZADO DE LA PANTALLA* *153*

Introducción 153
Adaptadores de video 154
Especificaciones del modo de video 155
Modo de texto 155
Páginas de pantalla 158
Interrupción 10H del BIOS para modo de texto 159
Uso del BIOS para desplegar el conjunto de caracteres ASCII 165
Caracteres ASCII extendidos 166
Intermitencia, video inverso y recorrido de la pantalla 169
Despliegue directo en video 171
Modo gráfico 173
Interrupción 10H del BIOS para gráficos 175
Cómo especificar y desplegar el modo gráfico 178
Determinación del tipo de adaptador de video 178

Puntos clave 180
Preguntas 180

*// PROCESAMIENTO AVANZADO DEL TECLADO* *182*

Introducción 182
El teclado 183
Estado del shift del teclado 184
Búfer del teclado 185
Interrupción 21H del DOS para entrada desde el teclado 185
Interrupción 16H del BIOS para entrada desde el teclado 187
Teclas de función extendidas y códigos de rastreo 189
Selección de un menú 191
Interrupción 09H y el búfer del teclado 195
Cómo ingresar el conjunto completo de caracteres ASCII 197
Puntos clave 198
Preguntas 198

**Parte D Manipulación de datos** **200**

*12 OPERACIONES CON CADENAS DE CARACTERES* *200*

Introducción 200
Características de las operaciones con cadenas de caracteres 201
REP: Prefijo de repetición de cadena 201
MOVS: Mover una cadena de caracteres 202
LODS: Carga una cadena de caracteres 204
STOS: Almacenar una cadena de caracteres 205
Cómo transferir datos con LODS y STOS 206
CMPS: Comparar cadenas 206
SCAS: Búsqueda en cadenas 209
Buscar y reemplazar 210
Codificación alterna para instrucciones de cadena de caracteres 211
Cómo duplicar un patrón 211
Cómo alinear a la derecha en la pantalla 212
Puntos clave 215
Preguntas 215

*13 ARITMÉTICA: I—PROCESAMIENTO DE DATOS BINARIOS* *217*

Introducción 217
Suma y resta 218
Aritmética con palabras múltiples 220
Datos con signo y sin signo 223
Multiplicación 224
Multiplicación de palabras múltiples 226
Instrucciones especiales de multiplicación 230
Multiplicación por corrimiento 231

División 232
División por medio de corrimientos 236
Cambio (inversión) del signo 237
Procesadores numéricos de datos (coprocesadores) 237
Puntos clave 239
Preguntas 239

*14 ARITMÉTICA: II—PROCESAMIENTO DE DATOS ASCII Y BCD* *241*

Introducción 241
Datos en formato decimal 242
Procesamiento de datos ASCII 243
Procesamiento de datos BCD desempaquetados 245
Procesamiento de datos BCD empaquetados 248
Conversión de formato ASCII a binario 250
Conversión de formato binario a ASCII 250
Corrimiento y redondeo 251
Programa para convertir datos ASCII 253
Puntos clave 258
Preguntas 259

*15 PROCESAMIENTO DE TABLAS* *260*

Introducción 260
Definición de tablas 260
Direccionamiento directo en tablas 262
Búsqueda en una tabla 266
La instrucción XLAT (Traducir) 271
Despliegue de caracteres hexadecimales y ASCII 272
Ordenamiento de entradas de una tabla 274
Listas ligadas (enlazadas) 275
Tipo, longitud y tamaño de los operadores 279
Puntos,clave 279
Preguntas 280

**Parte E Entrada/salida avanzada** **282**

*16 ORGANIZACIÓN DEL ALMACENAMIENTO EN DISCO* *282*

Introducción 282
Características de los discos 282
Área de sistemas y área de datos en disco 285
Registro de arranque 286
Directorio 287
Tabla de asignación de archivos 288
Ejercicio que implica el uso de la FAT 292
Procesamiento de archivos en disco 294
Puntos clave 294
Preguntas 295

*17 PROCESAMIENTO EN DISCO: I-ESCRITURA Y LECTURA DE ARCHIVOS*

Introducción 296
Cadenas ASCIIZ 297
Manejadores de archivos 297
Códigos de error de regreso 298
Apuntadores de archivo 298
Uso de manejadores de archivo para crear archivos en disco 298
Uso de manejadores de archivo para leer archivos en disco 303
Procesamiento de archivos ASCII 307
Uso de manejadores de archivo para procesamiento directo 310
Servicios de disco que usan bloques de control de archivo 312
Uso de FCB para crear archivos en disco 316
Uso de FCB para lectura secuencial de archivos en disco 318
Uso de FCB para procesamiento directo 319
Procesamiento directo de bloques 320
E/S absoluta de disco 321
Puntos clave 322
Preguntas 323

*18 PROCESAMIENTO EN DISCO: II-OPERACIONES DEL DOS PARA SOPORTE DE DISCOS Y ARCHIVOS* *325*

Introducción 325
Operaciones para manejo de unidades de disco 326
Programa: Lectura de información desde los sectores 336
Operaciones para manejar el directorio y la FAT 338
Programa: Despliegue del directorio 340
Operaciones para manejar archivos en disco 340
Programa: Borrar archivos de manera selectiva 347
Puntos clave 350
Preguntas 350

*19 PROCESAMIENTO EN DISCO: III-OPERACIONES DEL BIOS PARA DISCO* *352*

Introducción 352
Byte de estado del BIOS 353
Operaciones básicas del BIOS para disco 354
Uso del BIOS para leer sectores 356
Otras operaciones del BIOS para disco 356
Palabras clave 362
Preguntas 362

*20 IMPRESIÓN* *364*

Introducción 364
Caracteres comunes de control para impresora 365
DOS 21H, función 40H: Imprimir caracteres 365

Impresión con encabezados de página 366
Impresión de archivos ASCII y manejo de tabuladores 369
DOS 21H, función 05H: Imprimir un carácter 373
Caracteres especiales de control para la impresora 373
Funciones de la INT 17H del BIOS para impresión 374
Puntos clave 376
Preguntas 376

*21 OTRAS FACILIDADES DE ENTRADA/SALIDA*

Introducción 377
Características del ratón 377
Funciones del ratón 378
Operaciones comunes del ratón 379
Programa para el ratón 385
Puertos 388
Generación de sonidos 390
Puntos clave 391
Preguntas 392

Programación avanzada

*22 ESCRITURA DE MACROS*

Introducción 393
Una definición sencilla de una macro 394
Uso de parámetros en macros 394
Comentarios 396
Uso de una macro dentro de una definición de una macro 398
La directiva LOCAL 399
Incluir (include) desde una biblioteca de macros 401
Concatenación 402
Directivas de repetición 403
Directivas condicionales 404
Puntos clave 408
Preguntas 410

*23 ENLACE A SUBPROGRAMAS*

Introducción 411
Segmentos 412
Llamadas intrasegmento 413
Llamadas intersegmento 414
Atributos EXTRN y PUBLIC 415
Atributos EXTERN y PUBLIC para una etiqueta 417
Uso de PUBLIC en el segmento de código 419
Directivas simplificadas de segmento 421
Datos comunes en subprogramas 423

Definición de datos en ambos programas 423
Paso de parámetros 425
Enlace de programas en Pascal y en lenguaje ensamblador 429
Enlazando programas C y lenguaje ensamblador 431
Puntos clave 434
Preguntas 435

*24 ADMINISTRACIÓN DE LA MEMORIA DEL DOS* *437*

Introducción 437
Programas principales del DOS 438
Área de memoria alta 439
COMMAND.COM 439
Prefijo de segmento de programa (PSP) 440
Bloques de memoria 444
Estrategia de asignación de memoria 447
Cargador de programa 448
Asignación y liberación de memoria 453
Carga y ejecución de una función de programa 454
Traslape de programas 458
Programas residentes 462
Puntos clave 467
Preguntas 468

**Parte G Capítulos de referencia** **469**

*25 ÁREAS DE DATOS E INTERRUPCIONES DEL BIOS* *469*

Introducción 469
El proceso de arranque 470
El área de datos del BIOS 470
Servicios de interrupción 474
Interrupciones del BIOS 475
Puntos clave 478
Preguntas 479

*26 INTERRUPCIONES DEL DOS* *480*

Introducción 480
Interrupciones del DOS 481
Servicio de la INT 21H del DOS 481
Puntos clave 486
Preguntas 486

*27 OPERADORES Y DIRECTIVAS* *487*

Introducción 487
Especificadores de tipo 487

Operadores 488
Directivas 494

***28 EL CONJUNTO DE INSTRUCCIONES DE LA PC***

Introducción 514
Notación de registros 515
Byte del modo de direccionamiento 515
Instrucciones de dos bytes 517
Instrucciones de tres bytes 517
Instrucciones de cuatro bytes 517
Conjunto de instrucciones 518

***APÉNDICES***

A Conversión entre hexadecimal y decimal 542
B Códigos de caracteres ASCII 545
C Palabras reservadas 547
D Opciones de ensamblado y de enlace 549
E El programa DEBUG del DOS 557
F Códigos de rastreo del teclado y códigos ASCII

***RESPUESTAS A PREGUNTAS SELECCIONADAS***

***ÍNDICE***

# Prefacio

El corazón de la computadora es el microprocesador, éste maneja las necesidades aritméticas, de lógica y de control de la computadora. El microprocesador tiene su origen en la década de los sesenta, cuando se diseñó el circuito integrado (IC por sus siglas en inglés) al combinar varios componentes electrónicos en un solo componente sobre un "chip" de silicio. Los fabricantes colocaron este diminuto chip en un dispositivo parecido a un ciempiés y lo conectaron a un sistema en funcionamiento. A principios de los años setenta Intel introdujo el chip 8008 el cual, instalado en una computadora terminal, acompañó a la primera generación de microprocesadores.

En 1974 el 8008 evolucionó en el 8080, un popular microprocesador de la segunda generación para propósitos generales. En 1978 Intel produjo la tercera generación de procesadores 8086, para proporcionar alguna compatibilidad con el 8080 y que representan un avance significativo de diseño. Después, Intel desarrolló una variación del 8086 para ofrecer un diseño ligeramente más sencillo y compatibilidad con los dispositivos de entrada/salida de ese momento. Este nuevo procesador, el 8088, fue seleccionado por IBM para su computadora personal en 1981. Una versión mejorada del 8088 es el 80188, y versiones mejoradas del 8086 son los 80186, 80286, 80386, 80486 y el Pentium (también conocido como P5), cada uno de ellos permite operaciones adicionales y más procesamiento.

La variedad de microcomputadoras también ocasionó un renovado interés en el lenguaje ensamblador, cuyo uso conlleva diferentes ventajas:

- Un programa escrito en lenguaje ensamblador requiere considerablemente menos memoria y tiempo de ejecución que un programa escrito en los conocidos como lenguajes de alto nivel, como Pascal y C.
- El lenguaje ensamblador da a un programador la capacidad de realizar tareas muy técnicas que serían difíciles, si no es que imposibles de realizar en un lenguaje de alto nivel.

- El conocimiento del lenguaje ensamblador permite una comprensión de la arquitectura de la máquina que ningún lenguaje de alto nivel puede ofrecer.
- Aunque la mayoría de los especialistas en software desarrolla aplicaciones en lenguajes de alto nivel, que son más fáciles de escribir y de dar mantenimiento, una práctica común es recodificar en lenguaje ensamblador aquellas rutinas que han causado cuellos de botella en el procesamiento.
- Los programas residentes y rutinas de servicio de interrupción casi siempre son desarrollados en lenguaje ensamblador.

Los lenguajes de alto nivel fueron diseñados para eliminar las particularidades de una computadora específica, mientras que un lenguaje ensamblador está diseñado para una computadora específica, o, de manera más correcta, para una familia específica de microprocesadores. A continuación se listan los requisitos para aprender el lenguaje ensamblador de la PC:

- Tener acceso a una computadora personal de **IBM** (cualquier modelo) o una compatible.
- Una copia del sistema operativo **ms-dos** o **pc-dos** (de preferencia, una versión reciente) y estar familiarizado con su uso.
- Una copia de un programa ensamblador (otra vez, de preferencia, una versión reciente). Las versiones de Microsoft son conocidas como **MASM** y QuickAssembler: **TASM** es de Borland y **OPTASM** es de System.

Para el aprendizaje de lenguaje ensamblador no es necesario lo siguiente:

- Conocimiento previo de un lenguaje de programación, aunque el tenerlo puede ayudarle a comprender algunos conceptos de programación más rápido.
- Conocimiento previo de electrónica o circuitería. Este libro proporciona toda la información acerca de la arquitectura de la PC que usted necesita para programar en lenguaje ensamblador.

## SISTEMAS OPERATIVOS

Los propósitos principales de un sistema operativo son (1) permitir a los usuarios instruir a una computadora con respecto a las acciones que debe tomar (como ejecutar un programa en particular) y (2) facilitar los medios de almacenamiento de la información en disco ("catalogar") y de tener acceso a la misma.

El sistema operativo más común para la PC y sus compatibles es el **MS-DOS** de Microsoft, conocido como **PC-DOS** en la **IBM PC.** Cada una de las versiones del DOS ha proporcionado características adicionales que han extendido las capacidades de la PC. Un estudio de sistemas operativos avanzados, como os/2 y **UNix,** se encuentra fuera del los alcances de este libro.

## OBJETIVO DEL LIBRO

El propósito principal de este libro es ayudar a los lectores en el aprendizaje de la programación en lenguaje ensamblador. Para este fin, el libro cubre los aspectos más sencillos del hardware y del lenguaje y después conforme se requiere introduce las instrucciones necesarias. El texto también subraya la claridad de los programas de ejemplo. Así, los ejemplos utilizan aquellas instrucciones y enfoques que son más fáciles de entender, aunque un programador profesional resolvería problemas similares con un código más sofisticado, pero menos claro.

Los programas también omiten instrucciones de macros (éstas se explican en el capítulo 22). A pesar de que los programadores profesionales utilizan macros constantemente, su aparición en un libro de esta naturaleza interferiría con el aprendizaje de los principios del lenguaje. Una vez, que estos principios se han aprendido, un programador puede adoptar las técnicas inteligentes del profesional.

## CÓMO EMPLEAR EL LIBRO

Esta obra puede emplearse tanto como libro de texto que como de referencia permanente. Para hacer más eficaz su inversión en una microcomputadora y software, trabaje con cuidado en cada uno de los capítulos y relea cualquier material que no sea claro de inmediato. Teclee los programas de ejemplo en su computadora, conviértalos en "módulos" ejecutables y prepárelos para ejecutarlos (o "correrlos"). También, resuelva los del final de cada capítulo.

Los primeros nueve capítulos tratan el material fundamental para el libro y para el lenguaje ensamblador. Después de estudiarlos puede continuar con los capítulos 12, 13, 15, 16, 20, 21 o 22. Los capítulos 25, 26, 27 y 28 tienen la intención de ser referencias. Los capítulos interrelacionados son:

- 9 a 11 (sobre operaciones con la pantalla y el teclado).
- 13 y 14 (sobre operaciones aritméticas).
- 16 a 19 (sobre procesamiento en disco).
- 23 y 24 (sobre subprogramas y administración de la memoria).

Al terminar este libro, usted será capaz de:

- Entender el hardware de la computadora personal.
- Entender código en lenguaje de máquina y en formato hexadecimal.
- Entender los pasos al ensamblar, enlazar y ejecutar un programa.
- Escribir programas en lenguaje ensamblador para manejar el teclado y la pantalla, realizar aritmética, hacer conversiones entre los formatos ASCII y binario, formar tablas de búsqueda y ordenamiento y manejar entradas y salidas de disco.
- Rastrear la ejecución de la máquina como ayuda en la depuración de programas.
- Escribir sus macroinstrucciones para facilitar la codificación.
- Enlazar programas ensamblados aparte en un programa ejecutable.

Aprender lenguaje ensamblador y conseguir que sus programas funcionen es una experiencia excitante y desafiante. Por el tiempo y esfuerzo invertidos, las recompensas de seguro son grandes.

## NOTAS SOBRE LA TERCERA EDICIÓN

Esta tercera edición lleva una considerable cantidad de mejoras sobre la edición anterior. Algunas de ellas son:

- Inclusión y mayor énfasis en las funciones adicionales en versiones más recientes del DOS.
- Programación para operaciones con el ratón.

- Características de los procesadores 80486 y Pentium de Intel.
- Inclusión de material acerca del área de memoria superior y el área de memoria alta.
- Inclusión de material sobre las más recientes versiones de ensamblados.
- Mayor cobertura de funciones de procesamiento en disco para DOS, la tabla de asignación de archivos y procesamiento directo.
- Detalles completos de los códigos de rastreo y de las combinaciones de teclas del teclado extendido.
- Reorganización y revisión considerables de las explicaciones en todas las partes del texto.

## RECONOCIMIENTOS

El autor está agradecido por la ayuda y cooperación de todos aquellos que contribuyeron con sugerencias para la revisión y corrección de ediciones anteriores. Para esta tercera edición, vaya un agradecimiento especial a Brian R. Anderson del British Columbia Institute of Technology por la información sobre el ratón y la programación C.

PARTE A — Fundamentos del hardware y software de la PC

CAPÍTULO **1**

# Introducción al hardware de la PC

OBJETIVO

Explicar las características básicas del hardware de la microcomputadora y la organización de programas.

## INTRODUCCIÓN

Escribir un programa en lenguaje ensamblador requiere de conocimientos acerca del hardware (arquitectura) de la computadora, su conjunto de instrucciones y sus reglas de uso. En este capítulo se ofrece una explicación del hardware básico: bits, bytes, registros, el procesador y el bus de datos. El conjunto de instrucciones y su uso son desarrollados a lo largo del libro.

Los bloques fundamentales de información de una computadora son los *bits* y los *bytes.* Éstos proporcionan los medios por los cuales la computadora puede representar datos e instrucciones en la memoria.

Los elementos principales de hardware interno de la computadora son un microprocesador, la memoria y los registros; los elementos de hardware externo son los dispositivos de entrada/salida, como el teclado, el monitor y el disco. El software consta de diversos programas y archivos de datos (incluyendo al sistema operativo) almacenados en el disco. Para ejecutar (o correr) un programa, el sistema lo copia del disco a la memoria interna. (La memoria interna es lo que la gente entiende cuando pide que su computadora tenga, por ejemplo, 8 megabytes de memoria.) El microprocesador ejecuta las instrucciones del programa, y los registros manejan la aritmética, movimiento de datos y el direccionamiento.

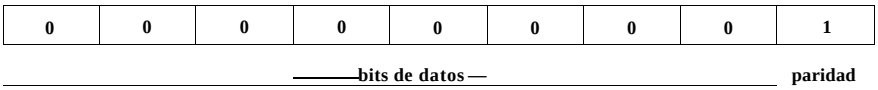
Un programa en lenguaje ensamblador consiste en uno o más *segmentos* para definir datos y almacenar instrucciones de máquina y un segmento llamado *stack* (o *pila)* que contiene direcciones almacenadas.

## BITS Y BYTES

La unidad más pequeña de información en la computadora es el *bit.* Un bit puede estar no magnetizado, o *apagado,* de modo que su valor es cero, o bien, magnetizado, o *encendido,* de modo que su valor es uno. Un solo bit no proporciona mucha información, pero es sorprendente lo que un conjunto de ellos puede hacer.

### Bytes

A un grupo de nueve bits se le llama *byte,* el cual representa localidades de almacenamiento, tanto en memoria interna como en discos externos. En memoria, cada byte tiene una dirección única, que inicia con cero para el primer byte. Cada byte tiene ocho bits para datos y un bit de paridad:




Los ocho bits de datos proporcionan la base para la aritmética binaria y para representar caracteres como la letra A o el símbolo de asterisco (*). Ocho bits permiten 256 combinaciones diferentes de condiciones de apagado-encendido (off-on), desde todos los bits apagados (00000000) hasta todos los bits encendidos (11111111). Por ejemplo, una representación de los bits para la letra A es 01000001 y para el asterisco es 00101010, aunque no tenemos que memorizarlas.

La paridad requiere que el número de bits encendidos en cada byte siempre sea *impar.* Puesto que la letra A contiene dos bits encendidos, para forzar la paridad impar el procesador establece de forma automática su bit de paridad en encendido (01000001-1). De forma similar, puesto que el asterisco tiene tres bits encendidos, para mantener la paridad impar el procesador establece el bit de paridad en apagado (00101010-0). Cuando una instrucción hace referencia a un byte en memoria interna, el procesador verifica su paridad. Si su paridad es par, el sistema supone que un bit está "perdido" y exhibe un mensaje 0 de error. Un error de paridad puede ser resultado de una falla en el hardware o un trastorno eléctrico; de cualquier forma, es un acontecimiento raro.

Puede preguntarse cómo es que la computadora "sabe" que el valor de los bits 01000001 representa la letra A. Cuando usted oprime la A en el teclado, el sistema envía una señal desde esa tecla a la memoria y establece un byte (en una posición de entrada) al valor 01000001. Usted puede mover el contenido de este byte de un lugar a otro de la memoria y aun imprimirlo o mostrarlo en la pantalla como la letra A.

Para propósitos de referencia, los bits en el byte se numeran del 0 al 7 de derecha a izquierda, como se muestra aquí para la letra A (ya no nos preocuparemos por el bit de paridad):

| **Número de bit:** | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| **Contenido en bits para la A:** | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |

### Bytes relacionados

Un programa puede tratar a un grupo de bytes como una unidad de información, como tiempo o distancia. A un grupo de uno o más bytes que definen un valor particular se le conoce comúnmente como *campo.* La computadora también emplea ciertos tamaños que le son naturales:

- *Palabra.* Un campo de 2 bytes (16 bits). Los bits en una palabra son numerados desde 0 hasta 15, de derecha a izquierda, como se muestra a continuación para las letras 'PC:

| Número de bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Contenidos en bits (PC): | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |

- *Palabra doble.* Un campo de 4 bytes (32 bits).
- *Palabra cuádruple.* Un campo de 8 bytes (64 bits).
- *Párrafo.* Un campo de 16 bytes (128 bits).
- Kilobyte (KB). El número $2^{10}$ es igual a 1024, el cual pasa a ser el valor de K, por kilobytes. Por tanto, una computadora con una memoria de 640K tiene 640 x 1024, o 655,360 bytes.
- Megabyte (MB). El número $2^{20}$ es igual a 1,048,576, o un megabyte.

## NÚMEROS BINARIOS

Puesto que la computadora sólo puede distinguir entre bits 0 y 1, trabaja con un sistema de numeración de base 2 conocido como binario. De hecho, la palabra "bit" es una contracción de las palabras inglesas "binary digit" (dígito binario).

Una colección de bits puede representar cualquier valor numérico. El valor de un número binario depende de las posiciones relativas de cero a uno de los bits. Al igual que en los números decimales, las posiciones de derecha a izquierda representan potencias ascendentes (pero de 2, no de 10). En el siguiente número de ocho bits, todos los bits se toman como uno (encendido):

| Posición: | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Valor del bit: | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Valor de la posición: | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |

El primer bit de la derecha toma el valor 1 ($2^0$);. el que sigue a la izquierda toma el valor 2 ($2^1$); el siguiente el valor 4 ($2^2$), y así sucesivamente. En este caso el valor del número binario es 1 + 2 + 4 + ... + 128 = 255 (o $2^8$ - 1).

En forma similar, el valor del número binario 01000001 se calcula como 1 más 64, o 65:

| Valor del bit: | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
|---|---|---|---|---|---|---|---|---|
| Valor de la posición: | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |

Pero, ¿no es 01000001 la letra A? En realidad, sí. Los bits 01000001 pueden representar ya sea el número 65 o bien la letra A, como a continuación se indica:

- Si un programa define los datos para propósitos aritméticos, entonces 01000001 es un número binario equivalente al número decimal 65.
- Si un programa define los datos con propósitos descriptivos, como encabezados, entonces 01000001 representa un carácter alfabético.

. Cuando inicie la programación, verá con más claridad esta distinción, puesto que define y utiliza cada elemento de información para un propósito específico. En la práctica, rara vez los dos usos son fuente de confusión.

Un número binario no está limitado a 8 bits. Un procesador que utiliza una arquitectura de 16 bits (o de 32 bits) maneja de manera automática números de 16 bits (o de 32 bits). Para 16 bits, $2^{16} - 1$, da valores hasta 65,535, y para 32 bits, $2^{32} - 1$, proporciona valores hasta 4,294,967,295.

**Aritmética binaria**

La microcomputadora realiza aritmética sólo en formato *binario.* En consecuencia, el programador de lenguaje ensamblador tiene que estar familiarizado con el formato binario y la suma binaria. Los siguientes ejemplos ilustran la suma binaria:

```
  0      0      1    1
+ 0    + 1    + 1    + 1
  0      1     10   ± 1
                     11
```

Note en los dos últimos ejemplos un 1 de acarreo. Ahora, sumemos 01000001 a 00101010. ¿Estamos sumando la letra A con el asterisco? No, son las cifras decimales 65 y 42:

| Decimal | Binario |
|---|---|
| 65 | 01000001 |
| +42 | +00101010 |
| 107 | 01101011 |

Verifique que la suma binaria 01101011 realmente es 107. Otro ejemplo: sume los valores decimales 60 y 53:

| Decimal | Binario |
|---|---|
| 60 | 00111100 |
| + 53 | +00110101 |
| 113 | 01110001 |

**Números negativos**

Los números binarios anteriores son todos positivos, porque en cada uno el último bit de la izquierda es un cero. Un número binario negativo tiene un 1 en el bit de la izquierda. Sin embargo, no es tan simple como cambiar el bit de la izquierdaa 1, tal como 01000001 (+65)a 11000001. Un valor negativo se expresa en *notación de complemento a dos;* esto es, para representar un número binario como negativo la regla es: invierta los bits y sume 1. (Se entiende por invertir un bit que si su valor es 1, lo cambiamos por 0, y si su valor es 0, lo cambiamos por 1.) Como ejemplo, encontrar el complemento a dos de 01000001 (o 65):

| | |
|---|---|
| Número +65: | 01000001 |
| Invertir los bits: | 10111110 |
| Sumar 1: | 1 |
| Número-65: | 10111111 |

Un número binario es negativo si su último bit a la izquierda es 1, pero si suma los valores de los bits que tienen 1, para convertir el número 10111111 a decimal, no obtendrá 65. Para determinar el valor absoluto de un número negativo binario, simplemente repita la operación anterior, esto es, invierta los bits y sume 1:

| | |
|---|---|
| Número-65: | 10111111 |
| Invertir los bits: | 01000000 |
| Sumar 1: | 1 |
| Número +65: | 01000001 |

La suma de +65 y -65 debe ser cero. Pruébelo:

| | |
|---|---|
| +65 | 01000001 |
| **-65** | +10111111 |
| 00 | (1)00000000 |

En la suma, el valor de los 8 bits es cero, y el acarreo de un 1 a la izquierda se pierde. Pero como existe un acarreo hacia el bit de signo y un acarreo hacia afuera del bit de signo, el resultado es correcto.

La resta binaria es simple: convierta el número que será restado a su complemento a dos y sume los números. Restar 42 de 65. La representación binaria de 42 es 00101010 y su complemento a dos es 11010110:

| | |
|---|---|
| 65 | 01000001 |
| +(-42) | +11010110 |
| 23 | (1)00010111 |

El resultado, 23, es correcto. Una vez más, existe un acarreo válido hacia el bit de signo y un acarreo hacia fuera.

Si la justificación para la notación de complemento a dos no es inmediatamente clara, considere la siguiente pregunta: ¿Qué valor tiene que ser sumado al número binario 00000001 para hacer que la suma sea igual a 00000000? En términos de números decimales, la respuesta sería -1. El complemento a dos del 1 es 11111111. Así sumamos +1 y -1 como sigue:

| | |
|---|---|
| 1 | 00000001 |
| +(-1) | 11111111 |
| Resultado: | (1)00000000 |

Ignorando el acarreo de 1, puede ver que el número binario 11111111 es equivalente al decimal -1. También puede ver un patrón en la forma en que los números binarios decrecen en valor

| | |
|---|---|
| +3 | 00000011 |
| +2 | 00000010 |
| +1 | 00000001 |
| 0 | 00000000 |
| -1 | 11111111 |
| -2 | 11111110 |
| -3 | 11111101 |

De hecho, en un número negativo los bits con cero indican su valor (absoluto): trate el valor posicional de cada uno de los bits con cero como si fueran 1, sume los valores y agregue 1.

Este material sobre aritmética binaria y números negativos lo encontrará provechoso cuando vea los capítulos 12 y 13, sobre aritmética.

## REPRESENTACIÓN HEXADECIMAL

Imagine que quiere ver los contenidos de cuatro bytes adyacentes, que representan un valor binario, en memoria (una palabra doble). Aunque un byte puede tener cualquiera de las 256 combinaciones de bits, no hay manera de mostrar o imprimir muchos de ellos como caracteres ASCII comunes. (Ejemplos de tales caracteres son las configuraciones de bits para Tab, Enter, Form Feed y Escape [tabulador, Intro, Avance de página y Escape.) En consecuencia, los diseñadores de computadoras desarrollaron un método abreviado para representar información binaria. El método divide todo byte en mitades y expresa el valor para cada medio byte. Como ejemplo, considere los siguientes cuatro bytes:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Binario:** | **0101** | **1001** | **0011** | **0101** | **1011** | **1001** | **1100** | **1110** |
| **Decimal:** | **5** | **9** | **3** | **5** | **11** | **9** | **12** | **14** |

Puesto que los números 11, 12 y 14 necesitan 2 dígitos, se extiende el sistema de numeración de manera que 10 = A, 11 = B, 12 = C, 13 = D, 14 = E y 15 = F. Aquí está el número en forma abreviada que representa el contenido de los bytes dados:

59 35 B9 CE

Por tanto, el sistema de numeración incluye los "dígitos" 0 a F, y ya que existen 16 de tales dígitos, el sistema es conocido como representación *hexadecimal* (o *hex).* La figura 1-1 muestra los números decimales de 0 a 15 junto con sus valores equivalentes en binario y en hexadecimal.

| Binario | Decimal | Hexadecimal | Binario | Decimal | Hexadecimal |
|---|---|---|---|---|---|
| 0000 | 0 | 0 | 1000 | 8 | 8 |
| 0001 | 1 | 1 | 1001 | 9 | 9 |
| 0010 | 2 | 2 | 1010 | 10 | A |
| 0011 | 3 | 3 | 1011 | 11 | B |
| 0100 | 4 | 4 | 1100 | 12 | C |
| 0101 | 5 | 5 | 1101 | 13 | D |
| 0110 | 6 | 6 | 1110 | 14 | E |
| 0111 | 7 | 7 | 1111 | 15 | F |

Figura **1-1** Representación binaria, decimal y hexadecimal

El lenguaje ensamblador hace uso considerable del formato hexadecimal. Un listado de un programa ensamblador muestra, en hexadecimal, todas las direcciones, instrucciones de código de máquina y el contenido de las constantes de datos. Para depurar sus programas, puede usar el programa DEBUG del DOS, el cual también muestra las direcciones y los contenidos de los bytes en formato hexadecimal.

Muy pronto estará trabajando en formato hexadecimal. Tenga en mente que el número hexadecimal que sigue inmediatamente a F es el 10 hexadecimal, que es el valor decimal 16. Veamos a continuación algunos ejemplos sencillos de aritmética hexadecimal:

| | | | | | |
|---|---|---|---|---|---|
| 6 | 5 | F | F | 10 | FF |
| +4 | +8 | +1 | +F | +30 | +1 |
| **A** | D | 10 | 1E | 40 | 100 |

Note también que el 40 hexadecimal es igual al 64 decimal, el 100 hexadecimal es el 256 decimal y el 1,000 hexadecimal es el 4,096 decimal.

En un programa para indicar un número hexadecimal, se escribe una "H" inmediatamente después del número; así 25H = 37 decimal. Por convención, un número hexadecimal siempre empieza con un dígito 0 a 9, así que debe codificar B8H, como 0B8H. En este libro indicamos un valor hexadecimal con la palabra "hex" o una "H" después del número (como en 4C hex o 4CH); un valor binario con la palabra "binario" o una "B" a continuación del número (como 01001100 binario o 01001100B), y un valor decimal simplemente por un número (como 76). Se exceptúan los casos en que la base es obvia por el contexto.

En el apéndice A se explica cómo convertir números hexadecimales a decimal, y viceversa.

## CÓDIGO ASCII

Para uniformar la representación de caracteres, los fabricantes de microcomputadoras han adoptado el código ASCII (American Standard Code for Information Interchange). Un código uniforme facilita la transferencia de información entre los diferentes dispositivos de la computadora. El código ASCII extendido de 8 bits que utiliza la PC proporciona 256 caracteres, incluyendo símbolos para alfabetos extranjeros. Por ejemplo, la combinación de bits 01000001 (41 hex) indica la letra A. El apéndice B tiene una lista de los 256 caracteres ASCII y el capítulo 8 enseña cómo mostrarlos en la pantalla.

## EL PROCESADOR

Un elemento importante del hardware de la PC es la unidad del sistema, que contiene una tarjeta de sistema, fuente de poder y ranuras de expansión para tarjetas opcionales. Los elementos de la tarjeta de sistema son un microprocesador Intel (o equivalente), memoria de sólo lectura (ROM) y memoria de acceso aleatorio (RAM).

El cerebro de la PC y compatibles es un *microprocesador* basado en la familia 8086 de Intel, que realiza todo el procesamiento de datos e instrucciones. Los procesadores varían en velocidad y capacidad de memoria, registros y bus de datos. Un *bus de datos* transfiere datos entre el procesador, la memoria y los dispositivos externos. En realidad, dirige el tráfico (tránsito) de datos. En seguida se anota una breve descripción de varios procesadores de Intel:

**8088/80188.** Estos procesadores tienen registros de 16 bits y un bus de datos de 8 bits, y pueden direccionar hasta un millón de bytes en memoria interna. Los registros pueden procesar dos bytes al mismo tiempo, mientras que el bus de datos sólo puede transferir un byte a la vez. El 80188 es un 8088 con mayor potencia por la adición de unas cuantas instrucciones. Ambos procesadores corren en lo que se conoce como *modo real,* esto es, un programa a la vez.

**8086/80186.** Estos procesadores son similares a los 8088/80188, pero tienen un bus de datos de 16 bits y corren más rápido. El 80186 es un 8086 más potente con unas cuantas instrucciones adicionales.

**80286.** Este procesador puede correr más rápido que los anteriores y direccionar hasta 16 millones de bytes. Puede correr en modo real o en modo protegido para multitareas.

**80386.** Este procesador tiene registros de 32 bits y un bus de datos de 32 bits, y puede direccionar hasta cuatro mil millones de bytes en memoria. Puede correr en modo real o en modo protegido para multitareas.

**80486.** Este procesador también tiene registros de 32 bits y un bus de datos de 32 bits (aunque algunos clones tienen un bus de datos de 16 bits) y está diseñado para mejorar el desempeño. Puede correr en modo real o en modo protegido para multitareas.

Pentium (o P5). Este procesador tiene registros de 32 bits, un bus de datos de 64 bits y puede ejecutar más de una instrucción por ciclo de reloj. (Intel adoptó el nombre "Pentium" porque, a diferencia de los números, los nombres pueden tener derechos reservados.)

### Unidad de ejecución y unidad de interfaz del bus

El procesador se divide en dos unidades lógicas: una unidad de ejecución (EU) y una unidad de interfaz del bus (BIU), como se ilustra en la figura 1-2. El papel de la EU es ejecutar instrucciones, mientras que la BIU envía instrucciones y datos a la EU. La EU contiene una unidad aritmético-lógica (ALU), una unidad de control (CU) y varios registros. Estos elementos ejecutan instrucciones y operaciones aritméticas y lógicas.

La función más importante de la BIU es manejar la unidad de control del bus, los registros de segmentos y la cola de instrucciones. La BIU controla los buses que transfieren los datos a la EU, a la memoria y a los dispositivos de entrada/salida externos, mientras que los registros de segmentos controlan el direccionamiento de memoria.

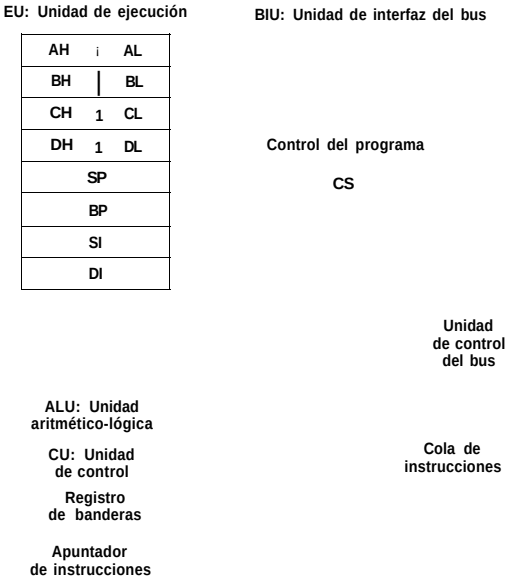



Figura 1-2 Unidad de ejecución y unidad de interfaz del bus

Otra función de la BIU es permitir el acceso a instrucciones. Ya que las instrucciones de un programa en ejecución se encuentran en la memoria, la BIU debe accesar instrucciones desde la memoria y colocarlas en la *cola de instrucciones.* Puesto que el tamaño de esta cola es de 4 a 32 bytes, dependiendo del procesador, la BIU es capaz de adelantarse y buscar con anticipación instrucciones de manera que siempre haya una cola de instrucciones listas para ser ejecutadas.

La EU y la BIU trabajan en paralelo, si bien la BIU se mantiene un paso adelante. La EU notifica a la BIU cuándo necesita acceso a los datos en memoria o a un dispositivo de E/S. También, la EU solicita instrucciones de máquina de la cola de instrucciones de la BIU. La instrucción que se encuentra adelante de la cola es la actualmente ejecutable, y mientras la EU está ocupada ejecutando una instrucción, la BIU busca otra en la memoria. Esta búsqueda se traslapa con la ejecución y aumenta la velocidad de procesamiento.

Los procesadores hasta el 80486 tienen lo que se conoce como *tubería sencilla,* la cual los restringe a completar una instrucción antes de iniciar la siguiente. El Pentium y procesadores posteriores tienen una *tubería doble* (o *dual)* que les permite correr varias operaciones en paralelo.

## MEMORIA INTERNA

La microcomputadora posee dos tipos de memoria interna: *memoria de acceso aleatorio* (RAM) y *memoria de sólo lectura* (ROM). Los bytes en memoria se numeran en forma consecutiva, iniciando con 00, de modo que cada localidad tiene un número de dirección único.

La figura 1-3 muestra un mapa físico de memoria de una PC tipo 8086. Del primer megabyte de memoria, los primeros 640K los ocupa la RAM, la mayor parte de la cual está disponible para su uso.

**ROM.** La ROM es un chip especial de memoria que (como su nombre lo indica) sólo puede ser leída. Ya que las instrucciones y los datos están "grabados" permanentemente en un chip de ROM, no pueden ser alterados. EL Sistema Básico de Entrada/Salida (BIOS) de ROM inicia en la dirección 768K y maneja los dispositivos de entrada/salida, como un controlador de disco duro. La ROM que inicia en 960K controla las funciones básicas de la computadora, como la autoprueba al encender, patrones de puntos para los gráficos y el autocargador de disco. Cuando se enciende la computadora, la ROM realiza ciertas verificaciones y carga, desde el disco, los datos especiales del sistema que envía a la RAM.

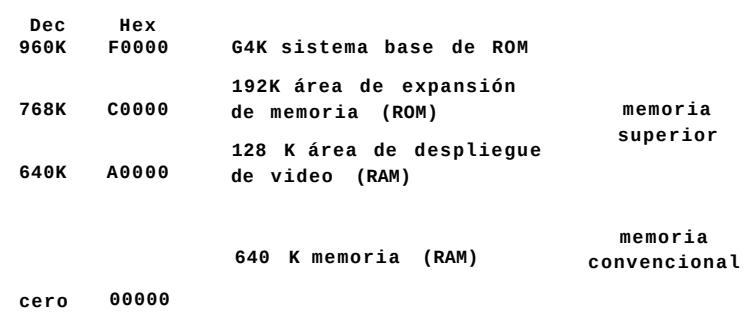


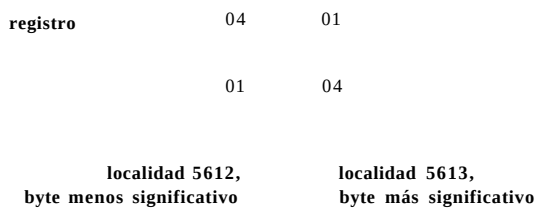

**Figura 1-3 Mapa de memoria física**

RAM. Un programador está preocupado principalmente con la RAM, que sería mejor llamada "memoria de lectura-escritura". La RAM se dispone como una "hoja de trabajo" para almacenamiento temporal y ejecución de programas.

Ya que el contenido de la RAM se pierde cuando se apaga la computadora, debe reservar almacenamiento externo para guardar programas y datos. Si cuando enciende la computadora tiene insertado un disco flexible con DOS o un disco duro instalado, el procedimiento de arranque en ROM carga el programa COMMAND.COM en RAM. Después se le pide a C0MMAND.COM realizar acciones, como cargar un programa de un disco a la RAM. Puesto que el COMMAND.COM ocupa una pequeña parte de RAM, también existe espacio para otros programas. Su programa se ejecuta en RAM y por lo común produce salida a la pantalla, a la impresora o a un disco. Cuando termina, usted puede pedir al C0MMAND.COM cargar otro programa en RAM, una acción que se escribe sobre el programa anterior. En todo el estudio posterior de la RAM se usará el término general "memoria".

### Direccionamiento de localidades de memoria

Dependiendo del modelo, el procesador puede accesar uno o más bytes de memoria a la vez. Considere el número decimal 1,025. La representación hexadecimal de esta cifra, 0401H, requiere de dos bytes (o una palabra) de memoria. Consta de un byte de orden alto (más significativo), 04, y un byte de orden bajo (menos significativo), 01. El sistema almacena en memoria estos bytes en *secuencia inversa de bytes:* el byte de orden bajo en la dirección baja de memoria y el byte de orden alto en la dirección alta de memoria. Por ejemplo, el procesador transferiría 0401H de un registro a las localidades de memoria 5612 y 5613 como:




El procesador espera que los datos numéricos en la memoria estén en secuencia inversa de bytes y los procesa de acuerdo con esto. Cuando el procesador recupera la palabra de la memoria, otra vez invierte los bytes, restableciéndolos de manera correcta en el registro como 04 01 hex. Aunque esta característica es enteramente automática, usted tiene que estar alerta cuando programe y depure programas en lenguaje ensamblador.

Un programador de lenguaje ensamblador tiene que distinguir claramente entre la *dirección* y los *contenidos* de una localidad de memoria. En el ejemplo anterior, el contenido de la localidad 5612 es 01 y el contenido de la localidad 5613 es 04.

## SEGMENTOS Y DIRECCIONAMIENTO

Un *segmento* es un área especial en un programa que inicia en un *límite de un párrafo,* esto es, en una localidad regularmente divisible entre 16, o 10 hex. Aunque un segmento puede estar ubicado casi en cualquier lugar de la memoria y, en modo real, puede ser hasta de 64K, sólo necesita tanto espacio como el programa requiera para su ejecución.

Un segmento en modo real puede ser de hasta 64K. Se puede tener cualquier número de segmentos; para direccionar un segmento en particular basta cambiar la dirección en el registro del segmento apropiado. Los tres segmentos principales son los segmentos de *código,* de *datos* y de la *pila.*

### Segmento de código

El *segmento de código* (CS) contiene las instrucciones de máquina que son ejecutadas. Por lo común, la primera instrucción ejecutable está en el inicio del segmento, y el sistema operativo enlaza a esa localidad para iniciar la ejecución del programa. Como su nombre indica, el registro del CS direcciona el segmento de código. Si su área de código requiere más de 64K, su programa puede necesitar definir más de un segmento de código.

### Segmento de datos

El *segmento de datos* (DS) contiene datos, constantes y áreas de trabajo definidos por el programa. El registro del DS direcciona el segmento de datos. Si su área de datos requiere de más de 64K, su programa puede necesitar definir más de un segmento de datos.

### Segmento de la pila

En términos sencillos, la *pila* contiene los datos y direcciones que usted necesita guardar temporalmente o para uso de sus "llamadas" subrutinas. El registro del segmento de la pila (SS) direcciona el segmento de la pila.

### Límites de los segmentos

Los registros de segmentos contienen la dirección inicial de cada segmento. La figura 1-4 presenta un esquema de los registros CS, DS y SS; los registros y segmentos no necesariamente están en el orden mostrado. Otros registros de segmentos son el ES (segmento extra) y, en los procesadores 80386 y posteriores, los registros FS y GS, que tienen usos especializados.

Como ya dijimos, un segmento inicia en un límite de párrafo, que es una dirección por lo común divisible entre el 16 decimal, o 10 hex. Suponga que un segmento de datos inicia en la localidad de memoria 045F0H. Ya que en este y todos los demás casos el último dígito hexadecimal de la derecha es cero, los diseñadores de computadora decidieron que sería innecesario almacenar el dígito cero en el registro del segmento. Así, 045F0H se almacena como 045F, con el cero de la extrema derecha sobrentendido. En donde sea apropiado, el texto indica al cero de la derecha con corchetes, como en 045FfO].

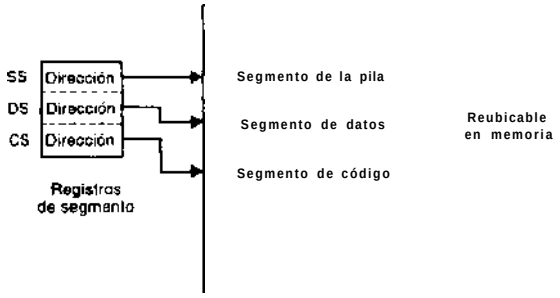



**Figura 1-4 Segmentos y registros**

### Desplazamientos de segmentos

En un programa, todas las localidades de memoria están referidas a una dirección inicial de segmento. La distancia en bytes desde la dirección del segmento se define como el *desplazamiento* (offset). Un desplazamiento de dos bytes (16 bits) puede estar en el rango de 000OH hasta FFFFH, o bien, desde cero hasta 65,535. Así, el primer byte del segmento de código tiene un desplazamiento 00, el segundo byte tiene un desplazamiento 01, etc., hasta el desplazamiento 65,535. Para referir cualquier dirección de memoria en un segmento, el procesador combina la dirección del segmento en un registro de segmento con un valor de desplazamiento.

En el ejemplo siguiente, el registro DS contiene la dirección de segmento del segmento de datos en 045F[0] hexadecimal y una instrucción hace referencia a una localidad con un desplazamiento de 0032H bytes dentro del segmento de datos.

| I | I |
|---|---|
| **dirección de segmento 045F0H** | **desplazamiento 32H** |

Por tanto, la localidad real de memoria del byte referido por la instrucción es 04622H:

| | |
|---|---|
| Dirección del segmento DS: | 045F0H |
| Desplazamiento: | +0032H |
| Dirección real: | 04622H |

Note que un programa tiene uno o más segmentos, los cuales pueden iniciar casi en cualquier lugar de memoria, variar en tamaño y estar en cualquier orden.

### Capacidad de direccionamiento

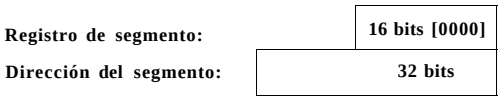
La serie de PC ha usado varios procesadores Intel que proporcionan diferentes capacidades de direccionamiento.

**Direccionamiento de 8086/8088.** Los registros de los procesadores 8086/8088 proporcionan 16 bits. Ya que una dirección de segmento está en el límite de un párrafo, los 4 bits de la extrema derecha de su dirección son cero. Como ya vimos, una dirección es almacenada en un registro de segmento, y la computadora asume los cuatro últimos bits de la derecha como ceros (un dígito hexadecimal), como nnnn[0] hex. Ahora, FFFF[0]H permite direccionar hasta 1,048,560 bytes. Si tiene duda, decodifique cada F hex como el 1111 binario, considere los cuatro últimos bits de la derecha como ceros y sume los valores de los bits a 1.

**Direccionamiento 80286.** En modo real, el procesador 80286 maneja el direccionamiento de la misma manera que lo hace el 8086. En modo protegido, el procesador utiliza 24 bits para direccionamiento, de manera que FFFFF[0] permite direccionar hasta 16 millones de bytes. Los registros de segmento actúan como seleccionadores para accesar una dirección de segmento de 24 bits de la memoria y sumar este valor a un desplazamiento de dirección de 16 bits:

| | | |
|---|---|---|
| **Registro de segmento:** | | **16 bits [0000]** |
| **Dirección del segmento:** | **24 bits** | |

**Direccionamiento 80386/486/586.** En modo real, estos procesadores manejan el direccionamiento de forma muy parecida a como lo hace un 8086. En modo protegido, los procesadores utilizan 48 bits para el direccionamiento, lo que permite direcciones de segmento de hasta cuatro mil millones de bytes. Los registros de segmento de 16 bits actúan como seleccionadores para el acceso a direcciones de segmento de 32 bits de la memoria y para agregar este valor a un desplazamiento de dirección de 32 bits:




## REGISTROS

Los *registros* del procesador se emplean para controlar instrucciones en ejecución, manejar direccionamiento de memoria y proporcionar capacidad aritmética. Los registros son direccionables por medio de un nombre. Los bits, por convención, se numeran de derecha a izquierda, como en:

... 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

### Registros de segmento

Un *registro de segmento* tiene 16 bits de longitud y facilita un área de memoria para direccionamiento conocida como el segmento actual. Como hemos dicho, un segmento se alinea en un límite de párrafo y su dirección en un registro de segmento supone cuatro bits 0 a su derecha.

**Registro CS.** El DOS almacena la dirección inicial del segmento de código de un programa en el registro CS. Esta dirección de segmento, más un valor de desplazamiento en el registro de apuntador de instrucción (IP), indica la dirección de una instrucción que es buscada para su ejecución. Para propósitos de programación normal, no se necesita referenciar el registro CS.

**Registro DS.** La dirección inicial de un segmento de datos de programa es almacenada en el registro DS. En términos sencillos, esta dirección, más un valor de desplazamiento en una instrucción, genera una referencia a la localidad de un byte específico en el segmento de datos.

**Registro SS.** El registro SS permite la colocación en memoria de una pila, para almacenamiento temporal de direcciones y datos. El DOS almacena la dirección de inicio del segmento de pila de un programa en el registro SS. Esta dirección de segmento, más un valor de desplazamiento en el registro del apuntador de la pila (SP), indica la palabra actual en la pila que está siendo direccionada. Para propósitos de programación normal, no se necesita referenciar el registro SS.

**Registro ES.** Algunas operaciones con cadenas de caracteres (datos de caracteres) utilizan el registro extra de segmento para manejar el direccionamiento de memoria. En este contexto, el registro ES está asociado con el registro DI (índice). Un programa que requiere el uso del registro ES puede inicializarlo con una dirección de segmento apropiada.

**Registros FS** y **GS.** Son registros extra de segmento en los procesadores 80386 y posteriores.

### Registro de apuntador de instrucciones

El registro apuntador de instrucciones (IP) de 16 bis contiene el desplazamiento de dirección de la siguiente instrucción que se ejecuta. El IP está asociado con el registro CS en el sentido de que el IP indica la instrucción actual dentro del segmento de código que se está ejecutando actualmente. Por lo común, usted no refiere el registro IP en un programa, pero, para probar un programa, sí puede cambiar su valor por medio del programa DEBUG del DOS. Los procesadores 80386 y posteriores tienen un IP ampliado de 32 bits, llamado EIP.

En el ejemplo siguiente, el registro CS contiene 25A4[0]H y el IP contiene 412H. Para encontrar la siguiente instrucción que será ejecutada, el procesador combina las direcciones en el CS y el IP:

| | |
|---|---|
| Segmento de dirección en el registro CS: | 25A40H |
| Desplazamiento de dirección en el registro IP: | + 412H |
| Dirección de la siguiente instrucción: | 25E52H |

### Registros apuntadores

Los registros SP (apuntador de la pila) y BP (apuntador base) están asociados con el registro SS y permiten al sistema accesar datos en el segmento de la pila.

**Registro** SP. El apuntador de la pila de 16 bits está asociado con el registro SS y proporciona un valor de desplazamiento que se refiere a la palabra actual que está siendo procesada en la pila. Los procesadores 80386 y posteriores tienen un apuntador de pila de 32 bits, el registro ESP. El sistema maneja de manera automática estos registros.

En el ejemplo siguiente, el registro SS contiene la dirección de segmento 27B3[0]H y el SP, el desplazamiento 312H. Para encontrar la palabra actual que está siendo procesada en la pila, la computadora combina las direcciones en el SS y el SP:

| | |
|---|---|
| Dirección de segmento en el registro SS: | 27B30H |
| Desplazamiento en el registro SP: | + 312H |
| Dirección en la pila: | 27E42H |

**Registro BP.** El BP de 16 bits facilita la referencia de parámetros, los cuales son datos y direcciones transmitidos vía la pila. Los procesadores 80386 y posteriores tienen un BP ampliado de 32 bits llamado el registro EBP.

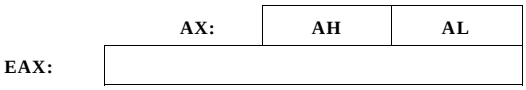
### Registros de propósito general

Los registros de propósito general AX, BX, CX y DX son los caballos de batalla del sistema. Son únicos en el sentido de que se puede direccionarlos como una palabra o como una parte de un byte. El último byte de la izquierda es la parte "alta", y el último byte de la derecha es la parte "baja". Por ejemplo, el registro CX consta de una parte CH (alta) y una parte CL (baja), y usted puede referirse a cualquier parte por su nombre. Las instrucciones siguientes mueven ceros a los registros CX, CH y CL, respectivamente.

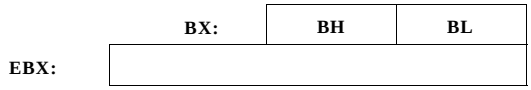
```
MOV   CX,00
MOV   CH,00
MOV   CL,00
```

Los procesadores 80386 y posteriores permiten el uso de todos los registros de propósito general, más sus versiones ampliadas de 32 bits: EAX, EBX, ECX y EDX.

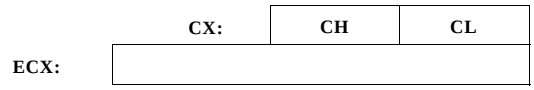
**Registro AX.** El registro AX, el acumulador principal, es utilizado para operaciones que implican entrada/salida y la mayor parte de la aritmética. Por ejemplo, las instrucciones para multiplicar, dividir y traducir suponen el uso del AX. También, algunas operaciones generan código más eficiente si se refieren al AX en lugar de a los otros registros.



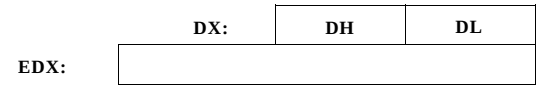

**Registro BX.** El BX es conocido como el registro base ya que es el único registro de propósito general que puede ser un índice para direccionamiento indexado. También es común emplear el BX para cálculos.




**Registro CX.** El **CX** es conocido como el registro contador. Puede contener un valor para controlar el número de veces que un ciclo se repite o un valor para corrimiento de bits, hacia la derecha o hacia la izquierda. El **CX** también es usado para muchos cálculos.




**Registro DX.** El DX es conocido como el registro de datos. Algunas operaciones de entrada/salida requieren su uso, y las operaciones de multiplicación y división con cifras grandes suponen al DX y al AX trabajando juntos.




Puede usar los registros de propósito general para suma y resta de cifras de 8, 16 o 32 bits.

**Registros índice**

Los registros SI y DI están disponibles para direccionamiento indexado y para sumas y restas.

**Registro SI.** El registro índice fuente de 16 bits es requerido por algunas operaciones con cadenas (de caracteres). En este contexto, el SI está asociado con el registro DS. Los procesadores 80386 y posteriores permiten el uso de un registro ampliado de 32 bits, el ESI.

**Registro DI.** El registro índice destino también es requerido por algunas operaciones con cadenas de caracteres. En este contexto, el DI está asociado con el registro ES. Los procesadores 80386 y posteriores permiten el uso de un registro ampliado de 32 bits, el EDI.

### Registro de banderas

De los 16 bits del registro de banderas, nueve son comunes a toda la familia de procesadores 8086, y sirven para indicar el estado actual de la máquina y el resultado del procesamiento. Muchas instrucciones que piden comparaciones y aritmética cambian el estado de las banderas, algunas de cuyas instrucciones pueden realizar pruebas para determinar la acción subsecuente.

En resumen, los bits de las banderas comunes son como sigue:

**OF (overflow, desbordamiento).** Indica desbordamiento de un bit de orden alto (más a la izquierda) después de una operación aritmética.

**DF (dirección).** Designa la dirección hacia la izquierda o hacia la derecha para mover o comparar cadenas de caracteres.

**IF (interrupción).** Indica que una interrupción externa, como la entrada desde el teclado, sea procesada o ignorada.

**TF (trampa).** Permite la operación del procesador en modo de un paso. Los programas depuradores, como DEBUG, activan esta bandera de manera que usted pueda avanzar en la ejecución de una sola instrucción a un tiempo, para examinar el efecto de esa instrucción sobre los registros y la memoria.

**SF (signo).** Contiene el signo resultante de una operación aritmética (0 = positivo y 1 = negativo).

**ZF (cero).** Indica el resultado de una operación aritmética o de comparación (0 = resultado diferente de cero y 1 = resultado igual a cero).

**AF (acarreo auxiliar).** Contiene un acarreo externo del bit 3 en un dato de ocho bits, para aritmética especializada.

**PF (paridad).** Indica paridad par o impar de una operación en datos de ocho bits de bajo orden (más a la derecha).

**CF (acarreo).** Contiene el acarreo de orden más alto (más a la izquierda) después de una operación aritmética; también lleva el contenido del último bit en una operación de corrimiento o de rotación.

Las banderas están en el registro de banderas en las siguientes posiciones:

| Núm. de bit: | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Bandera: | | | | | O | D | I | T | S | Z | | A | | P | | C |

Las banderas más importantes para la programación en ensamblador son O, S, Z y C, para operaciones de comparación y aritméticas, y D para operaciones de cadenas de caracteres. Los procesadores 80286 y posteriores tienen algunas banderas usadas para propósitos internos, en especial las que afectan al modo protegido. Los procesadores 80386 y posteriores tienen un registro extendido de banderas conocido como Eflags. El capítulo 8 contiene detalles adicionales acerca del registro de banderas.

## PUNTOS CLAVE

- La computadora distingue entre bits 0 (apagado) y 1 (encendido), y realiza aritmética sólo en formato binario.
- El valor de un número binario se determina por la ubicación de sus bits. Así, 1011 binario es igual a $2^3 + 2^2 + 0 + 2^0$, o 13.
- Un número binario negativo se representa en notación de complemento a dos: se invierten los bits de su representación positiva y se suma 1.
- Un solo carácter de memoria es un byte; comprende ocho bits de datos y un bit de paridad. Dos bytes adyacentes comprenden una palabra, y cuatro bytes adyacentes, una palabra doble.
- El valor de K es igual a $2^{10}$, o 1,024 bytes.
- El formato hexadecimal es una notación abreviada para representar grupos de cuatro bits. Los dígitos hexadecimales 0-9 y A-F representan los números binarios desde 0000 hasta 1111.
- La representación de datos de caracteres es realizado en el formato ASCII.
- El corazón de la PC es el microprocesador. El procesador almacena datos numéricos en palabras de memoria en secuencia inversa de bytes.
- Los dos tipos de memoria son ROM y RAM.
- Un programa en lenguaje ensamblador consiste en uno o más segmentos: un segmento de la pila para mantener las direcciones de regreso, un segmento de datos para definir áreas de datos y de trabajo y un segmento de código para instrucciones ejecutables. Las localidades en un segmento son expresadas como un desplazamiento relativo a la dirección inicial del segmento.
- Los registros de CS, DS y SS permiten el direccionamiento de los segmentos de código, datos y de la pila, respectivamente.
- El registro IP contiene la dirección de desplazamiento de la siguiente instrucción que es ejecutada.
- Los registros de apuntador SP y BP están asociados con el registro SS y permiten al sistema accesar datos en el segmento de la pila.
- Los registros de propósito general AX, BX, CX y DX son los caballos de batalla del sistema. El último byte a la izquierda es la parte "alta", y el último byte a la derecha es la parte "baja". El AX (acumulador principal) se emplea para entrada/salida y para la mayor parte de la aritmética. El BX (registro base) puede ser usado como un índice en direccionamiento extendido. El CX es conocido como el registro contador y el DX como el registro de datos.
- Los registros SI y DI están disponibles para direccionamiento extendido y para sumas y restas. Estos registros también se necesitan para algunas operaciones con cadenas de caracteres (carácter).

- El registro de banderas indica el estado actual de la computadora y los resultados de la ejecución de las instrucciones.

PREGUNTAS

1-1. Determine la configuración binaria en bits de los siguientes números: (a) 6; (b) 14; (c) 22; (d) 28; (e) 30.

1-2. Sume los siguientes números binarios:

| (a) | (b) | (c) | (d) |
|---|---|---|---|
| 00010101 | 00111101 | 00011101 | 01010111 |
| 00001101 | 00101010 | 00000011 | 00111101 |

1-3. Halle el complemento a dos de los siguientes números binarios: (a) 00010110; (b) 00111101; (c) 00111100.

1-4. Encuentre el valor positivo (absoluto) de los siguientes números binarios negativos: (a) 11001000; (b) 10111101; (c) 11111110; (d) 11111111.

1-5. Determine la representación hexadecimal de los valores siguientes: (a) código ASCII de la letra Q; (b) código ASCII del número 7; (c) 01011101 binario; (d) 01110111 binario.

1-6. Sume los números hexadecimales siguientes:

| (a) | (b) | (c) | (d) | (e) |
|---|---|---|---|---|
| 23 A6 | 51FD | 7779 | EABE | FBAC |
| +0022 | +0003 | +0887 | +26C4 | +0CBE |

1-7. Determine la representación hexadecimal de los números decimales siguientes. Consulte el apéndice A para ver el método de conversión. También debe verificar su resultado al convertir el hexadecimal a binario y al sumar los bits de 1. (a) 19; (b) 33; (c) 89; (d) 255; (e) 4,095; (f) 63,398.

1-8. Proporcione la configuración ASCII, en bits, de los siguientes caracteres de un byte. Utilice el apéndice B como guía: (a) P; (b) p; (c) #; (d) 5.

1-9. ¿Cuál es objetivo del procesador?

1-10. ¿Cuáles son las dos clases principales de memoria en la PC y cuáles, sus principales usos?

1-11. Muestre cómo el sistema almacena 012345 hex como un valor en la memoria.

1-12. Explique lo siguiente: (a) segmento; (b) desplazamiento (offset); (c) límite de dirección.

1-13. ¿Cuáles son: (a) las tres clases de segmentos; (b) su tamaño máximo; y (c) el límite de dirección en el que ellos inician?

1-14. Señale el objetivo de cada uno de los tres registros de segmentos.

1-15. Explique qué registros se utilizan para los siguientes propósitos: (a) sumar y restar; (b) contar los ciclos; (c) multiplicar y dividir; (d) segmentos de direccionamiento; (e) indicación de un resultado igual a cero; (f) desplazamiento de dirección de una instrucción que se va a ejecutar.

1-16. Muestre el registro EAX y el tamaño y posición de AH, AL y AX en él.

1-17. Codifique las instrucciones en lenguaje de ensamblador para mover el número 25 a los registros siguientes: (a) CH; (b) CL; (c) CX; (d) ECX.

# CAPÍTULO 2

# Requerimientos de software de la PC

OBJETIVO

Explicar el ambiente general de software para la PC.

## INTRODUCCIÓN

En este capítulo describimos el ambiente de software de la PC: las funciones del DOS y sus componentes principales. Examinamos el proceso de arranque (cómo es que el sistema se autocarga cuando usted enciende su computadora) y consideramos cómo el sistema carga un programa para ejecutarlo, cómo utiliza la pila y cómo una instrucción en el segmento de código direcciona datos en el segmento de datos.

El capítulo se completa con la explicación básica del software y hardware de la PC y nos permite continuar con el capítulo 3, en donde cargamos programas clave en la memoria y los ejecutamos paso a paso.

## CARACTERÍSTICAS DEL SISTEMA OPERATIVO

El DOS es un sistema operativo que proporciona acceso general e independiente de los dispositivos a los recursos de la computadora. Los dispositivos que permite incluyen teclados, pantallas y unidades de disco. Por "independencia de dispositivos" debe entender que no es preciso dirigirse específicamente a los dispositivos, ya que el DOS y sus controladores de dispositivos pueden manejar las operaciones a nivel de dispositivo.

Entre las funciones del DOS que nos conciernen en este libro, están las siguientes:

- *Administración de archivos.* El DOS mantiene los directorios y archivos en los discos de sistema. Los programas crean y actualizan archivos, pero el DOS tiene la responsabilidad de administrar sus ubicaciones en el disco.
- *Entrada/salida (E/S).* Los programas solicitan datos de entrada al DOS o entregan información al DOS por medio de interrupciones. El DOS releva al programador de codificar a nivel de E/S.
- *Carga de programas.* Un usuario o programa solicita la ejecución de un programa; el DOS maneja los pasos necesarios para tener acceso al programa desde el disco, colocarlo en la memoria e inicializarlo para su ejecución.
- *Administración de la memoria.* Cuando el DOS carga un programa para su ejecución, asigna suficiente espacio en memoria para el código del programa y sus datos. Los programas pueden procesar datos dentro de su área de memoria, liberar memoria que no necesiten y solicitar memoria adicional.
- *Manejo de interrupciones.* El DOS permite a los usuarios instalar programas residentes en memoria que se adhieren al sistema de interrupciones para realizar funciones especiales.

### Organización del DOS

Los tres componentes principales del DOS son IO.SYS, MSDOS.SYS y COMMAND.COM.

El IO.SYS realiza las funciones de inicialización en el momento del arranque y también contiene importantes funciones de E/S y controladores de dispositivos que dan el soporte de E/S básico en el BIOS de ROM. Este componente está almacenado en disco como un archivo de sistema oculto y es conocido como IBMBI0.COM en el PC-DOS.

El MSDOS.SYS actúa como el núcleo *(kernel)* del DOS y se ocupa de la administración de archivos, de memoria y de entrada/salida. Este componente está almacenado en disco como un archivo de sistema y en el PC-DOS se conoce como IBMD0S.COM.

C0MMAND.COM es un procesador de comandos o *shell* que actúa como la interfaz entre el usuario y el sistema operativo. Muestra la indicación del DOS, monitorea el teclado y procesa los comandos del usuario, como borrado de un archivo o carga de un programa para su ejecución.

## EL PROCESO DE ARRANQUE

Encender la computadora provoca una "inicialización" (algunos le llaman "arranque en frío"). El procesador introduce un estado de restauración, limpia todas las localidades de memoria (es decir, coloca cero en todas ellas), realiza una verificación de paridad de la memoria y asigna al registro CS la dirección del segmento FFFF[0]H y al registro IP el desplazamiento cero. Por tanto, la primera instrucción a ejecutarse está en la dirección formada por la pareja CS:IP, que es FFFFOH, la cual es el punto de entrada al BIOS en ROM.

La rutina de BIOS que inicia en FFFFOH verifica los diferentes puertos para identificarlos e inicializa los dispositivos que están conectados a la computadora. Después el BIOS establece dos áreas de datos:

1. Una *tabla de servicios de interrupción,* que inicia en memoria baja en la localidad O y contiene las direcciones de las interrupciones que ocurren.
2. Un *área de datos de BIOS* que inicia en la localidad 40[0], que está estrechamente relacionada con los dispositivos conectados.

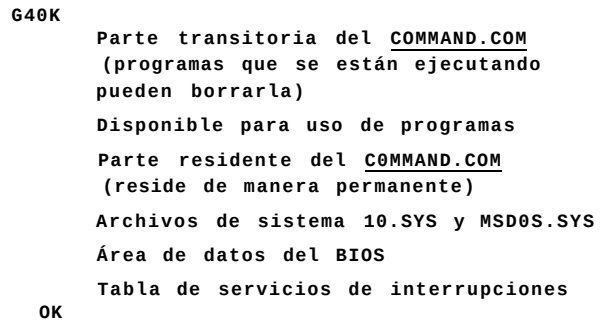



**Figura** 2-1 Mapa de la memoria convencional

A continuación el BIOS determina si está presente un disco que contenga los archivos de sistema del DOS y, en caso de que así sea, accesa el cargador de arranque desde ese disco. Este programa carga los archivos de sistema 10.SYS y MSDOS.SYS desde el disco hacia la memoria y transfiere el control al punto de entrada del IO.SYS, el cual contiene los controladores de dispositivos y otro código específico del hardware. El IO.SYS se reubica él mismo en memoria y transfiere el control al MSDOS.SYS. Este módulo inicializa las tablas internas del DOS y la porción del DOS de la tabla de interrupciones. También lee el archivo CONFIG.SYS y ejecuta sus comandos. Finalmente, el MSDOS.SYS pasa el control al C0MMAND.COM, el cual procesa el archivo AUTOEXEC.BAT, muestra su indicación y monitorea las entradas dadas desde el teclado.

En este punto, la memoria convencional hasta los 640K aparece como se muestra en la figura 2-1. Por medio de un administrador de memoria, parte del DOS puede ser reubicado en la memoria alta.

## INTERFAZ DOS-BIOS

El BIOS contiene un conjunto de rutinas en ROM para dar soporte a los dispositivos. El BIOS prueba e inicializa los dispositivos conectados y proporciona los servicios que son usados para la lectura y escritura desde los dispositivos. Una tarea del DOS es hacer interfaz con el BIOS cuando exista una necesidad de accesar estas facilidades.

Cuando un programa usuario solicita un servicio del DOS, éste podría transferir la solicitud al BIOS, el cual a su vez accesa el dispositivo solicitado. Sin embargo, algunas veces un programa hace la petición directamente al BIOS, específicamente para servicios del teclado y de la pantalla. Y en otras ocasiones -aunque es raro y no recomendable- un programa puede pasar por alto tanto al DOS como al BIOS para accesar un dispositivo directamente. La figura 2-2 muestra estas trayectorias alternas.

## PROGRAMA CARGADOR DEL SISTEMA

El DOS da soporte a dos tipos de programas ejecutables: .COM y .EXE. Un *programa .COM* consta de un segmento que contiene código, datos y la pila. Si se necesita un pequeño programa de utilería o un programa residente en memoria (un programa que es instalado permanentemente y está disponible mientras otros programas están ejecutándose), se escribe un programa .COM. Un *programa .EXE* consta de segmentos de código, datos y de la pila separados y es el método usado por la mayoría de los programas serios. Este libro usa ambos tipos de programas.

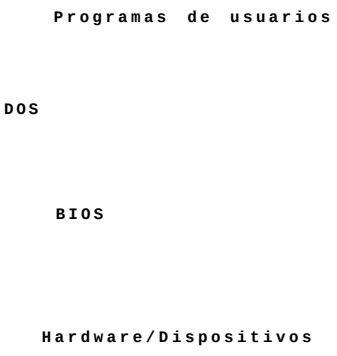



Figura 2-2 Interfaz DOS-BIOS

Cuando usted le solicita al DOS cargar un programa .EXE desde el disco a la memoria para su ejecución, el cargador realiza las siguientes operaciones:

1. Accesa el programa .EXE desde el disco.
2. Construye un prefijo de segmento de programa (PSP) de 256 bytes (100H) en un límite de párrafo en memoria interna disponible.
3. Almacena el programa en memoria inmediatamente después del PSP.
4. Carga la dirección del PSP en los registros DS y ES.
5. Carga la dirección del segmento de código en el CS y establece el IP al desplazamiento de la primer instrucción (por lo común cero) en el segmento de código.
6. Carga la dirección de la pila en el SS y establece el SP al tamaño de la pila.
7. Transfiere el control al programa para ejecución, iniciando (por lo común) con la primer instrucción en el segmento de código.

En esta forma, el cargador DOS inicializa correctamente los registros CS:IP y SS:SP. Pero note que el programa cargador almacena la dirección del PSP tanto en el registro DS como en el ES, aunque su programa normalmente necesita la dirección del segmento de datos en estos registros. Como consecuencia, sus programas tienen que inicializar el DS con la dirección del segmento de datos, como se verá en el capítulo 4.

Ahora examinaremos la pila y después los segmentos de código y datos.

## LA PILA (STACK)

Los programas .COM y .EXE, requieren un área en el programa reservada como una pila *(stack).* El propósito de la pila es mantener un espacio para el almacenamiento temporal de direcciones y datos.

El DOS define de manera automática la pila para un programa .COM, mientras que para un programa .EXE usted debe definir en forma explícita la pila. Cada elemento de dato en la pila es una palabra (dos bytes). El registro SS, como es inicializado por el DOS, contiene la dirección del inicio de la pila. Inicialmente, el SP contiene el tamaño de la pila, un valor que apunta al byte que está pasando el final de la pila. La pila difiere de otros segmentos en su método de almacenar los datos: empieza en la localidad más alta y almacena los datos hacia abajo por la memoria.

La instrucción PUSH (entre otras) disminuye el SP en 2 hacia abajo, hacia la siguiente palabra almacenada de la pila y coloca (o empuja, *push)* un valor ahí. La instrucción POP (entre otras) regresa el valor de la pila e incrementa el SP en 2 hacia arriba, hacia la siguiente palabra almacenada.

El ejemplo siguiente ilustra cómo meter el contenido de los registros AX y BX a la pila y la subsecuente extracción de ellos. Suponga que el AX contiene 015AH, el BX contiene 03D2H y el SP contiene 28H (aquí no nos concierne la dirección en el SS).

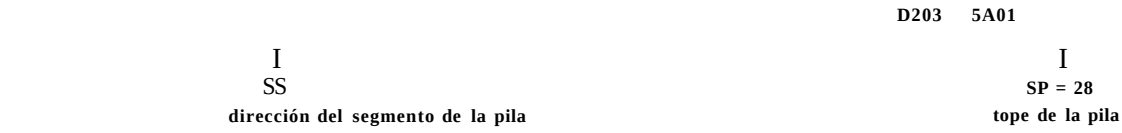
**1.** Al comienzo, la pila está vacía y se ve así:

PUSH AX: disminuye el SP en 2 (a 26H) y almacena el contenido del AX, 015AH, en la pila. Observe que la operación invierte la secuencia de bytes almacenados, de modo que 015A se convierte en 5A01:

**3.** PUSH BX: disminuye el SP en 2 (a 24H) y almacena el contenido del BX, 03D2H, en la pila:

**4.** POP BX: regresa la palabra que se encuentra en la pila, en donde apunta el SP, y la envía al registro BX e incrementa el SP en 2 (a 26H). El BX ahora contiene 03D2H, con los bytes correctamente invertidos:

POP AX: regresa la palabra que se encuentra en la pila, en donde apunta el SP, y la envía al registro AX e incrementa el SP en 2 (a 28H). El AX ahora contiene 015AH, con los bytes correctamente invertidos:




Note que las instrucciones POP son codificadas en secuencia inversa a las instrucciones PUSH. Así, en el ejemplo se guardaron AX y BX, pero se sacaron el BX y AX, en ese orden. Además, los valores sacados de la pila aún están allí, aunque el SP ya no apunta a ellos.

Siempre debe asegurarse que su programa coordine los valores que guarda en la pila con los valores que saca de ella. Como éste es un requisito directo, un error puede causar que un programa no funcione. También, para un programa .EXE usted tiene que definir una pila que sea suficientemente grande para contener todos los valores que podrían ser guardados en ella.

Otras instrucciones relacionadas con los valores que guarda y saca de la pila son:

- PUSHF y POPF: Guarda y restablece el estado de los banderas.
- PUSHA y POPA (para el 80286 y posteriores): Guarda y restaura el contenido de todos los registros de propósito general.

## DIRECCIONAMIENTO DE PROGRAMAS

Normalmente, los programadores escriben en código *simbólico* y utilizan ensamblador para traducirlo a *código de máquina.* Para ejecutar un programa, el DOS carga sólo código de máquina en la memoria. Cada instrucción consta de al menos una operación, como mover, sumar o regresar. Dependiendo de la operación, una instrucción también puede tener uno o más operandos que referencian los datos que la operación procesa.

Como se estudió en el capítulo 1, el registro CS proporciona la dirección de inicio de un segmento de código de programa y el registro DS ofrece la dirección de inicio del segmento de datos. El segmento de código contiene instrucciones que serán ejecutadas, mientras que el segmento de datos contiene los datos que las instrucciones referencian. El registro IP indica la dirección del desplazamiento de la instrucción actual, en el segmento de código, que es ejecutada. Un operando de la instrucción indica una dirección de desplazamiento en el segmento de datos que es referenciada.

Considere un ejemplo en el que el DOS ha determinado que se carga un programa .EXE en memoria, iniciando en la localidad 04AF0H. El DOS, de acuerdo con esto, asigna el registro CS la dirección del segmento 04AF[0]H y al DS con, digamos, la dirección de segmento 04B1[0]H. El programa ya ha iniciado su ejecución, y el IP actualmente contiene el desplazamiento 0023H. La pareja CS:IP determina la dirección de la siguiente instrucción a ser ejecutada, como sigue:

| | |
|---|---|
| Dirección del segmento CS: | 4AF0H |
| Desplazamiento IP: | +0013H |
| Dirección de la instrucción: | 4B03H |

Digamos que la instrucción que inicia en 04B03H copia los contenidos de un byte en memoria al registro AL; el byte está en el desplazamiento 0012H en el segmento de datos. Aquí están tanto el código de máquina como el código simbólico para esta operación:

```
                    A01200    MOV    AL, [0012]
                      |
Localidad  04B03H
```

La localidad de memoria 04B03H contiene el primer byte (A0) de la instrucción que el procesador accesa. El segundo y tercer bytes contienen el valor del desplazamiento, en secuencia invertida de bytes (0012 se convierte en 1200). Para accesar el elemento de dato, el procesador determina su localidad de la dirección del segmento en el registro DS más el desplazamiento (0012H) en el operando de la instrucción. Ya que el DS contiene 04B1[0]H, la localidad actual del elemento de dato referenciado es:

| | |
|---|---|
| Dirección del segmento DS: | 4B10H |
| Desplazamiento del segmento: | +0012H |
| Dirección del dato: | 4B22H |

Hagamos que la localidad 04B22H contenga 1BH. Entonces el procesador extrae el 1BH de la localidad 04B22H y la copia en el registro AL, como se muestra en la figura 2-3.

Cuando el procesador busca cada byte de la instrucción, incrementa el registro IP de manera que éste contenga el desplazamiento (0016H) para la siguiente instrucción. El procesador ahora está preparado para ejecutar la siguiente instrucción, la cual se deriva otra vez de la dirección del segmento en el CS (04AF0H) más el desplazamiento actual en el IP (0016H) -de hecho, 04B06H.

Una instrucción también puede accesar más de un byte a la vez. Por ejemplo, supongamos que una instrucción es almacenar los contenidos del registro AX (0567H) en dos bytes adyacentes en el segmento de datos empezando en el desplazamiento 0012H. El código simbólico es MOV [0012],AX. El operando [0012] entre corchetes (un operador de índice) indica una localidad de memoria para distinguirlo del simple número 12. El procesador carga los dos bytes en el AX en secuencia inversa de bytes como

| | | |
|---|---|---|
| Contenido de los bytes: | 67 | 05 |
| | **I** | **I** |
| Desplazamiento en el segmento de datos: | 0012 | 0013 |

Otra instrucción, MOV AX,[0012], puede recuperar subsecuentemente estos bytes para copiarlos de la memoria de regreso al AX. La operación invierte (y corrige) los bytes en el AX como 05 67.

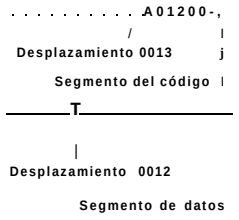



Figura 2-3 Segmentos y desplazamientos

## REFERENCIAS A MEMORIA Y A REGISTROS

Una característica para obtener claridad en las instrucciones es el uso de nombres de operandos, de nombres entre corchetes y de números. En los ejemplos siguientes, WORDA está definida como una palabra (dos bytes) en memoria:

```
WORDA                          ;Define una palabra


        MOV       AX,BX        /Mueve los contenidos de BX a AX
        MOV       AX, WORDA    /Mueve los contenidos de WORDA a AX
        MOV       AX, 25       /Mueve el valor 25 a AX
        MOV       AX, [BX]     /Mueve los contenidos de la localidad especificada
                               por BX
```

Los corchetes en el cuarto ejemplo definen un *operador de índice* que significa: utilizar una dirección de desplazamiento en el BX (combinada con la dirección del segmento en el DS, como DS:BX) para localizar una palabra en memoria y mover su contenido al AX. Compárese el efecto de esta instrucción con aquella del primer ejemplo, la cual simplemente mueve los contenidos del BX al AX.

## PUNTOS CLAVE

- Los tres componentes principales del DOS son IO.SYS, MSDOS.SYS y COMMAND.COM.
- Al encender la computadora se provoca una inicialización, también llamada "arranque en frío". El procesador introduce un estado de restauración, limpia todas las localidades de memoria poniéndolas en cero, realiza una verificación de la paridad de la memoria y establece los registros CS e IP al punto de entrada del BIOS en ROM.
- Los dos tipos de programas del DOS son .COM y .EXE.
- Cuando usted solicita al DOS cargar un programa .EXE para su ejecución, el DOS construye un PSP de 256 bytes (100H) en un límite de párrafo en memoria y almacena el programa inmediatamente después del PSP. Después carga la dirección del PSP en los registros DS y ES, carga la dirección del segmento de código en el CS, establece el IP al desplazamiento de la primera instrucción en el segmento de código, carga la dirección de la pila en el SS y establece el tamaño de la pila. Finalmente, el cargador transfiere el control al programa por ejecutarse.
- El propósito de la pila es proporcionar un espacio para el almacenamiento temporal de direcciones y datos. Cada dato en la pila es una palabra (dos bytes).
- El DOS define la pila para un programa .COM, mientras que para un programa .EXE se debe definir de manera explícita la pila.
- Cuando el procesador busca cada byte de una instrucción, incrementa el registro IP de manera que el IP contenga el desplazamiento para la siguiente instrucción.

## PREGUNTAS

2-1. ¿Cuáles son las cinco funciones principales del DOS?

2-2. ¿Cuáles son los tres componentes principales del DOS y cuál es el propósito de cada uno de ellos?

2-3. ¿Qué pasos realiza el sistema en una inicialización (arranque en frío)?

2-4. (a) ¿Qué área de datos construye el DOS y almacena en frente de un módulo ejecutable, cuando el módulo es cargado para su ejecución? (b) ¿Cuál es el tamaño de esta área de datos?

2-5. El DOS realiza ciertas operaciones cuando carga un programa .EXE para su ejecución. ¿Qué valores inicializa el DOS (a) en los registros CS e IP? (b) ¿en los registros SS y SP? (c) ¿en los registros DS y ES?

2-6. ¿Cuál es el objetivo de la pila?

2-7. ¿De qué forma se define la pila para (a) un programa .COM y (b) un programa .EXE? (Esto es, ¿quién o qué define la pila?)

2-**8.** (a) ¿Cuál es el tamaño de cada entrada de la pila? (b) ¿En dónde se encuentra inicialmente la parte superior de la pila y cómo es direccionada?

2-9. Durante la ejecución de un programa, el CS contiene 5A2B[0], el SS contiene 5B53[0], el IP contiene 52H y el SP contiene 48H. (Los valores se muestran en secuencia normal, no en secuencia invertida de bytes.) Calcule las direcciones de (a) la instrucción a ejecutarse y (b) la parte superior de la pila (localidad actual).

2-10. El DS contiene 5B24[0] y una instrucción que mueve datos de la memoria al AL es A03A01 (donde A0 significa "mover"). Calcule la dirección de memoria referenciada.

## CAPÍTULO 3

# Ejecución de instrucciones

OBJETIVO

Dar a conocer cómo introducir y ejecutar programas en la memoria.

### INTRODUCCIÓN

Este capítulo utiliza un programa del DOS llamado DEBUG, que permite visualizar la memoria, introducir programas en ella y rastrear su ejecución. El texto explica cómo se pueden introducir estos programas directamente en la memoria en un segmento de código y da una explicación de cada paso ejecutado. Algunos lectores pueden tener acceso a depuradores sofisticados, como CODEVIEW o TurboDebugger; sin embargo, usaremos DEBUG, ya que es sencillo de usar y está disponible en cualquier parte.

En los ejercicios iniciales se inspeccionan los contenidos de áreas particulares de la memoria. El primer programa de ejemplo utiliza datos "inmediatos" definidos dentro de las instrucciones para cargar datos en registros y realizar aritmética. El segundo programa de ejemplo utiliza datos definidos de forma separada en el segmento de datos. El rastreo de cómo se ejecutan estas instrucciones da una idea de la operación de una computadora y la función de los registros.

Usted puede empezar sin el conocimiento previo de un lenguaje ensamblador o de uno de programación. Todo lo que necesita es una IBM PC o compatible y un disco que contenga el sistema operativo DOS. No obstante, asumimos que está familiarizado con el arranque de la computadora, manejo de discos flexibles y la selección de discos y archivos.

## EL PROGRAMA DEBUG

El DOS viene con un programa llamado DEBUG que es utilizado para probar y depurar programas ejecutables. Una característica de DEBUG es que despliega todo el código del programa y los datos en formato hexadecimal, y cualquier dato que se introduzca a la memoria también está en formato hexadecimal. Otra característica es que DEBUG permite ejecutar un programa en *modo de paso sencillo (un paso a la vez),* de manera que se pueda ver el efecto de cada instrucción sobre las localidades de memoria y los registros.

### Comandos de DEBUG

DEBUG proporciona un conjunto de comandos que permiten realizar diferentes operaciones útiles. Los comandos que nos interesan en este momento son los siguientes:

| | |
|---|---|
| A | Ensamblar instrucciones simbólicas y pasarlas a código de máquina. |
| D | Mostrar el contenido de un área de memoria. |
| E | Introducir datos en memoria, iniciando en una localidad específica. |
| G | Correr el programa ejecutable que se encuentra en memoria. |
| N | Nombrar un programa. |
| P | Proceder o ejecutar un conjunto de instrucciones relacionadas. |
| Q | Salir de la sesión con DEBUG. |
| R | Mostrar el contenido de uno o más registros. |
| T | Rastrear la ejecución de una instrucción. |
| **U** | "Desensamblar" código de máquina y pasarlo a código simbólico. |
| W | Escribir o grabar un programa en disco. |

### Reglas de los comandos de DEBUG

Para sus propósitos, DEBUG no distingue entre letras minúsculas y mayúsculas, de manera que se pueden introducir comandos de cualquier forma. También se introduce un espacio sólo en donde sea necesario separar parámetros en un comando. Los tres ejemplos siguientes utilizan el comando D de DEBUG para mostrar la misma área de memoria, iniciando en el desplazamiento 200H en el segmento de datos (DS):

```
D DS:200      (comando en mayúsculas, con un espacio en blanco después de él)
DDS:200       (comando en mayúsculas, con un espacio en blanco después de él)
dds:200       (comando en minúsculas, sin espacio en blanco después de él)
```

Note que especifica segmentos y desplazamientos con dos puntos (:), en la forma segmento:desplazamiento. Además, DEBUG supone que todos los números están en formato hexadecimal.

### El despliegue de DEBUG

El despliegue de DEBUG consiste en tres partes. A la izquierda está la dirección hexadecimal del último byte de la izquierda que se despliega en la forma segmento desplazamiento. El área amplia del centro es la representación hexadecimal del área desplegada. A la derecha está la representación en ASCII de los bytes que contienen caracteres desplegables, los cuales pueden ayudarlo a interpretar el área hexadecimal. En forma de diagrama tenemos:

```
Dirección  |<_______Representación hexadecimal_________>|<–ASCII–>|

  xxxx:xxl0   xx xx-xx xx x x

  xxxx:xx20   xx xx-xx xxx x

  xxxx:xx30   xx xx-xx xxx x
```

Cada línea despliega 16 bytes de memoria. La dirección de la izquierda se refiere sólo al último byte de la izquierda, en la forma segmento desplazamiento; puede contar atravesando la línea para determinar la posición de cada byte. El área de representación hexadecimal muestra dos caracteres hexadecimales por cada byte, seguidos por un espacio en blanco por legibilidad. Además, un guión separa a los segundos ocho bytes de los primeros ocho, otra vez por legibilidad. Así, si usted necesita localizar el byte en el desplazamiento xxl3H, inicie con xxlOH y cuente tres bytes sucesivos a la derecha.

Este libro hace un uso considerable de DEBUG y explica en detalle sus comandos conforme se necesitan. El apéndice E proporciona una descripción completa de los comandos de DEBUG.

### Inicio con DEBUG

Para empezar con DEBUG, coloque el sistema en el directorio del disco duro que contenga DEBUG o bien inserte un disco flexible con el DOS que contenga el DEBUG en la unidad por omisión. Para iniciar el programa, teclee la palabra DEBUG y presione la tecla Enter. DEBUG debe cargarse del disco a la memoria. Cuando el indicador de DEBUG, un guión (-), aparezca en la pantalla, DEBUG está listo para recibir sus comandos (esto *es* un guión, aunque parezca el cursor). Ahora usemos DEBUG para curiosear por la memoria.

## VISUALIZACIÓN DE LAS LOCALIDADES DE MEMORIA

En nuestro primer ejercicio, usted usará DEBUG para ver el contenido de localidades seleccionadas de la memoria. El único comando por el que estará interesado en este ejercicio es D (Display, mostrar), el cual lista ocho líneas de 16 bytes cada una y muestra su representación hexadecimal y ASCII.

### Verificación del equipo del sistema

Primero veamos qué es lo que ha determinado el BIOS que tiene instalado su equipo. Una palabra del estado del equipo en el área de datos del BIOS, ofrece una indicación rudimentaria de los dispositivos instalados. Esta palabra está en las localidades 410H-411H, que puede ver desde DEBUG por medio de una dirección de dos partes: 40 para la dirección del segmento (se sobrentiende el último cero) y 10 para el desplazamiento desde la dirección del segmento. Lea la dirección 40:10 como segmento 40[0]H más un desplazamiento de 10H. Teclee de manera exacta lo siguiente:

```
D 40:10 [y presione la tecla Enter]
```

El despliegue debe empezar con algo como esto:

```
0040:0010 63 44 . . . . . .
```

En este ejemplo, los dos bytes en la palabra del estado del equipo contienen los valores hexadecimales 63 y 44. Invierta los bytes (44 63) y conviértalos a binario:

| Bit: | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Binario: | 0 | | 1 | 0 | 0 | 0 | | 1 | 0 | | 0 | 0 | | 1 | 1 | 0 | 0 | 0 | | | 1 | 1 |

A continuación está una explicación del código hexadecimal:

| BITS | DISPOSITIVO |
|---|---|
| 15,14 | Número de puertos paralelos para impresora conectados = 1 (binario 01) |
| 11-9 | Número de puertos seriales conectados = 2 (binario 010) |
| 7,6 | Número de dispositivos de disco flexible = 2 (donde 0 0 = l , 0 1 = 2 , 1 0 = 3y 11 = 4) |
| 5,4 | Modo inicial de video = 10 (donde 01 = 40 x 25 en color, 10 = 80 x 12 25 en color y 11 = 80 x 12 25 monocromático) |
| **1** | 1 = coprocesador matemático está presente |
| 0 | 1 = unidad de disco flexible está presente |

Los bytes no citados no son usados.
Puede permanecer en DEBUG para el siguiente ejercicio o introduzca Q para salir.

### Verificación del tamaño de la memoria

El siguiente paso es examinar la cantidad de memoria que el DOS "piensa" que tiene instalada. Dependiendo del modelo de su computadora, el valor puede estar basado en interruptores internos y puede indicar menos memoria de la que realmente está instalada. El valor está en el área de datos del BIOS en las localidades 413H y 414H. Teclee lo siguiente exactamente como lo ve:

```
D 40:13  [y presione Enter]
```

El despliegue debe empezar con algo como esto:

```
0040:0013. . . . . xx xx . . -
```

Los primeros dos bytes mostrados en el desplazamiento 0013H son los kilobytes de memoria en hexadecimal, con los bytes en secuencia inversa. Aquí están dos ejemplos que muestran hexadecimales en orden inverso, hexadecimales corregidos y el equivalente en decimal:

| HEXADECIMAL INVERSO | HEXADECIMAL CORREGIDO | DECIMAL (K) |
|---|---|---|
| 00 02 | 02 00 | 512 |
| 80 02 | 02 80 | 640 |

### Verificación del número de serie y de la nota de derechos reservados

El número de serie de la computadora está alojado en el ROM de BIOS en la localidad FE000H, Para verlo, teclee

```
D FE00:0  [y presione Enter]
```

La pantalla debe mostrar un número de serie de siete dígitos seguido, en máquinas convencionales, de una nota de derechos reservados. El número de serie se muestra como número hexadecimal, mientras que la nota de derechos reservados es más reconocible en el área ASCII a la derecha. La

nota de derechos reservados puede continuar pasando sobre lo que ya está mostrado; para verla, basta con presionar D, seguida de la tecla Enter.

### Verificación de la fecha en el ROM BIOS

La fecha de fabricación de su ROM BIOS inicia en la localidad FFFF5H, registrada como mm/dd/aa. Para verla, teclee

```
D FFFF:5 [y presione Enter]
```

El conocimiento de esta fecha puede ser útil para determinar la edad y modelo de la computadora.

### Verificación de la identificación del modelo

Inmediatamente después de la fecha de fabricación del ROM BIOS está la identificación del modelo en la localidad FFFFEH, o FFFF:E. Aquí están varias identificaciones de modelos:

| CÓDIGO | MODELO |
|---|---|
| F8 | PS/2 modelos 70 y 80 |
| F9 | PC convertible |
| FA | PS/2 modelo 30 |
| FB | PC-XT (1986) |
| FC | PC-AT (1984), PC-XT modelo 286, PS/2 modelos 50 y 60, etcétera |
| FE | PC-XT (1982), portátil (1982) |
| FF | Primera IBM PC |

Ahora que ya sabe cómo usar el comando para desplegar información, puede ver el contenido de cualquier localidad de almacenamiento. También puede avanzar por la memoria con sólo presionar D de forma repetida: DEBUG muestra de manera sucesiva ocho líneas, continuando a partir de la última operación D.

Cuando haya terminado de curiosear, introduzca Q (por Quit), para salir de DEBUG o continúe con el ejercicio siguiente.

## EJEMPLO I DE LENGUAJE DE MÁQUINA: DATOS INMEDIATOS

Ahora usemos DEBUG para introducir el primero de dos programas directamente en memoria y rastrear su ejecución. Ambos programas ilustran un sencillo código de lenguaje de máquina y cómo aparece en el almacenamiento principal y los efectos de su ejecución. Para este propósito, empezaremos con el comando DEBUG E (Enter, introducir). Sea muy cuidadoso en su uso, ya que introducir datos incorrectos o en una localidad equivocada puede causar resultados impredecibles. No es probable que cause daños, pero puede sorprenderse y perder datos que haya introducido durante la sesión de DEBUG.

El primer programa utiliza *datos inmediatos,* datos definidos como parte de una instrucción. Mostramos el lenguaje de máquina en formato hexadecimal y para legibilidad en código simbólico, junto con una explicación. Para la primera instrucción, el código simbólico es MOV AX,0123, la cual mueve (o copia) el valor 0123H al registro AX (no tiene que definir un valor inmediato en

secuencia inversa de byte). MOV es la instrucción, el registro AX es el primer operando y el valor inmediato 0123H es el segundo operando.

| INSTRUCCIÓN DE MÁQUINA | CÓDIGO SIMBÓLICO | EXPLICACIÓN |
|---|---|---|
| B82301 | MOV AX,0123 | Mover el valor 0123H a AX. |
| 052S00 | ADD AX,0025 | Sumar el valor 0025H a AX. |
| 8BD8 | MOV BX, AX | Mover el contenido de AX a BX |
| 03D8 | ADD BX, AX | Sumar el contenido de AX a BX |
| 8BCB | MOV CX, BX | Mover el contenido de BX a CX |
| 2BC8 | SUB CX, AX | Restar el contenido de AX del |
| 2BC0 | SUB AX, AX | Restar AX de AX (limpiar AX). |
| 90 | NOP | No operación (no hacer nada). |

Puede haber notado que las instrucciones de máquina pueden tener uno, dos o tres bytes de longitud. El primer byte es la operación real y cualesquiera otros bytes, si están presentes, son operandos: referencia a un valor inmediato, un registro o una localidad de memoria. La ejecución del programa empieza con la primera instrucción de máquina y avanza por cada instrucción, una después de otra. Al llegar a este punto no esperamos que tenga mucho sentido el código de máquina. Por ejemplo, en un caso el código de máquina (el primer byte) para mover es B8 hex y en otro caso el código para mover es 8B hex.

### Cómo introducir instrucciones de programa

Iniciamos este ejercicio como lo hicimos con el anterior: teclee el comando DEBUG y presione Enter. Cuando DEBUG está cargado por completo, despliega su indicación (-). Para introducir este programa directamente en memoria, sólo teclee la parte de lenguaje de máquina, pero no el código simbólico o la explicación. Teclee el siguiente comando E (Enter), incluso los espacios en blanco en dónde se indican:

```
E CS:1000 B8 23 01 05 25 00 [presione Enter]
```

CS:100 indica la dirección de memoria inicial en la que los datos se almacenarán -100H (256) bytes siguiendo al inicio del segmento de código (la dirección de inicio usual para el código de máquina con DEBUG). El comando E hace que DEBUG almacene cada par de dígitos hexadecimales en un byte de memoria, desde CS: 100 hasta CS: 105.

El siguiente comando E almacena seis bytes, empezando en CS: 106 a 107, 108, 109, 10A y 10B:

```
E CS:106 8B D8 03 D8 8B CB [seguido por Enter]
```

El último comando E almacena cinco bytes, iniciando en CS:10C a 10D, 10E, 10F y 110:

```
E CS:10C 2B C8 2B C0 90 [seguido por Enter]
```

Si teclea un comando de manera incorrecta, sólo repítalo con los valores correctos.

```
-E CS:100 B8 23 01 05 25 00
-E CS:106 8B D8 03 D8 8B CB
-E CS:10C 2B C8 2B CO 90
-R
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS = 21C1 ES = 21C1 SS = 21C1 CS = 21C1 IP = 0100 NV UP El PL NZ NA PO NC
21C1:0100 B82301         MOV     AX,0123
-T

AX=0123  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=21C1  ES=21C1  SS=21C1  CS=21C1  IP=0103   NV UP El PL NZ NA PO NC
21C1:0103 052500         ADD     AX.0025
-T

AX=0148  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=21C1  ES=21C1  SS=21C1  CS=21C1  IP=0106   NV UP El PL NZ NA PE NC
21C1:0106 8BD8           MOV     BX,AX
-T

AX=0148  BX=0148  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=21C1  ES=21C1  SS=21C1  CS=21C1  IP=0108   NV UP El PL NZ NA PE NC
21C1:0108 03D8           ADD     BX,AX
-T

AX=0148  BX=0290  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=21C1  ES=21C1  SS=21C1  CS=21C1  IP=010A   NV UP El PL NZ AC PE NC
21C1:010A 8BCB           MOV     CX,BX
-T

AX=0148  BX=0290  CX=0290  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=21C1  ES=21C1  SS=21C1  CS=21C1  IP=010C   NV UP El PL NZ AC PE NC
21C1:010C 2BC8           SUB     CX,AX
-T

AX=0148  BX=0290  CX=0148  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=21C1  ES=21C1  SS=21C1  CS=21C1  IP=010E   NV UP El PL NZ AC PE NC
21C1:010E 2BC0           SUB     AX,AX
-T

AX=0000  BX=0290  CX=0148  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=21C1  ES=21C1  SS=21C1  CS=21C1  IP=0110   NV UP El PL ZR NA PE NC
21C1:0110 90             NOP
```

Figura 3-1 Rastreo de las instrucciones de máquina

### Ejecución de instrucciones de programa

Ahora es algo sencillo ejecutar las instrucciones anteriores, una a la vez. La figura 3-1 muestra todos los pasos, incluyendo los comandos E. Su pantalla debe mostrar resultados semejantes cuando introduzca cada comando DEBUG. Al mismo tiempo, puede ver el contenido de los registros después de cada instrucción. Los comandos DEBUG que nos conciernen aquí son R (registro) y T(trace, rastreo).

Para ver los contenidos iniciales de los registros y las banderas, teclee el comando R, seguido por la tecla Enter. DEBUG muestra el contenido de los registros en formato hexadecimal, por ejemplo,

```
AX=0000 BX=0000 . . .
```

A causa de las diferencias entre las distintas versiones del DOS, el contenido de algunos registros en su pantalla pueden diferir de los que muestra en la figura 3-1. El registro IP muestra IP=0100, indicando que la ejecución de instrucciones inicia 100H bytes después del inicio del segmento de código (por esto se usó E CS:100 para introducir el inicio del programa).

El registro de banderas en la figura 3-1 muestra la siguiente configuración:

```
NV UP El PL NZ NA PO NC
```

Esta configuración significa no desbordamiento, dirección hacia arriba (o hacia la derecha), interrupción habilitada, signo positivo, no cero, no acarreo auxiliar, paridad impar y no acarreo, respectivamente. En este momento, ninguno de estos valores es importante para nosotros.

El comando R también muestra en el desplazamiento 0100H la primera instrucción que es ejecutada. Note que en la figura el registro CS contiene 21C1. Ya que es seguro que su dirección de segmento CS, difiera de ésta, la mostraremos como xxxx para las instrucciones:

```
xxxx:0100 B82301 MOV AX,0123
```

- xxxx indica el inicio del segmento de código como xxxx[0]. El valor xxxx:0100 significa desplazarse 100H bytes después de la dirección del segmento CS xxxx[0].
- B82301 es el código de máquina que usted introdujo en CS:100.
- MOV AX,0123 es la instrucción simbólica en ensamblador para el código de máquina. Esta instrucción significa, en realidad, mover el valor inmediato 0123H al registro AX. DEBUG ha "desensamblado" las instrucciones de máquina de manera que usted pueda interpretarlas de manera más fácil. En capítulos posteriores, codificará exclusivamente instrucciones en código ensamblador.

En este momento, la instrucción MOV no ha sido ejecutada. Para ese propósito, teclee T (trace, rastrear) y presione la tecla Enter. El código de máquina es B8 (mover al registro AX) seguido por 2301. La operación mueve el 23 a la mitad baja (AL) del registro AX y el 01 a la mitad alta (AH) del registro AX:

```
        AH   AL
AX:     01   23
```

DEBUG muestra los resultados en los registros. El contenido del registro IP es 0103H, que indica la ubicación del desplazamiento en el segmento de código de la siguiente instrucción que será ejecutada, a saber:

```
xxxx:0103 052500 ADD AX,0025
```

Para ejecutar esta instrucción, introduzca otra T. La instrucción ADD suma 25H a la mitad baja (AL) del registro AX y 00H a la mitad alta (AH), en realidad suma 0025H al AX. Ahora AX contiene 0148H y el IP contiene 016H para la siguiente instrucción que será ejecutada:

```
xxxx:0106 8BD8 MOV BX,AX
```

Teclee otro comando T. La instrucción MOV mueve el contenido del registro AX al registro BX. Note que después de mover BX contiene 0148H. AX aún contiene 0148H, ya que MOV *copia* en lugar de realmente mover los datos de una localidad a otra.

Ahora teclee de manera sucesiva comandos T para pasar por el resto de las instrucciones. La instrucción ADD suma el contenido de AX a BX, dando 0290H en BX. Después el programa mueve (copia) el contenido de BX a CX, resta AX de CX y resta AX de él mismo. Después de la última operación, la bandera de cero se cambia de NZ (no cero) a ZR (cero), para indicar que el resultado de la última operación fue cero (restar AX de él mismo lo deja en cero).

Si quiere volver a ejecutar estas instrucciones, inicie el registro IP con 100H y rastree otra vez. Introduzca R IP, introduzca 100 y después R y el número requerido de comandos T, todos seguidos por la tecla Enter.

### Cómo mostrar el contenido de memoria

Aunque también puede presionar T para la última instrucción, NOP (no operación), esta instrucción no realiza cosa alguna. En lugar de eso, para ver el programa en lenguaje de máquina en el segmento de código, requiere un despliegue como:

```
D CS:100
```

Ahora DEBUG muestra 16 bytes (32 dígitos hexadecimales) de datos en cada línea. A la derecha está la representación ASCII (si es imprimible) de cada byte (pareja de dígitos hexadecimales). En el caso de código de máquina, la representación ASCII carece de significado y puede ser ignorada. Secciones posteriores estudian con mayor detalle el lado correcto del despliegue.

La primera línea del despliegue inicia en el desplazamiento 100H del segmento de código y representa el contenido de las localidades CS:100 hasta CS:10F. La segunda línea representa el contenido de CS:110 hasta CS:11F. Aunque su programa termina en CS:110, el comando D en forma automática muestra ocho líneas desde CS:100 hasta CS:170.

La figura 3-2 muestra los resultados del comando D CS:100. Esperemos que el código de máquina desde CS:100 hasta 110 sea idéntico al que muestre su pantalla; los bytes que siguen pueden contener algo. También, la figura (3-1) muestra que los registros DS, ES, SS y CS todos contienen la misma dirección. Esto es porque DEBUG trata el área de programa como un segmento, con código y datos (si existen) en el mismo segmento, aunque usted debe mantenerlos separados.

Introduzca Q (Quit) para terminar la sesión con DEBUG, o continúe con el ejercicio siguiente.

```
-D CS:100
21C1:0100  B8 23 01 05 25 00 8B D8-03 D8 8B CB 2B C8 2B C0   .#. .%......+.+.
2101:0110  90 £3 8D 46 14 50 51 52-FF 76 28 E8 74 00 8B E5   ...F.PQR.v(.t
2101:0120  B8 01 00 50 FF 76 32 FF-76 30 FF 76 2E FF 76 28   ...P.v2.vO.v.'v(
21C1:0130  E8 88 15 8B E5 FF 36 18-12 FF 36 16 12 8B 76 28   ......6...6..•v(
21C1:0140  FF 74 3A 89 46 06 E8 22-CE 8B E5 30 E4 3D 0A 00   .t:.F0.
21C1:0150  75 32 Al 16 12 2D 01 00-8B 1E 18 12 83 DB 00 53   u2...-.........S
21C1:0160  50 8B 76 28 FF 74 3A A3-16 12 89 1E 18 12 E8 FA   P.v(.t:......
21C1:0170  CD 8B E5 30 E4 3D 0D 00-74 0A 83 06 16 12 01 83   ...0.-..t____
```

Figura 3-2 Vaciado del segmento de código

### Cómo corregir una entrada

Si usted introduce un valor erróneo en el segmento de datos o en el segmento de código, reintroduzca el comando E para corregirlo. También, reanude la ejecución en la primer instrucción iniciando el registro IP con 0100. Teclee el comando R seguido por el registro designado, esto es, R IP [Enter]. DEBUG muestra el contenido del IP y espera por una entrada. Teclee el valor 0100 (seguido por Enter). Después, teclee un comando R (sin el IP). DEBUG muestra los registros, banderas y la primera instrucción que será ejecutada. Usted ahora puede utilizar T para volver a rastrear las instrucciones paso a paso. Si su programa acumula totales, puede limpiar algunos registros y localidades de memoria; pero asegúrese de no cambiar el contenido de los registros CS, DS, SP y SS, todos ellos tienen propósitos específicos.

## EJEMPLO II DE LENGUAJE DE MAQUINA: DATOS DEFINIDOS

El ejemplo anterior usó valores inmediatos definidos directamente en las instrucciones MOV y ADD. Ahora ilustraremos un ejemplo parecido que define los valores de los datos (o constantes) 0123H y 025H como elementos separados dentro del programa. El programa es para accesar las localidades de memoria que contienen estos valores.

Al avanzar en este ejemplo debe hacerse una idea de cómo una computadora accesa los datos por medio de direcciones en el registro DS y direcciones de desplazamiento. El ejemplo define los siguientes elementos de datos y contenidos:

| DESPLAZAMIENTO DS | CONTENIDO HEXADECIMAL |
|---|---|
| 0200H | 2301H |
| 0202H | 2500H |
| 0204H | 0000H |
| 0206H | 2A2A2AH |

Recuerde que un dígito hexadecimal ocupa medio byte, así que, por ejemplo, 23H (el primer byte) es almacenado en el desplazamiento 0200H del área de datos y 01H (el segundo byte) es almacenado en el desplazamiento 0201H. A continuación están las instrucciones en lenguaje de máquina que procesan estos datos:

| INSTRUCCIÓN | EXPLICACIÓN |
|---|---|
| A10002 | Mover la palabra (dos bytes) que inicia en el DS con desplazamiento 0200H al registro AX. |
| 03060202 | Sumar el contenido de la palabra (dos bytes) que inicia en el DS con desplazamiento 0202H al registro AX. |
| A30402 | Mover el contenido del registro AX a la palabra que inicia en el DS con desplazamiento 0204H. |
| 90 | No operación. |

Puede haber notado que las dos instrucciones para mover tienen diferentes códigos de máquina: Al y A3. El código real de máquina es dependiente de los registros a los que esté referenciando,

**el** tamaño de los datos, la dirección de transferencia de datos (de o hacia un registro) y de la referencia a datos inmediatos o en memoria.

### Cómo introducir instrucciones de programa

Otra **vez,** puede utilizar DEBUG para introducir el programa y observar su ejecución. Primero, utilice **los** comandos E (Enter) para definir los datos, iniciando en DS:0200:

```
E  DS:0200  23  01  25  00  00  00  [presione Enter]

E  DS:0206  2A  2A  2A  [presione Enter]
```

Ahora utilice el comando E para teclear las instrucciones, otra vez iniciando en CS: 100:

```
E  CS:100  Al  00  02  03  06  02  02  [presione Enter]

E  CS:107  A3  A4  02  90  [presione Enter]
```

El primes comando E almacena las tres palabras (seis bytes) en el inicio del área de datos, DS:0200. Note que tiene que introducir estas palabras con los bytes en orden inverso, de manera que 0123 es 2301 y 0025 es 2500. Cuando la instrucción MOV accesa de manera secuencial estas palabras y las carga en un registro, "deshace la inversión", es decir, vuelve a invertir el orden de los bytes, de modo que 2301 se convierte en 0123 y 2500 en 0025.

El segundo comando E almacena tres asteriscos (***), definidos como 2A2A2A, de modo que usted pueda verlos más tarde utilizando el comando D (Display, mostrar). De lo contrario, estos asteriscos no sirven para algún propósito particular en el segmento de datos.

La figura 3-3 muestra todos los pasos en el programa, incluyendo los comandos E. Su pantalla debe mostrar resultados parecidos, aunque las direcciones en el CS y DS tal vez puedan diferir. Para examinar los datos almacenados (en DS:200H a 208H) y las instrucciones (en CS: 100H a 10AH), teclee los siguientes comandos D:

```
Para ver los datos:    D  DS:200,208  [presione Enter]

Para ver el código:    D  CS:100,10A  [presione Enter]
```

Verifique que los contenidos de ambas áreas (distintas a las direcciones de segmento) sean idénticas a las que se muestran en la figura 3-3.

### Cómo ejecutar instrucciones de programa

Puede ejecutar las instrucciones mostradas en la forma que ya se dijo. Presione R para ver el contenido de los registros y de las banderas y para mostrar la primera instrucción. Los registros contienen los mismos valores que al inicio del primer ejemplo. La primera instrucción mostrada es:

```
xxxx:0100 A10002 MOV AX,  [0200]
```

)

**CS**:0100 hace referencia a su primera instrucción, A10002. DEBUG interpreta esta instrucción como un MOV y determina que la referencia es a la primera localidad [0200H] en el área de datos. Los corchetes son para indicarle que esta referencia es a una dirección de memoria y no es

```
-E DS:200 23 01 25 00 00 00
-E DS:206 2A 2A 2A
-E CS:100 Al 00 02 03 06 02 02
-E CS:107 A3 04 02 90
-D DS:200,208
21C1:0200  23 01 25 00 00 00 2A 2A-2A                        #.%...***
-D CS:100,10A
21C1:0100  Al 00 02 03 06 02 02 A3-04 02 90
-R
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=21C1  ES=21C1  SS=21C1  CS=21C1  IP=0100   NV UP El PL NZ NA PO NC
21C1:0100 A10002        MOV     AX,[0200]                    DS:0200=0123
-T

AX=0123  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=21C1  ES=21C1  SS=21C1  CS=21C1  IP=0103   NV UP El PL NZ NA PO NC
21C1:0103 03060202      ADD     AX,[0202]                    DS:0202=0025
-T

AX=0148  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=21C1  ES=21C1  SS=21C1  CS=21C1  IP=0107   NV UP El PL NZ NA PE NC
21C1:0107 A30402        MOV     [0204],AX                    DS:0204=0000
-T

AX=0148  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=21C1  ES=21C1  SS=21C1  CS=21C1  IP=010A   NV UP El PL NZ NA PE NC
21C1:010A 90            NOP
-D DS:0200,0208
21C1:0000  23 01 25 00 48 01 2A 2A-2A                        #.%.H.***
-Q
```

**Figura** 3-3 Rastreo de las instrucciones de máquina

un valor inmediato. (Un valor inmediato para mover 0200H al registro AX aparecería como MOV AX,0200.)

Ahora teclee el comando T (trace, rastrear). La instrucción MOV AX,[0200] mueve el contenido de la palabra en el desplazamiento 0200H al registro AX. El contenido es 2301H, el cual aparece en orden inverso en el AX como 0123H.

Ingrese otro comando T para provocar la ejecución de la siguiente instrucción, ADD. La operación suma el contenido de la palabra de memoria en DS con desplazamiento 0202 al registro AX. El resultado en el AX ahora es la suma de 0123H y 0025H, o 0148H.

La siguiente instrucción es MOV [0204],AX. Teclee un comando T para ejecutarla. La instrucción mueve el contenido del registro AX a la palabra de memoria en DS con desplazamiento 0204H. Para ver los contenidos cambiados de los datos desde 200H hasta 208H, teclee

```
D DS:200,208 [Enter]
```

Los valores mostrados deben ser:

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **Valor en el área de datos:** | 23 | 01 | 25 | 00 | 48 | 01 | 2A | 2A | 2A |
| **Desplazamiento:** | 200 | 201 | 202 | 203 | 204 | 205 | 206 | 207 | 208 |

El valor 0148H es movido del registro AX al área de datos con desplazamientos 204H y 205H y es invertido como 4801H. El lado izquierdo de la pantalla muestra los códigos reales de máquina

como aparece en memoria. El lado derecho sólo ayuda a localizar los caracteres de manera más sencilla. Advierta que estos valores hexadecimales son representados a la derecha de la pantalla por sus equivalentes ASCII. Así 23H genera un símbolo de número (#) y 25H genera un símbolo de por ciento (%) mientras que los tres bytes con 2AH generan asteriscos (*).

Ya que no hay más instrucciones por ejecutar, introduzca Q (quit) para terminar la sesión con DEBUG o continúe con el siguiente ejercicio (y recuerde restablecer el IP a 100).

## CÓMO INTRODUCIR UN PROGRAMA SIMBÓLICO EN ENSAMBLADOR

Aunque en este momento los ejemplos de programas han sido en formato de lenguaje de máquina, también puede utilizar DEBUG para ingresar instrucciones simbólicas en lenguaje ensamblador. Puede encontrar ocasiones para usar ambos métodos. Ahora examinemos cómo introducir enunciados en lenguaje ensamblador.

### El comando A

El comando A (Assemble, ensamblar) le dice a DEBUG que acepte instrucciones simbólicas en ensamblador y las convierta a lenguaje de máquina. Inicialice la dirección de inicio en el segmento de código con desplazamiento 100H para sus instrucciones como

```
A 100  [Enter]
```

DEBUG muestra el valor del segmento de código y el desplazamiento como xxxx:0100. Teclee cada instrucción, seguida por Enter. Intente ingresar el programa siguiente:

```
MOV   AL,25   [Enter]
MOV   BL,32   [Enter]
ADD   AL,BL   [Enter]
NOP           [Enter,
```

Cuando haya tecleado el programa, presione otra vez Enter para salirse del comando A. Esto es, un Enter extra, que le indica a DEBUG que ya no tiene más instrucciones simbólicas por ingresar. Al terminar, DEBUG debe mostrar lo siguiente:

```
xxxx:0100        MOV    AL,25
xxxx:0102        MOV    BL,32
xxxx:0104        ADD    AL,BL
xxxx:0106        NOP
```

Puede ver que DEBUG ha determinado la localidad de inicio de cada instrucción. Pero antes de ejecutar el programa, usemos el comando U (Unassemble, desensamblar) de DEBUG para examinar el lenguaje de máquina generado.

### El comando U (Unassemble, desensamblar)

El comando U de DEBUG muestra el código de máquina para sus instrucciones en lenguaje ensamblador. Puede usar este comando para indicarle a DEBUG las localidades de la primera y última instrucciones que quiere ver, en este caso, 100H y 106H. Teclee

```
U 100,106  [Enter]
```

La pantalla debe mostrar columnas para la localidad, el código de máquina y el código simbólico:

```
xxxx:0100  B025      MOV  AL,25
xxxx:0102  B332      MOV  BL,32
xxxx:0104  00D8      ADD  AL,BL
xxxx:0106  90        NOP
```

Ahora rastree la ejecución del programa; el código de máquina es lo que en realidad se ejecuta. Empiece por introducir R para desplegar los registros y la primer instrucción, y después T de manera sucesiva para rastrear las instrucciones subsecuentes. Cuando llegue a NOP de la localidad 106H, continúe con el ejercicio siguiente o presione Q para salir de la ejecución.

Ahora puede ver cómo ingresar un programa en cualquiera de los dos, lenguaje de máquina o lenguaje ensamblador. Sin embargo, DEBUG está proyectado para lo que su nombre implica —depurar (debug) programas— y la mayoría de los esfuerzos que involucrarán el uso de lenguaje ensamblador convencional no están asociados con DEBUG.

## USO DE LA INSTRUCCIÓN INT

Los tres ejemplos siguientes muestran cómo accesar el DOS y el BIOS para enviar información acerca del sistema. Para este fin, se utiliza la instrucción INT (interrupción), la cual sale de su programa, ingresa una rutina del DOS o de BIOS, realiza la función solicitada y regresa a su programa. En lugar de avanzar un solo paso, usaremos el comando P (Proceed) de DEBUG para ejecutar toda la rutina de interrupción.

### Cómo obtener el número de versión del DOS

Existen ocasiones en que un programa necesita saber cuál es la versión del DOS que la computadora está corriendo, ya que cada versión tiene disponibles nuevas funciones. La instrucción que entrega el número de versión es INT 21H del DOS, función 30H; esto es, cargue 30H en el registro AH y solicite INT 21H.'Para probar esto, ingrese el comando A 100 de DEBUG y estas instrucciones en ensamblador:

```
MOV   AH,30
INT   21
NOP    (seguido por un Enter adicional)
```

Para rastrear la ejecución de las instrucciones, primero ingrese R para ver los registros y T para rastrear MOV. En lugar de rastrear la instrucción INT, ingrese P (Proceed, proceder) para ejecutar toda la rutina del DOS. El proceso termina con la instrucción NOP. Ahora puede ver en el AL el número principal de la versión del DOS, como X en DOS X.20, y en el AH el número secundario de la versión, como 14H (o 20) en DOS X.20.

Presione Q para salir o continúe con el ejercicio siguiente (y restablezca el IP a 100).

**Cómo obtener la fecha actual**

Ahora que ya sabe cómo accesar el número de versión del DOS, puede utilizar un enfoque semejante para accesar la fecha actual. La instrucción para este propósito es INT 21H del DOS, función 2AH. Una vez más, ingrese el comando A 100 de DEBUG y después el programa siguiente de ensamblador:

```
MOV   AH.2A
INT   21
NOP
```

Ingrese R para mostrar los registros y R para ejecutar MOV. Después ingrese P para proceder con la rutina de interrupción; la operación se detiene en la instrucción NOP. Los registros muestran esta información:

- AL: Día de la semana (donde 0 = Domingo)
- CX: Año (en hexadecimal; por ejemplo, 07CDH = 1997)
- DH: Mes (01 a 12)
- DL: Día del mes (01 a 31)

- Presione Q para salir o continúe con el ejercicio siguiente.

**Cómo determinar el tamaño de la memoria**

En un ejercicio anterior, verificó las localidades 413H y 414H para saber la cantidad de memoria (RAM) que tiene su computadora. El BIOS también proporciona una rutina de interrupción, INT 12H, que entrega el tamaño de la memoria. Ingrese el comando A 100 de DEBUG y después estas instrucciones:

```
INT   12
NOP
```

Ingrese R para mostrar los registros y la primera instrucción. La instrucción, INT 12H, transfiere el control a una rutina en el BIOS que entrega el tamaño de la memoria al AX. Presione T (y Enter) de forma repetida para ver cada instrucción del BIOS que se ejecuta (sí, estamos violando una regla contra el rastreo de una interrupción, pero esta vez todo funciona bien).

Las instrucciones reales en su BIOS pueden diferir de éstas, dependiendo de la versión instalada (los comentarios a la derecha son del autor):

```
STI                 /Establece la interrupción
PUSH    DS          ;Guarda la dirección del DS en la
MOV     AX,0040     ;Segmento 40[0]H
MOV     DS,AX       ; más
MOV     AX,[0013]     desplazamiento 0013H
POP     DS          ,•Restaura la dirección en el DS
IRET                ,•Regresa de la interrupción
```

Si sobrevive a esta aventura con el BIOS, el AX contiene el tamaño de la memoria, en 1K bytes. El último comando T sale del BIOS y regresa a DEBUG. La instrucción mostrada es el NOP que usted ingresó. Presione Q para salir o continúe con el ejercicio siguiente (y restablezca el IP a 100).

## CÓMO GUARDAR UN PROGRAMA DESDE DEBUG

Usted puede utilizar DEBUG para guardar un programa en disco bajo dos circunstancias:

**1.** Para leer el programa, modificarlo y después guardarlo, siga estos pasos:
- Lea el programa bajo su nombre: DEBUG mnombredearchivo.
- Utilice el comando D para ver el programa en lenguaje de máquina y use E para ingresar los cambios.
- Utilice el comando W (Write, escribir) para grabar el programa revisado.

2. Usar DEBUG para escribir un pequeño programa en lenguaje de máquina que ahora quiera guardar; siga estos pasos:
- Solicite el programa DEBUG.
- Utilice A (ensamblar) y E (ingresar) para crear el programa.
- Ponga nombre al programa: N nombredearchivo.COM. La extensión del programa debe ser .COM. (Véase el capítulo 7 para detalles de los archivos .COM.)
- Ya que sólo usted sabe dónde termina en realidad el programa, indique a DEBUG el tamaño del programa en bytes. Examine este ejemplo:

```
xxxx:0100  MOV AL,25
xxxx:0102  MOV BL,32
xxxx:0104  ADD AL,BL
xxxx:0106  NOP
```

Puesto que la última instrucción, NOP, es de un byte, el tamaño del programa es de 7 bytes, desde 100H hasta 106H, inclusive.
- Primero utilice R BX para mostrar el BX, e ingrese 0 para limpiarlo.
- Ahora use R CX para mostrar el registro CX. DEBUG responde con CX 0000 (valor cero) y usted contesta con el tamaño del programa, 7.
- Grabe el programa revisado: W [Enter].

La razón para limpiar el BX es porque la longitud del programa está en la pareja BX:CX, aunque el CX es adecuado para nuestros propósitos.

DEBUG muestra un mensaje "Writing nnnn bytes" (Se escribieron nnnn bytes). Si el número es cero, se ha equivocado al introducir la longitud del programa; inténtelo otra vez. Tenga cuidado en el tamaño del programa, ya que la última instrucción puede ser mayor de un byte.

## EJEMPLO DE LENGUAJE ENSAMBLADOR: EL OPERADOR PTR

Ahora examinemos otro programa que introduce algunas características nuevas. En este ejemplo, movemos y sumamos datos entre las localidades de memoria y los registros. Aquí están las instrucciones para este propósito:

```
100    MOV    AX, [HA]
103    ADD    AX,[11C]
107    ADD    AX,25
10A    MOV     [HE] ,AX
10D    MOV    WORD PTR [120],25
113    MOV    BYTE PTR [122],30
118    NOP
119    NOP
HA     DB     14 23
11C    DB     05 00
HE     DB     00 00
120    DB     00 00 00
```

Una explicación de las instrucciones es la siguiente:

100: Mueve el contenido de las localidades de memoria 11AH-11BH al AX. Los corchetes indican una dirección de memoria y no valores inmediatos.

103: Suma los contenidos de las localidades de memoria 11CH-11DH al AX.

107: Suma el valor inmediato 25H al AX.

10A: Mueve el contenido de AX a las localidades de memoria 11EH-11FH.

10D: Mueve el valor inmediato 25H a las localidades de memoria 120H-121H. Note el uso del operador WORD PTR, que indica a DEBUG que debe mover el 25H a una *palabra* de memoria. Si estuviera codificada la instrucción como MOV [120],25, DEBUG no tendría manera de determinar la longitud que se pretende y mostraría un mensaje de ERROR. Aunque rara vez necesita usar el operador PTR, es vital saber cuando se necesita.

113: Mueve el valor inmediato 30H a la localidad de memoria 122H. Esta vez, queremos mover un byte, y el operador BYTE PTR indica esta longitud.

11A: Define los valores de byte 14H y 23H. DB significa "definir byte(s)" y le permite definir datos que sus instrucciones (como la que está en 100) están referenciando

11C, 11E y 120: Definen otros valores de byte para uso en el programa.

Para ejecutar este programa, primero teclee A 100 [Enter], y después teclee cada instrucción simbólica (pero no la localidad). Al terminar, teclee un Enter adicional para salir del comando A. Empiece por introducir R para mostrar los registros y la primera instrucción; después ingrese de manera sucesiva comandos T. Salga de la ejecución cuando llegue a NOP en 118. Teclee D 110 para mostrar los contenidos cambiados de AX (233E) y de las localidades 11EH-11FH (3E23), 120H-121H (2500) y 122H (30).

## PUNTOS CLAVE

- El programa DEBUG es útil para probar y depurar programas escritos en lenguaje de máquina y en lenguaje ensamblador.
- DEBUG proporciona un conjunto de comandos que permiten realizar diferentes operaciones útiles, como desplegar, introducir y rastrear.
- Como DEBUG no distingue entre letras minúsculas y mayúsculas, puede introducir los comandos de cualquier forma.
- DEBUG supone que todos los números están en formato hexadecimal.
- Si usted introduce un valor incorrecto en el segmento de datos o en el segmento de código, vuelva a introducir el comando E para corregirlo.
- Para reasumir la ejecución en la primera instrucción, asigne al registro de apuntador de instrucción (IP) un 0100. Teclee el comando R (registro), seguido por el registro designado, como R IP [Enter], DEBUG muestra el contenido de IP y espera otra entrada. Teclee el valor 0100 (seguido por Enter).

## PREGUNTAS

3-1. ¿Cuál es el propósito de cada uno de los siguientes comandos de DEBUG? (a) A; (b) D; (c) E; (d) P; (e) Q; (f) R; (g) T; (h) U.

3-2. Proporcione los comandos de DEBUG para las siguientes necesidades.
   (a) Muestre la memoria iniciando en el desplazamiento 264H en el segmento de datos.
   (b) Muestre la memoria iniciando en la localidad 410H. (Nota: Separe esta dirección en los valores de su segmento y del desplazamiento.)
   (c) Ingrese el valor hexadecimal A8B364 en el segmento de datos iniciando en la localidad 200H.
   (d) Muestre el contenido de *(i)* todos los registros y *(ii)* sólo del registro IP.
   (e) Desensamble el código de máquina que se encuentra en las localidades desde la 100H hasta 11EH.

3-3. Proporcione instrucciones en código de máquina para las siguientes operaciones: (a) Mover el valor 4629 hexadecimal al registro AX; (b) sumar el valor hexadecimal 036A al registro AX.

3-4. Suponga que ha utilizado DEBUG para introducir el comando siguiente:

```
E CS:100 B8 45 01 05 25 00
```

El valor 45 hexadecimal supuestamente era 54. Codifique otro comando E para corregir sólo el byte que está incorrecto; esto es, cambie el 45 por el 54 de forma directa.

3-5. Suponga que ha utilizado DEBUG para introducir el comando E siguiente:

```
E CS:100 B8 04 30 05 00 30 90
```

(a) ¿Cuáles son las tres instrucciones simbólicas representadas aquí? (El primer programa en este capítulo da una pista.)

**(b)** Al ejecutar este programa, usted descubre que el registro AX termina con 6004 en lugar del esperado 0460. ¿Cuál es el error y cómo lo corregiría?

(c) Habiendo corregido las instrucciones, usted ahora vuelve a ejecutar el programa desde la primera instrucción. ¿Cuáles son los dos comandos de DEBUG que se requieren?

3-6. Considere el programa en lenguaje de máquina

```
B0 25 DO E0 B3 15 F6 E3 90
```

Este programa realiza lo siguiente:

- Mueve el valor 25 hexadecimal al registro AL.
- Recorre el contenido de AL un bit a la izquierda. (El resultado es 4A.)
- Mueve el valor 15 hexadecimal al registro BL.
- Multiplica el AL por el BL.

Utiliza el comando E de DEBUG para introducir el programa, iniciando en CS: 100. Recuerde que estos son valores hexadecimales. Después de introducir el programa, teclee D CS:100 para verlo. Después teclee R y suficientes comandos T, de manera sucesiva para avanzar por el programa hasta que alcance NOP. ¿Cuál es el resultado final en el registro AX?

3-7. Utilice el comando E de DEBUG para introducir el siguiente programa en lenguaje de máquina:

```
Código de máquina (en 100H): 00 02 DO E0 F6 26 01 02 A3 02 02 90

        Datos (en 200H): 25 15 00 00
```

Este programa realiza lo siguiente:

- Mueve el contenido de un byte en DS:0200 (25) al registro AL.
- Recorre el contenido de AL un bit a la izquierda. (El resultado es 4A.)
- Multiplica el AL por un byte contenido en DS:0201 (15).
- Mueve el producto de AX a la palabra que inicia en DS:0202.

Después de introducir el programa, teclee los comandos D para ver el código y los datos. Después teclee R y suficientes comandos T, de manera sucesiva, para avanzar por el programa hasta que llegue a NOP. Al llegar a este punto, el AX debe contener el producto en memoria en 0612H. Teclee otro D DS:0200 y note que el producto en DS:0202 es almacenado como 1206H.

3-8. Para la pregunta 3-7, codifique los comandos que graben el programa en disco con el nombre TRIAL.COM.

3-9. Utilice el comando A de DEBUG para introducir las siguientes instrucciones:

```
MOV   BX,25

ADD   BX,30
```

```
SHL   BX,01
SUB   BX,22
NOP
```

Desensamble las instrucciones y rastree su ejecución hasta NOP y revise el valor en el BX después de cada instrucción.

3-10. ¿Cuál es el propósito de la instrucción INT?

## PARTE B — Fundamentos de lenguaje ensamblador

## CAPÍTULO 4

# Requerimientos de lenguaje ensamblador

OBJETIVO

Cubrir los requerimientos básicos para codificar un programa en lenguaje ensamblador y definir los elementos de datos.

### INTRODUCCIÓN

El capítulo 3 mostró cómo usar DEBUG para teclear y ejecutar programas en lenguaje de máquina. Sin duda usted fue muy consciente de la dificultad de descifrar el código de máquina, aun para un programa pequeño. Probablemente ningún programa se codifica más en serio en lenguaje de máquina que los programas más pequeños. Un nivel más alto de codificación es el nivel ensamblador, en el que un programador utiliza instrucciones simbólicas en lugar de instrucciones de máquina y nombres descriptivos para los elementos de datos y para las localidades de memoria. Usted escribe un programa en ensamblador de acuerdo con un conjunto estricto de reglas que después utiliza el programa traductor de ensamblador para convertir el programa en ensamblador en código de máquina.

En este capítulo explicamos los requisitos básicos para desarrollar un programa en ensamblador: el uso de comentarios, el formato general de codificación, las directivas de impresión del listado de un programa y las directivas para definir segmentos y procedimientos. También cubrimos la organización general de un programa, incluyendo la inicialización y la terminación de su ejecución. Por último, tratamos los requisitos para definir elementos de datos.

## ENSAMBLADORES Y COMPILADORES

Primero identificamos dos clases de lenguajes de programación: de *alto nivel* y de *bajo nivel.* Los programadores que escriben en un lenguaje de alto nivel, como C y Pascal, codifican comandos poderosos, cada uno de los cuales puede generar muchas instrucciones en lenguaje de máquina. Por otro lado, los programadores que escriben en un lenguaje ensamblador de bajo nivel codifican instrucciones simbólicas, cada una de las cuales genera una instrucción en lenguaje de máquina. A pesar del hecho de que codificar en un lenguaje de alto nivel es más productivo, algunas ventajas de codificar en lenguaje ensamblador son:

- Proporciona más control sobre el manejo particular de los requerimientos de hardware.
- Genera módulos ejecutables más pequeños y más compactos.
- Con mayor probabilidad tiene una ejecución más rápida.

Una práctica común es combinar los beneficios de ambos niveles de programación: codificar el grueso de un proyecto en un lenguaje de alto nivel y los módulos críticos (aquellos que provocan notables retardos) en lenguaje ensamblador.

Sin importar el lenguaje de programación que utilice, de todos modos es un lenguaje simbólico que tiene que traducirse a una forma que la computadora pueda ejecutar. Un lenguaje de alto nivel utiliza un *compilador* para traducir el código fuente a lenguaje de máquina (técnicamente, código objeto). Un lenguaje de bajo nivel utiliza un *ensamblador* para realizar la traducción. Un programa *enlazador* para ambos niveles, alto y bajo, completa el proceso al convertir el código objeto en lenguaje ejecutable de máquina.

## COMENTARIOS EN LENGUAJE ENSAMBLADOR

El uso de comentarios a lo largo de un programa puede mejorar su claridad, en especial en lenguaje ensamblador, donde el propósito de un conjunto de instrucciones con frecuencia no es claro. Un comentario empieza con punto y coma (;) y, en donde quiera que lo codifique, el ensamblador supone que todos los caracteres a la derecha en esa línea son comentarios.Un comentario puede contener cualquier carácter imprimible, incluyendo el espacio en blanco.

Un comentario puede aparecer sólo en una línea o a continuación de una instrucción en la misma línea, como lo muestran los dos ejemplos siguientes:

```
1.  ;Toda esta línea es un comentario

2.  ADD AX,BX  /Comentario en la misma línea que la instrucción
```

Ya que un comentario aparece sólo en un listado de un programa fuente en ensamblador y no genera código de máquina, puede incluir cualquier cantidad de comentarios sin afectar el tamaño o la ejecución del programa ensamblado. En este libro, las instrucciones ensambladas están en letras mayúsculas y los comentarios en letras minúsculas, sólo como convención y para hacer que los programas sean más legibles. Técnicamente, usted está en libertad de usar letras mayúsculas o minúsculas para las instrucciones y comentarios.

Otra manera de proporcionar comentarios es por medio de la directiva COMMENT, que se estudia en el capítulo 27.

## PALABRAS RESERVADAS

Ciertas palabras en lenguaje ensamblador están *reservadas* para sus propósitos propios, y son usadas sólo bajo condiciones especiales. Por categorías, las palabras reservadas incluyen

- instrucciones, como MOV y ADD, que son operaciones que la computadora puede ejecutar;
- directivas, como END o SEGMENT, que se emplean para proporcionar comandos al ensamblador;
- operadores, como FAR y SIZE, que se utilizan en expresiones; y
- símbolos predefinidos, como ©Data y @Model, que regresan información a su programa.

El uso de una palabra reservada para un propósito equivocado provoca que el ensamblador genere un mensaje de error. El apéndice C muestra una lista de las palabras reservadas del lenguaje ensamblador.

## IDENTIFIC ADORES

Un *identificador* es un nombre que se aplica a elementos en el programa. Los dos tipos de identificadores son: *nombre,* que se refiere a la dirección de un elemento de dato, y *etiqueta,* que se refiere a la dirección de una instrucción. Las mismas reglas se aplican tanto para los, nombres como para las etiquetas. Un identificador puede utilizar los siguientes caracteres:

| | |
|---|---|
| • Letras del alfabeto: | desde la A hasta la Z |
| • Dígitos: | desde el 0 hasta 9 (no puede ser el primer carácter) |
| • Caracteres especiales | signo de interrogación (?) |
| | subrayado (_) |
| | signo de pesos ($) |
| | arroba (@) |
| | punto (.) (no puede ser el primer carácter) |

El primer carácter de un identificador debe ser una letra o un carácter especial, excepto el punto. Ya que el ensamblador utiliza algunos símbolos especiales en palabras que inician con el símbolo @, debe evitar usarlo en sus definiciones.

El ensamblador trata las letras mayúsculas y minúsculas como iguales. La longitud máxima de un identificador es de 31 caracteres (247 desde el MASM 6.0). Ejemplos de nombres válidos son COUNT, PAGE25 y $E10. Se recomienda que los nombres sean descriptivos y con significado. Los nombres de registros, como AX, DI y AL, están reservados para hacer referencia a esos mismos registros. En consecuencia, en una instrucción tal como:

```
ADD AX,BX
```

el ensamblador sabe de forma automática que AX y BX se refieren a los registros. Sin embargo, en una instrucción como:

```
MOV  REGSAVE,AX
```

el ensamblador puede reconocer el nombre REGSAVE sólo si se define en algún lugar del programa.

## INSTRUCCIONES

Un programa en lenguaje ensamblador consiste en un conjunto de *enunciados.* Los dos tipos de enunciados son:

1. instrucciones, tal como MOV y ADD, que el ensamblador traduce a código objeto; y
2. directivas, que indican al ensamblador que realice una acción específica, como definir un elemento de dato.

A continuación está el formato general de un enunciado, en donde los corchetes indican una entrada opcional:

```
[identificador]    operación    [operando(s)]    [;comentario]
```

Un identificador (si existe), una operación y un operando (si existe) están separados por al menos un espacio en blanco o un carácter de tabulador. Existe un máximo de 132 caracteres en una línea (512 desde el MASM 6.0), aunque la mayoría de los programadores prefiere permanecer en los 80 caracteres ya que es el número máximo que cabe en la pantalla. A continuación se presentan dos ejemplos de enunciados:

```
IDENTIFICADOR      OPERACIÓN    OPERANDO        COMENTARIO
Directiva: COUNT       DB           1       /Nombre, operación, operando

Instrucción:           MOV          AX, 0   ,-Operación, dos operandos
```

Identificador, operación y operando pueden empezar en cualquier columna. Sin embargo, si de manera consistente se inicia en la misma columna para estas entradas se hace un programa más legible. También, la mayoría de los programas editores proporcionan marcas de tabulador cada ocho posiciones para facilitar el espaciamiento.

### Identificador

Como ya se explicó, el término *nombre* se aplica al nombre de un elemento o directiva definida, mientras que el término *etiqueta* se aplica al nombre de una instrucción; usaremos estos términos de ahora en adelante.

### Operación

La *operación,* que debe ser codificada, es con mayor frecuencia usada para la definición de áreas de datos y codificación de instrucciones. Para un elemento de datos, una operación tal como DB o DW define un campo, área de trabajo o constante. Para una instrucción, una operación como MOV o ADD indica una acción a realizar.

### Operando

El *operando* (si existe) proporciona información para la operación que actúa sobre él. Para un elemento de datos, el operando identifica su valor inicial. Por ejemplo, en la definición siguiente de un elemento de datos llamado COUNTER, la operación DB significa "definir byte", y el operando inicializa su contenido con un valor cero:

```
NOMBRE    OPERACIÓN   OPERANDO          COMENTARIO
COUNTER       DB          0 .     ,-Define un byte (DB) con el valor cero
```

Para una instrucción, un operando indica en dónde realizar la acción. Un operando de una instrucción puede tener una, dos o tal vez ninguna entrada. Aquí están tres ejemplos:

```
OPERACIÓN    OPERANDO          COMENTARIO                 OPERANDO
   RET                    /Regresa                      Ninguno
   INC          CX        /Incrementa el registro CX    Uno
   ADD          AX.12     /Suma 12 al registro AX       Dos
```

## DIRECTIVAS

El lenguaje ensamblador permite usar diferentes enunciados que permiten controlar la manera en que un programa ensambla y lista. Estos enunciados, llamados *directivas,* actúan sólo durante el ensamblado de un programa y no generan código ejecutable de máquina. Las directivas más comunes son explicadas en las siguientes secciones. El capítulo 27 trata con detalle todas las directivas; en cualquier momento usted puede utilizar ese capítulo como referencia.

### Directivas para listar: PAGE y TITLE

Las directivas PAGE y TITLE ayudan a controlar el formato de un listado de un programa en ensamblador. Éste es su único fin, y no tienen efecto sobre la ejecución subsecuente del programa.

PAGE. Al inicio de un programa, la directiva PAGE designa el número máximo de líneas para listar en una página y el número máximo de caracteres en una línea. Su formato general es

```
PAGE [longitud] [,ancho]
```

El ejemplo siguiente proporciona 60 líneas por página y 132 caracteres por línea:

```
PAGE 50,132
```

El número de líneas por página puede variar desde 10 hasta 255, mientras que el número de caracteres por línea desde 60 hasta 132. La omisión de un enunciado PAGE causa que el ensamblador tome PAGE 50,80.

Suponga que el número de líneas para PAGE se definió como 60. Entonces, cuando el programa ensamblado haya listado 60 líneas avanza las formas al inicio de la siguiente página e incrementa en uno el contador de páginas. También puede usted querer forzar un salto de página

en una línea específica en el listado del programa, como al final de un segmento. En la línea requerida, sólo codifique PAGE sin operandos. Al encontrar PAGE el ensamblador salta la página de manera automática y reasume la impresión en la parte superior (al inicio) de la siguiente página.

**TITLE.** Se puede emplear la directiva TITLE para hacer que un título para un programa se imprima en la línea 2 de cada página en el listado del programa. Puede codificar TITLE de una vez, al inicio del programa. Su formato general es

```
TITLE texto
```

Para el operando texto, una técnica recomendada es utilizar el nombre del programa como se registra en el disco. Por ejemplo, si a su programa le puso por nombre ASMSORT, codifique el nombre más un comentario descriptivo opcional, hasta 60 caracteres, como esto:

```
TITLE ASMSORT Programa en ensamblador para ordenar los nombres de los clientes
```

### Directiva SEGMENT

Un programa ensamblado en formato .EXE consiste en uno o más segmentos. Un segmento de pila define el almacén de la pila, un segmento de datos define los elementos de datos y un segmento de código proporciona un código ejecutable. Las directivas para definir un segmento, SEGMENT y ENDS, tienen el formato siguiente:

```
NOMBRE      OPERACIÓN   OPERANDO        COMENTARIO
nombre        SEGMENT   [opciones]    ;Inicia el segmento


nombre        ENDS                      ;Fin del segmento
```

El enunciado SEGMENT define el inicio de un segmento. El nombre del segmento debe estar presente, ser único y cumplir las convenciones para nombres del lenguaje. El enunciado ENDS indica el final del segmento y contiene el mismo nombre del enunciado SEGMENT. El tamaño máximo de un segmento es 64K. El operando de un enunciado SEGMENT puede tener tres tipos de opciones: alineación, combinar y clase, codificadas en este formato:

```
nombre  SEGMENT  alineación  combinar  'clase'
```

**Tipo alineación.** La entrada alineación indica el límite en el que inicia el segmento. Para el requerimiento típico, PARA, alinea el segmento con el límite de un párrafo, de manera que la dirección inicial es divisible entre 16, o 10H. En ausencia de un operando hace que el ensamblador por omisión tome PARA.

**Tipo combinar.** La entrada combinar indica si se combina el segmento con otros segmentos cuando son *enlazados* después de ensamblar (se explica posteriormente en "Cómo enlazar el programa"). Los tipos combinar son STACK, COMMON, PUBLIC y la expresión AT. Por ejemplo, el segmento de la pila por lo común es definido como

```
nombre  SEGMENT  PARA  STACK
```

Puede utilizar PUBLIC y COMMON en donde tenga el propósito de combinar de forma separada programas ensamblados cuando los enlaza. En otros casos, donde un programa no es comoinado con otros, puede omitir la opción o codificar NONE.

**Tipo clase.** La entrada clase, encerrada entre apóstrofos, es utilizada para agrupar segmentos cuando se enlazan. Este libro utiliza la clase 'code' para el segmento de códigos (recomendado por Microsoft), 'data' por segmento de datos y 'stack' para el segmento de la pila.

El ejemplo siguiente define un segmento de pila con tipos alineación, combinar y clase:

```
nombre  SEGMENT  PARA  STACK <Stack'
```

La parte del programa en la figura 4-1 ilustra enunciados SEGMENT con varias opciones.

### **Directiva** PROC

El segmento de código contiene el código ejecutable de un programa. También tiene uno o más *procedimientos,* definidos con la directiva PROC. Un segmento que tiene sólo un procedimiento puede aparecer como sigue:

```
NOMBRE      OPERACIÓN     OPERANDO     COMENTARIO
nomsegmto   SEGMENT       PARA
nomproc     PROC          FAR          Un
                                       procedimiento
                                       dentro
                                       del segmento
nomproc     ENDP                       de código
nomsegmto   ENDS
```

El nombre del procedimiento debe estar presente, ser único y seguir las reglas para la formación de nombres del lenguaje. El operando FAR en este caso está relacionado con la ejecución del programa. Cuando usted solicita la ejecución de un programa, el cargador de programas del DOS utiliza este nombre de procedimiento como el punto de entrada para la primera instrucción a ejecutar.

La directiva ENDP indica el fin de un procedimiento y contiene el mismo nombre que el enunciado PROC para permitir que el ensamblador relacione a los dos. Ya que los procedimientos deben estar por completo dentro de un segmento, ENDP define el final de un procedimiento antes que ENDS defina el final de un segmento.

El segmento de código puede contener cualquier número de procedimientos usados como subrutinas, cada uno de los cuales va con su característico conjunto de enunciados PROC y ENDP. Cada PROC adicional por lo común se codifica con (o por omisión) el operando NEAR; el capítulo 7 analiza esta situación.

### **Directiva** ASSUME

Un programa utiliza al registro SS para direccionar la pila, al registro DS para direccionar el segmento de datos y al registro CS para direccionar el segmento de código. Para este fin, usted tiene que indicar al ensamblador el propósito de cada segmento en el programa. La directiva para este propósito es ASSUME, codificada en el segmento de código como sigue:

| OPERACIÓN | OPERANDO |
|---|---|
| ASSUME | SS:nompila,DS: nomsegdatos,CS:nomsegcódigo, . . . |

SS:nompila significa **que el** ensamblador asocia el nombre del segmento de la pila con el registro SS, y de manera similar con los otros operandos mostrados. Los operandos pueden aparecer en cualquier orden. ASSUME también puede contener una entrada para el ES, tal como ES:nomsegdatos; si su programa no utiliza el registro ES, puede omitir su referencia o codificar ES:NOTHING (desde el MASM 6.0, el ensamblador de forma automática genera un ASSUME para el segmento de código).

Al igual que otras directivas, ASSUME es sólo un mensaje que ayuda al ensamblador a convertir código simbólico a código de máquina; aún puede tener que codificar instrucciones que físicamente cargan direcciones en registros de segmentos en el momento de la ejecución.

### Directiva END

Como ya se mencionó, la directiva ENDS finaliza un segmento y la directiva ENDP finaliza un procedimiento. Una directiva END finaliza todo el programa. Su formato general es:

| OPERACIÓN | OPERANDO |
|---|---|
| END | [nomproc] |

El operando puede estar en blanco si el programa no es para ejecutarse; por ejemplo, usted puede ensamblar sólo las definiciones de datos o puede querer enlazar el programa con otro módulo (principal). En la mayoría de los programas, el operando contiene el nombre del primero o único PROC designado como FAR, donde inicia la ejecución del programa.

## CÓMO INICIALIZAR UN PROGRAMA PARA SU EJECUCIÓN

Los dos tipos básicos de programas ejecutables son .EXE y .COM. Primero desarrollaremos los requisitos para programas .EXE y dejamos los programas .COM para el capítulo 7. La figura 4-1 proporciona una estructura de un programa .EXE que muestra los segmentos de la pila, de los datos y del código.

Examinemos las instrucciones del programa por número de línea:

| LÍNEA | EXPLICACIÓN |
|---|---|
| 1 | La directiva PAGE para este listado establece 60 líneas y 132 columnas por página. |
| 2 | La directiva TITLE identifica el nombre del programa P04ASM 1. |
| **3** | Las líneas 3, 7 y 11 son comentarios que clarifican la declaración de los segmentos definidos. |
| 4-6 | Estos enunciados definen el segmento de la pila, STACKSG (pero no su contenido, en este ejemplo). |
| 8-10 | Estos enunciados definen el segmento de datos, DATASG (pero no su contenido). |
| 12-21 | Estos enunciados definen el segmento de código, CODESG. |
| 13-20 | Estos enunciados definen el segmento de código del único procedimiento, llamado BEGIN. Este procedimiento ilustra los requisitos comunes de inicialización y de salida para un programa .EXE. Los dos requisitos para inicializar son (1) avisar al |

```
 1           PAGE    60,132
 2           TITLE   P04ASM1  Estructura de un programa  .EXE
 3
 4   STACKSG SEGMENT PARA  STACK 'Stack'

 6   STACKSG ENDS
 7
 8   DATASG  SEGMENT PARA  'Data'
 g
10   DATASG  ENDS
11
12   CODESG  SEGMENT PARA ' Code
13   BEGIN   PROC    FAR
14           ASSUME  SS:STACKSG, DS:DATASG,CS:CODESG
15           MOV     AX,DATASG    /Obtiene dirección del segmento de datos
16           MOV     DS, AX       ,-Almacena dirección en DS
17
18           MOV     AX,4C00H     ;Petición
19           INT     21H          ;Salida a DOS
20   BEGIN   ENDP
21   CODESG  ENDS
22           END     BEGIN
```

Figura **4-1** Estructura de un programa .EXE

ensamblador qué segmentos asocia con los registros de segmentos y (2) cargar el DS con la dirección del segmento de datos.

14 La directiva ASSUME avisa al ensamblador que asocie ciertos segmentos con ciertos registros de segmento, en este caso, STACKSG con el SS, DATASG con el DS y CODESG con el CS:

```
ASSUME   SS:STACKSG,DS:DATASG,CS;CODESG
```

Al asociar segmentos con registros de segmentos, el ensamblador puede determinar las direcciones de desplazamientos para los elementos en la pila, para los elementos en el segmento de datos y para las instrucciones en el segmento de código. Por ejemplo, cada instrucción de máquina en el segmento de código es de una longitud específica. La primera instrucción en lenguaje de máquina tendría un desplazamiento de 0 y si es de dos bytes de longitud, la segunda instrucción tendría un desplazamiento de 2 y así sucesivamente.

15,16 Dos instrucciones inicializan la dirección del segmento de datos en el registro DS:

```
MOV    AX,DATASG     ;Obtiene la dirección del segmento de datos

MOV    DS,AX         /Almacena la dirección en DS
```

El primer MOV carga la dirección del segmento de datos en el registro AX y el segundo MOV copia la dirección del AX al DS. Se requieren dos MOV ya que ninguna instrucción puede mover datos de forma directa de la memoria a un registro de segmento; usted tiene que mover la dirección desde otro registro al registro del segmento. Así, el enunciado MOV DS,DATASG sería ilegal. El capítulo 5 estudia cómo inicializar los registros de segmento con mayor detalle.

18,19 Estas dos instrucciones hacen la petición de terminación del programa y regresan al DOS. Una sección posterior las estudia con mayor detalle.

22 El enunciado END indica al ensamblador que éste es el final del programa y el operando BEGIN proporciona el punto de entrada para la ejecución subsecuente del programa.

La secuencia en la que define los segmentos por lo regular no es importante. La figura 4-1 los define como sigue:

```
STACKSG  SEGMENT  PARA  STACK  'Stack'
DATASG   SEGMENT  PARA  'Data'
CODESG   SEGMENT  PARA  'Code'
```

Tenga esto en mente: el programa en la figura está codificado en *lenguaje simbólico.* Para ejecutarlo, usted tiene que usar un programa ensamblador y un enlazador para traducirlo a código ejecutable de máquina. En ese caso, se convertiría en un programa .EXE.

Como se dijo en el capítulo 2, cuando el DOS carga un programa .EXE del disco a la memoria para su ejecución construye un PSP de 256 bytes (100H) en un límite de párrafo en memoria interna disponible y almacena el programa inmediatamente después del límite. Después, el DOS

- carga la dirección del segmento de código en el CS;
- carga la dirección de la pila en el SS; y
- carga la dirección del PSP en los registros DS y ES.

El cargador del DOS inicializa los registros CS:IP y SS:IP, pero no los registros DS y ES. Sin embargo, por lo común su programa necesita la dirección del segmento de datos en el DS (y con frecuencia también en el ES). Como consecuencia, tiene que inicializar el DS con la dirección del segmento de datos, como se muestra con las dos instrucciones MOV en la figura 4-1.

Ahora, aunque en este momento esta inicialización no sea clara, am'mese: cada programa .EXE tiene virtualmente los mismos pasos de inicialización que usted puede duplicar cada vez que codifique un programa en ensamblador.

## CÓMO TERMINAR LA EJECUCIÓN DE UN PROGRAMA

INT 21H es una operación de interrupción común del DOS que utiliza un código de función en el registro AH para especificar una acción que será realizada. Las diferentes funciones de INT 21H incluyen entrada desde el teclado, manejo de la pantalla, E/S de disco y salida a impresora. La función que nos interesa aquí es la 4CH, que INT 21H reconoce como una petición para la terminación de la ejecución de un programa. También puede usar esta operación para pasar un código de regreso en el AL para pruebas subsecuentes por medio de un archivo de procesamiento por lotes (vía el enunciado IF ERRORLEVEL), como sigue:

```
MOV   AH,4CH          ;Solicitud de terminación
MOV   AL,retcode      ,• Código de regreso (opcional)
INT   21H             ;Salir al DOS
```

El código de regreso para una terminación normal de un programa por lo común es 0 (cero). También puede codificar dos MOV como un enunciado (como se muestra en la figura 4-1):

```
MOV  AX,4C00H        ;Petición de terminación normal
```

La función 4CH del DOS ha sustituido las operaciones originales de terminación INT 20H e INT 21H, función OOH.

## EJEMPLO DE UN PROGRAMA FUENTE

La figura 4-2 combina la información precedente en un programa fuente en ensamblador, sencillo pero completo, que suma dos elementos de datos en el registro AX.

STACKSG contiene una entrada, DW (definir palabra), que define 32 palabras inicializadas a cero, un tamaño adecuado para la mayoría de los programas.

DATASG define tres palabras de datos llamadas FLDA, FLDB y FLDC.

CODESG contiene las instrucciones ejecutables para el programa, aunque el primer enunciado, ASSUME, no genera código ejecutable.

La directiva ASSUME realiza estas operaciones:

- Asigna STACKSG al registro SS, de forma que el sistema utilice la dirección en el registro SS para direccionamiento de STACKSG.
- Asigna DATASG al registro DS, de modo que el sistema utilice la dirección en el registro DS para direccionamiento de DATASG.
- Asigna CODESG al registro CS, de modo que el sistema utilice la dirección en el registro CS para direccionamiento de CODESG.

```
          page  60,132
TITLE     P04ASM1  (EXE)   Operaciones de mover y sumar

STACKSG   SEGMENT  PARA STACK  'Stack'
          DW       32  DUP(0)
STACKSG   ENDS

DATASG    SEGMENT  PARA  'Data'
FLDA      DW       250
FLDB      DW       125
FLDC      DW       ?
DATASG    ENDS

CODESG    SEGMENT  PARA  'Code'
BEGIN     PROC     FAR
          ASSUME   SS:STACKSG,DS:DATASG,CS:CODESG
          MOV      AX,DATASG         ;Se asigna dirección de DATASG
          MOV      DS,AX             ;  en registro DS

          MOV      AX,FLDA           ,-Mover  0250  a  AX
          ADD      AX,FLDB           ;Sumar  0125  a  AX
          MOV      FLDC, AX          ¡Almacenar  suma  en  FLDC
          MOV      AX,4C00H          ,•Salida a DOS
          INT      21H
BEGIN     ENDP                       ;Fin  de  procedimiento
CODESG    ENDS                       ,-Fin  de  segmento
          END      BEGIN             ;Fin  de  programa
```

Figura **4-2** Programa fuente .EXE con los segmentos convencionales

Cuando se carga un programa desde el disco a la memoria para su ejecución, el cargador del sistema establece las direcciones reales en los registros CS y SS pero, como se mostró por las dos primeras instrucciones MOV, usted tiene que inicializar el registro DS (y ES).

En el capítulo 5 revisaremos el ensamble, enlace y ejecución de este programa.

## CÓMO INICIALIZAR EL MODO PROTEGIDO

En modo protegido bajo el 80386 y procesadores posteriores, un programa puede direccionar hasta 16 megabytes de memoria. El uso de DWORD para alinear segmentos en direcciones de palabras dobles incrementa la velocidad de acceso a memoria para buses de datos de 32 bits. En el código siguiente, la directiva .386 le indica al ensamblador que acepte instrucciones que son sólo para estos procesadores; el operando USE32 indica al ensamblador que genere código apropiado para el modo protegido de 32 bits:

```
.386
nomseg   SEGMENT   DWORD   USE32
```

La inicialización del registro del segmento de datos podría parecerse a esto, ya que en estos procesadores el registro DS aún tiene un tamaño de 16 bits:

```
MOV    EAX,DATASEG    ;Obtiene la dirección del segmento de datos
MOV    DS,AX          ;Carga la parte de 16 bits
```

Las instrucciones STI, CLI, IN y OUT, disponibles en modo real, no están permitidas en modo protegido.

## DIRECTIVAS SIMPLIFICADAS DE SEGMENTOS

Los ensambladores de Microsoft y de Borland proporcionan algunas formas abreviadas para definir segmentos. Para usar estas abreviaturas, inicialice el *modelo de memoria* antes de definir algún segmento. El formato general (incluyendo el punto inicial) es

```
.MODEL  modelo de memoria
```

El modelo de memoria puede ser TINY, SMALL, MÉDIUM, COMPACT o LARGE (otro modelo, HUGE, no necesitamos tratarlo aquí). Los requisitos para cada modelo son:

| MODELO | NÚMERO DE SEGMENTOS DE CÓDIGO | NÚMERO DE SEGMENTOS DE DATOS |
|---|---|---|
| TINY | * | * |
| SMALL | 1 | 1 |
| MÉDIUM | Más de 1 | 1 |
| COMPACT | 1 | Más de 1 |
| LARGE | Más de 1 | Más de 1 |

Puede utilizar cualquiera de estos modelos para un programa autónomo (esto es, un programa que no esté enlazado con algún otro). El modelo TINY está destinado para uso exclusivo de progra-

mas .COM, los cuales tienen sus datos, código y pila en un segmento. El modelo SMALL exige que el código quepa en un segmento de 64K y los datos en otro segmento de 64K; este modelo es adecuado para la mayor parte de los ejemplos de este libro. La directiva .MODEL genera de forma automática el enunciado ASSUME necesario.

Los formatos generales (incluyendo el punto inicial) para las directivas que define los segmentos de la pila, de datos y de código son:

```
.STACK   [tamaño]

.DATA

.CODE    [nombre]
```

Cada una de estas directivas hacen que el ensamblador genere el enunciado SEGMENT necesario y su correspondiente ENDS. Los nombres por omisión de los segmentos (que usted no tiene que definir) son STACK, DATA y TEXT (para el segmento de código). El carácter de subrayado al inicio de DATA y TEXT es intencional. Cuando el formato codificado lo indica, puede no hacer caso al nombre por omisión del segmento de código. El tamaño, por omisión, de la pila es de 1,024 bytes, el cual también puede pasarse por alto. Se utilizan estas directivas para identificar en dónde, en el programa, están ubicados los tres segmentos. Sin embargo, note que las instrucciones que ahora usa para inicializar la dirección del segmento de datos en el DS son:

```
MOV    AX,©datos

  MOV    DS,AX
```

La figura 4-2 dio un ejemplo de un programa que utiliza segmentos definidos de modo convencional. La figura 4-3 proporciona el mismo ejemplo, pero esta vez usando las directivas simplifica-

```
        page    50,132
TITLE   P04ASM2 (EXE)   Operaciones de mover y sumar

        .MODEL  SMALL
        .STACK  G4                  ;Se define la pila
        .DATA                       ;Se definen los datos
FLDA    DW      250
FLDB    DW      125
FLDC    DW      ?

        .CODE                       ;Se define el segmento de código
BEGIN   PROC    FAR
        MOV     AX,@data            ;Se asigna la dirección de DATASG
        MOV     DS,AX               ;  en el registro DS

        MOV     AX,FLDA             ;Mover 0250 a AX
        ADD     AX,FLDB             ;Sumar 0125 a AX
        MOV     FLDC,AX             ,-Almacenar suma en FLDC

        MOV     AX,4C00H            ,-Salida a DOS
        INT     21H
BEGIN   ENDP                        ;Fin de procedimiento
        END     BEGIN               ;Fin de programa
```

Figura **4-3** Programa fuente .EXE con directivas simplificadas de segmentos

das de segmentos .STACK, .DATA y .CODE. En la cuarta línea, el modelo de memoria es especificado como SMALL. La pila está definida como 64 bytes (32 palabras). Advierta que el ensamblador no genera los enunciados convencionales SEGMENT y ENDS, y que tampoco se codifica un enunciado ASSUME.

Como verá en el siguiente capítulo, el ensamblador maneja programas codificados con directivas simplificadas de segmentos de forma un poco diferente de aquella que utiliza directivas convencionales de segmentos.

### Las directivas .STARTUP y .EXIT

MASM 6.0 introdujo las directivas .STARTUP y .EXIT para simplificar la inicialización y terminación de programas. .STARTUP genera las instrucciones para inicializar los registros de segmentos, mientras que .EXIT genera las instrucciones de la INT 21H, función 4CH para la salida del programa. Para propósitos de aprendizaje del lenguaje ensamblador, los ejemplos en este texto codifican el conjunto completo de instrucciones y dejan las formas abreviadas para los programadores con más experiencia.

## DEFINICIÓN DE DATOS

Como ya se estudió, el propósito del segmento de datos en un programa .EXE es definir constantes, áreas de trabajo y áreas de entrada/salida. El ensamblador permite la definición de elementos de varias longitudes de acuerdo con el conjunto de directivas que defina datos. Por ejemplo, DB define un byte y DW define una palabra. Un elemento de datos puede contener un *valor indefinido* (esto es, no inicializado) o una *constante,* definida como una cadena de caracteres o como un valor numérico. A continuación está el formato general para la definición de datos:

```
[nombre]  Dn  expresión
```

**Nombre.** Un programa que hace referencia a un elemento de dato lo hace por medio de un nombre. Por otro lado, el nombre de un elemento es opcional, indicado por los corchetes. La sección anterior "Instrucciones", proporciona las reglas para la formación de los nombres.

**Directivas.** Las directivas que definen elementos de datos son DB (byte), DW (palabra), DD (palabra doble), DF (palabra larga), DQ (palabra cuádruple) y DT (diez bytes), cada una indica de manera explícita la longitud del elemento definido.

**Expresión.** La expresión es un operando que puede contener un signo de interrogación para indicar un elemento no utilizado, como

```
FLDl  DB  ?      ;Elemento no inicializado
```

En este caso, cuando su programa inicie la ejecución el valor inicial de FLDl no es conocido por usted. En la práctica, lo normal antes de usar este elemento es mover algún valor a él (lo que sea, pero debe ser apropiado al tamaño definido).

También puede utilizar el operando para definir una constante, como

```
FLD2  DB  25      ;Elemento inicializado
```

Puede usar con libertad este valor inicializado en su programa y aun puede cambiar el contenido de FLD2.

Una expresión puede contener varios valores constantes separados por comas y limitados sólo por la longitud de la línea, como sigue:

```
FLD3  DB  11,  12,  13,  14,  15,  16,
```

El ensamblador define estas constantes en bytes contiguos. Una referencia a FLD3 es a la primera constante de un byte, 11 (puede pensar en el primer byte como FLD3+0), y una referencia a FLD3 + 1 es a la segunda constante, 12. Por ejemplo, la instrucción

```
MOV  AL,FLD3+3
```

carga el valor 14 (OEH) en el registro AL. También la expresión permite duplicación de constantes en un enunciado de la forma general

```
[nombre]  Dn  contador de repeticiones DUP  (expresión)
```

Los ejemplos siguientes ilustran la duplicación:

```
DW   10 DUP(?)            Diez palabras, no inicializadas
DB   5 DUP(14)            Cinco palabras con 0E0E0E0E0E hexadecimal
DB   3 DUP(4 DUP(8) )     Doce 8
```

El tercer ejemplo genera cuatro copias del dígito 8 (8888) y duplica él valor tres veces, produciendo en total doce 8.

Una expresión puede definir e inicializar una cadena de caracteres o una constante numérica.

## Cadenas de caracteres

Las cadenas de caracteres son usadas para datos descriptivos como nombres de personas y títulos de páginas. La cadena está definida dentro de apóstrofos, como 'PC, o dentro de comillas, como "PC". El ensamblador traduce las cadenas de caracteres en código objeto en formato ASCII normal.

Extrañamente, DB es el único formato que define una cadena de caracteres que excede a dos caracteres y los almacena en la secuencia normal de izquierda a derecha. En consecuencia, DB es el formato convencional para la definición de datos de caracteres de cualquier longitud. Un ejemplo es

```
DB  'Cadena de caracteres'
```

El ensamblador almacena los caracteres en formato ASCII, sin apóstrofos. Si la cadena debe contener un apóstrofo o una comilla, usted puede definirlo en una de las forma siguientes:

```
DB  "Honest Ed's PC Emporium"     /Comillas para la cadena,
                                  una comilla para el apóstrofo
DB  'Honest Ed''s PC Emporium'    /Una comilla para la cadena,
                                  dos comillas seguidas para el apóstrofo
```

### Constantes numéricas

Las constantes numéricas son usadas para definir valores aritméticos y direcciones de memoria. Las constantes no están definidas entre comillas, pero van seguidas por un *especificador de base* opcional, tal como H en el valor hexadecimal 12H. Para la mayoría de las directivas de definición de datos, el ensamblador convierte constantes numéricas definidas a hexadecimal y almacena los bytes generados en código objeto en orden inverso —de derecha a izquierda. A continuación están los diferentes formatos numéricos.

**Decimal.** El formato decimal permite definir con los dígitos decimales 0 a 9, seguidos de manera opcional por el especificador de base D, tal como 125 o 125D. Aunque el ensamblador permite que usted defina valores en formato decimal, como una conveniencia al codificar, él convierte sus valores decimales a código objeto binario y los representa en hexadecimal. Por ejemplo, una definición del decimal 125 se convierte en 7D hexadecimal.

**Hexadecimal.** El formato hexadecimal permite definir con los dígitos hexadecimales 0 a F, seguidos por el especificador de base H, que se puede usar para definir valores binarios. Ya que el ensamblador espera que una referencia que empiece con una letra es un nombre simbólico, el primer dígito de una constante hexadecimal debe ser 0 a 9. Ejemplos son 2EH yOFD8H, que el ensamblador almacena como 2E y D80F, respectivamente. Note que los bytes en el segundo ejemplo son almacenados en orden inverso.

**Binario.** El formato binario permite definir con los dígitos binarios 0 y 1, seguidos por el especificador de base B. El uso normal del formato binario es para distinguir valores en las instrucciones de manejo de bits AND, OR, XOR y TEST.

Ya que el ensamblador convierte todos los valores numéricos a binario (y los representa en hexadecimal), las definiciones de 12, C hex y 1100 binario generan el mismo valor: 00001100 binario o 0C hex, dependiendo de cómo vea el contenido del byte.

Cómo las letras D y B actúan tanto como especificadores de base como dígitos hexadecimales, pueden causar alguna confusión. Como solución, MASM 6.0 introdujo el uso de la T (por *ten,* diez) y la Y (por *binary,* binario) como especificadores de base para decimal y binario, respectivamente.

**Real.** El ensamblador convierte un valor real dado —una constante decimal o hexadecimal seguida por el especificador de base R— en formato de punto flotante para uso con un coprocesador matemático.

Asegúrese de distinguir entre el uso de las constantes numéricas y de caracteres. Una constante de carácter definida como DB '12' genera dos caracteres ASCII, representados como 3132 hex. Una constante numérica definida como 12 genera un número binario, representado como 0C hex.

## DIRECTIVAS PARA LA DEFINICIÓN DE DATOS

Las directivas convencionales usadas para definir datos, junto con los nombres introducidos por MASM 6.0, son:

| DESCRIPCIÓN | DIRECTIVAS CONVENCIONALES | DIRECTIVAS MASM 6.0 |
|---|---|---|
| Definir byte(s) | DB | BYTE |
| Definir una palabra | DW | WORD |

```
Definir una palabra doble              DD        DWORD

Definir una palabra larga              DF        FWORD

Definir una palabra cuádruple          DQ        QWORD

Definir diez bytes                     DT        TBYTE
```

El texto utiliza las directivas convencionales porque su uso es aceptado de manera general.

El programa ensamblado de la figura 4-4 proporciona ejemplos de las directivas que definen cadenas de caracteres y constantes numéricas, con el código objeto generado a la izquierda, el cual

```
                                page 60,132
                        TITLE   P04DEFIN (EXE)   Define data items
                                .MODEL   SMALL
                                .DATA
                                Se definen Bytes - DB:

0000 00                 FLD1DB  DB   ?             /No se inicia
0001 20                 FLD2DB  DB   32            /Constante decimal
0002 20                 FLD3DB  DB   20H           ;Constante hexadecimal
0003 59                 FLD4DB  DB   01011001B     ;Constante binaria
0004 OOOAt 00           FLD5DB  DB   10 DUP(0)     /Diez ceros
000E 50 65 72 73 6F 6E  FLD6DB  DB   'Personal Computer'
     61 6C 20 43 6F 6D
     70 75 74 65 72                                ;Cadena de caracteres
001F 33 32 36 35 34     FLD7DB  DB   '32654'       /Números como caracteres
0024 01 4A 61 6E 02 46  FLD8DB  DB   01,'Jan',02,'Feb',03,'Mar'
     65 62 03 4D 61 72                             ,-Tabla de meses

                                Se definen Words - DW:

0030  FFF0               FLD1DW  DW   0FFF0H        Constante hexadecimal
0032  0059               FLD2DW  DW   01011001B     Constante binaria
0034  001F R             FLD3DW  DW   FLD7DB        Constante de dirección
0036  0003 0004 0007     FLD4DW  DW   3,4,7,8,9     Tabla de cinco
      0008 0009                                     constantes
0040  0005[ 0000 ]       FLD5DW  DW   5 DUP(0)      Cinco ceros

                        ,•      Se definen Double Words   DD:

004A 00000000           FLD1DD  DD   ?             ;No se inicia
004E 00007F3C           FLD2DD  DD   32572         /Valor decimal
0052 OOOOOOOE 00000031  FLD3DD  DD   14,49         ;Dos constantes
005A 00000001           FLD4DD  DD   FLD3DB - FLD2DB   /Diferencia
                                                  ;  entre direcciones
005E 00005043           FLD5DD  DD   'PC           ,-Cadena de caracteres

                        ;       Se definen Quad Words - DQ:

0062 0000000000000000   FLD1DQ  DQ                 No se inicia
006A 474D000000OOOOO0   FLD2DQ  DQ                 Constante hexadecimal
0072 3C7FOOOOOOOOOOO0   FLD3DQ  DQ                 Constante decimal

                        i       Se

007A 000000000000000000 FLD1DT  DT                 /No se inician
     00
0084 563412000000000000 FLD2DT  DT                 /Constante decimal
     00
008E 435000000000000000 FLD3DT  DT                 /Cadena de caracteres
     00
```

Figura **4-4** Definiciones de cadenas de caracteres y valores numéricos (parte 1 de 2)

```
Segments and Groups:
          Name         Length    Align     Combine    Class
DGROUP . . . . . .  .GROUP
  _DATA . . . . . .  .0098      WORD      PUBLIC     'DATA'
_TEXT . . . . . .  .0000      WORD      PUBLIC     'CODE'

Symbols:
          Name         Type      Valué      Attr
FLD1DB . . . . . .  .L BYTE     0000       _DATA
FLD1DD . . . . . .  .L DWORD    004A       _DATA
FLD1DQ . . . . . .  .L QWORD    0062       JDATA
FLD1DT . . . . . .  .L TBYTE    0 07A      _DATA
FLD1DW . . . . . .  .L WORD     003 0      _DATA
FLD2DB . . . . . .  .L BYTE     0001       _DATA
FLD2DD . . . . . .  .L DWORD    004E       _DATA
FLD2DQ . . . . . .  .L QWORD    006A       _DATA
FLD2DT . . . . . .  .L TBYTE    0084       _DATA
FLD2DW . . . . . .  .L WORD     0032       _DATA
FLD3DB . . . . . .  .L BYTE     0002       _DATA
FLD3DD . . . . . .  .L DWORD    0052       _DATA
FLD3DQ . . . . . .  .L QWORD    0072       _DATA
FLD3DT . . . . . .  .L TBYTE    008E       _DATA
FLD3DW . . . . . .  .L WORD     0 034      _DATA
FLD4DB . . . . . .  .L BYTE     0 0 03     _DATA
FLD4DD . . . . . .  .L DWORD    00 5A      _DATA
FLD4DW . . . . . .  .L WORD     00 3 6     _DATA
FLD5DB . . . . . .  .L BYTE     0004       _DATA    Length = 000A
FLD5DD . . . . . .  .L DWORD    005E       JDATA
FLD5DW . . . . . .  .L WORD     0040       _DATA    Length = 0005
FLD6DB . . . '. . . .L BYTE     O00E       _DATA
FLD7DB . . . . . .  .L BYTE     001F       _DATA
FLD8DB . . . . . .  .L BYTE     0024        DATA

     0 Warning Errors
     0 Severe  Errors
```

**Figura 4-4** (continuación)

lo exhortamos a examinar. Note que el código objeto para valores no inicializados aparece como ceros hexadecimales. Ya que este programa consiste sólo en un segmento de datos, no es adecuado para ejecución.

### Definir byte: DB o BYTE

De las directivas que definen elementos de datos, una de las más útiles es DB (definir byte).

Una expresión numérica DB (o BYTE) puede definir una o más constantes de un byte. El máximo de un byte significa dos dígitos hexadecimales. Con el bit de más a la izquierda actuando como el de signo, el número hexadecimal más grande positivo de un byte es 7F; todos los números "superiores", del 80 al FF (en donde el bit de signo es 1), representan valores negativos. En términos de números decimales, estos límites son +127 y -128. El ensamblador convierte constantes numéricas en código objeto binario (representado en hexadecimal). En la figura 4-4, constantes DB numéricas son FLD2DB, FLD3DB, FLD4DB y FLD5DB.

Una expresión de carácter DB puede contener una cadena de cualquier longitud, hasta el final de la línea. Por ejemplo, vea FLD6DB y FLD7DB en la figura. El código objeto muestra el carácter ASCII para cada byte en orden normal de izquierda a derecha; 20H representa un carácter espacio en blanco.

FLD8DB muestra una mezcla de constantes numéricas y de cadenas de caracteres adecuada para definir una tabla.

**Definir una palabra: DW o WORD**

La directiva DW define elementos con una longitud de una palabra (dos bytes). Una expresión numérica DW (o WORD) puede definir una o más constantes de una palabra. El número hexadecimal positivo de una palabra es 7FFF; todos los números "superiores", desde 8000 hasta FFFF (donde el bit de signo es 1), representan valores negativos. En términos de números decimales, los límites son +32,767 y -32,768.

El ensamblador convierte constantes numéricas DW a código objeto binario (representado en hexadecimal), pero almacena los bytes en orden inverso. En consecuencia, un valor decimal definido como 12345 lo convierte a 3039 hex, pero es almacenado como 3930.

En la figura 4-4, FLD1DW y FLD2DW definen constantes numéricas DW. FLD3DW define el operando como una dirección —en este caso, la dirección desplazada de FLD7DB. El código objeto generado es 001F (la R a la derecha significa *reubicable),* y una inspección de la figura muestra que la dirección desplazada de FLD7DB (la columna de la extrema izquierda) en realidad es 001F.

Una expresión de caracteres DW está limitada a dos caracteres, que el ensamblador invierte en el código objeto, así que 'PC se convertiría en 'CP'. Si piensa que DW es de uso limitado para la definición de cadenas de caracteres, está en lo correcto.

FLD4DW define una tabla de cinco constantes numéricas. Note que la longitud de cada constante es de una palabra (dos bytes).

**Definir palabra doble: DD o DWORD**

La directiva DD define elementos que tienen longitud de dos palabras (cuatro bytes). Una expresión numérica DD (o DWORD) puede definir una o más constantes, cada una con un máximo de cuatro bytes (ocho dígitos hexadecimales). El número hexadecimal más positivo en una palabra doble es 7FFFFFFF (en donde el bit de signo es 1), todos los números "superiores", desde 80000000 hasta FFFFFFFF (en donde el bit de signo es 1), representan valores negativos. En términos de números decimales, estos máximos son +2,147,483,647 y -2,147,483,648.

El ensamblador convierte las constantes numéricas DD a código objeto binario (representado en hexadecimal), pero almacena los bytes en orden inverso. En consecuencia, un valor decimal definido como 12345678 se convierte en 00BC614EH, pero es almacenado como 4E61BC00H.

En la figura 4-4, FLD2DD define una constante numérica DD, y FLD3DD define dos constantes numéricas. FLD4DD genera la diferencia numérica entre las dos direcciones definidas; en este caso, el resultado es la longitud de FLD2DB.

Una expresión de carácter DD también está limitada a dos caracteres y es tan trivial como la de DW. El ensamblador invierte los caracteres y los ajusta a la izquierda en una palabra doble de cuatro bytes, como se muestra en el código objeto para FLD5DD.

**Definir palabra larga: DF o FWORD**

La directiva DF define una palabra larga de seis bytes. Su uso normal es para el 80386 y procesadores posteriores.

**Definir palabra cuádruple: DQ o QWORD**

La directiva DQ define elementos que tienen una longitud de cuatro palabras (ocho bytes). DQ (o QWORD) de una expresión numérica puede definir una o más constantes, cada una con un máximo de ocho bytes, o 16 dígitos hexadecimales. El mayor número hexadecimal positivo de cuatro

palabras es 7 seguido de 15 F. Como un indicio de la magnitud de este número, el número hexadecimal 1 seguido de 15 ceros es igual al número decimal 1,152,921,504,606,846,976.

El ensamblador maneja la DQ de valores numéricos y cadenas de caracteres igual que lo hace DD y DW para valores numéricos. En la figura 4-4, FLD2DQ y FLD3DQ ilustran sólo valores numéricos.

### Definir diez bytes: DT o TBYTE

La directiva DT define elementos de datos que son de 10 bytes de longitud. Su propósito está relacionado con los valores numéricos empacados BCD (decimal codificado en binario), que son más útiles para coprocesadores matemáticos que para operaciones aritméticas estándar. Un número BCD está empacado con dos dígitos decimales por byte, con el último bit de la izquierda como el bit de signo (0 o 1). Para una constante definida como 12345678, el ensamblador almacena los bytes en orden inverso como 78 56 34 12 00 00 00 00 00 00. Note que DT (o TBYTE), a diferencia de las otras directivas de datos, almacena constantes numéricas como decimal en lugar de valores hexadecimales.

La figura 4-4 ilustra DT para un elemento no inicializado, un valor numérico y una constante de dos caracteres.

### Desplegar el segmento de datos

El programa en la figura 4-4 contiene sólo un segmento de datos. Aunque el ensamblador no generó mensajes de error, el mapa del enlace desplegó "Warning: No STACK segment" (Advertencia: No existe segmento de la PILA) y el enlazador mostró "There were 1 errors detected" (Se detectó un error). A pesar de las advertencias, usted aún puede utilizar DEBUG para ver el código objeto, el cual se muestra en la figura 4-5.

Ensamble y enlace el programa, utilice DEBUG para cargar el archivo .EXE e ingrese D DS:100 para mostrar los datos. El lado derecho del despliegue muestra la representación ASCII, tal como "Personal Computer", mientras que los valores hexadecimales a la izquierda indican los contenidos realmente almacenados. Su desplegado debe ser idéntico al de la figura 4-5 para los

```
0F07  0000   00 20 20 59 00 00 00 00-00 00 00 00 00 00 50 65   Y..........Pe
0F07  0010   72 73 6F 6E 61 6C 20 43-6F 6D 70 75 74 65 72 33   rsonal Computer3
0F07  0020   32 36 35 34 01 4A 61 6E-02 46 65 62 03 4D 61 72   2654.Jan.Feb.Mar
0F07  0030   F0 FF 59 00 1F 00 03 00-04 00 07 00 08 00 09 00   ..Y.............
0F07  0040   00 00 00 00 00 00 00 00-00 00 00 00 00 00 3C 7F   ..............<.
0F07  0050   00 00 0E 00 00 00 31 00-00 00 01 00 00 00 43 50   ......1.......CP
0F07  0060   00 00 00 00 00 00 00 00-00 00 47 4D 00 00 00 00   ..........GM....
0F07  0070   00 00 3C 7F 00 00 00 00-00 00 00 00 00 00 00 00   ..<.............
 n
 u
0F07  0080   00 00 00 00 56 34 12 00-00 00 00 00 00 00 43 50   ....V4......CP
0F07  0090   00 00 00 00 00 00 00 00-00 72 03 E9 6B 01 2B CO 50   .........r..k.+.P
0F07  00A0   50 FF 76 04 E8 F5 5D 83-C4 06 0B DO 74 03 E9 57   P.v...].....t..W
0F07  00B0   01 B8 FF FF 50 2B CO 50-FF 76 04 E8 DE 5D 83 C4   ....P+.P.v...]..
0F07  OOCO   06 8B 1E A4 43 FF 06 A4-43 DI E3 DI E3 Al 0E 3C   ....C...C.....<
0F07  OOD0   8B 16 10 3C 89 87 8A 32-89 97 8C 32 5E 8B E5 5D   ...<...2...2"..]
OF07  00E0   C3 90 B8 05 00 50 B8 CC-07 50 8D 46 80 50 E8 23   .....P...P.F.P.#
0F07  00F0   6C 83 C4 06 FF 76 04 8D-46 80 50 E8 98 0D 83 C4   1....v..F.P.....

          <—      - Representación hexadecimal –      —>   <_____ASCII_____>
```

**Figura 4-5** Despliegue del segmento de datos

desplazamientos 0000 hasta 0097. Esperamos que difieran la dirección de su segmento (0F07 en la figura) y los datos después del desplazamiento 0097.

Usted dio la instrucción DS: 100 para el despliegue porque el cargador estableció DS con la dirección del PSP, y el segmento de datos para este programa es 100 bytes después de esa dirección. Luego, cuando use DEBUG para programas .EXE que inicializan el DS con la dirección del segmento de datos, usará DS:0 para desplegarlo.

## LA DIRECTIVA EQU

La directiva EQU no define elementos de datos. En lugar de eso, define un valor que el ensamblador puede usar para sustituir en otras instrucciones. Considere el enunciado EQU siguiente, codificado en el segmento de datos:

```
TIMES  EQU  10
```

El nombre, en este caso TIMES, puede ser cualquier nombre aceptable por el ensamblador. Ahora, siempre que en una instrucción o en otra directiva aparezca la palabra TIMES, el ensamblador la sustituye por el valor 10. Por ejemplo, el ensamblador convierte la directiva

```
FIELDA  DB  TIMES  DUP(?)
```

a su valor equivalente

```
FIELDA  DB  10  DUP(?)
```

Una instrucción también puede tener un operando con EQU, como en el siguiente:

```
COUNTR   EQU  05

         MOV  CX,COUNTR
```

El ensamblador reemplaza COUNTR en el operando MOV con el valor 05, haciendo del operando un valor inmediato, como si estuviera codificado

```
MOV  CX,05      ;E1 ensamblador sustituye 05
```

La ventaja de EQU es que muchos enunciados pueden utilizar valores definidos por COUNTR. Si el valor ha sido cambiado, sólo necesita cambiar el enunciado EQU. No necesita decirse que puede usar un valor igualado (con EQU) sólo en donde una sustitución tenga sentido para el ensamblador. También puede igualar (con EQU) nombres simbólicos, como en el siguiente código:

```
TOTALPAY  DW  0

TP   EQU  TOTALPAY
MFY       EQU  MUL
```

El primer EQU hace equivalente (iguala) el alias TP al elemento definido TOTALPAY. Para cualquier instrucción que tenga el operando TP, el ensamblador lo reemplaza con la dirección de

TOTALPAY. El segundo EQU permite a un programa usar la palabra MPY en lugar de la instrucción simbólica MUL.

MASM 6.0 introdujo una directiva TEXTEQU, para datos de texto, con el formato

```
nombre  TEXTEQU  <TEXTO>
```

## PUNTOS CLAVE

- Un comentario está precedido por punto y coma (;).
- Las palabras reservadas en lenguaje ensamblador son usadas para propósitos especiales, bajo condiciones especiales.
- Un identificador es un nombre que se aplica a elementos en sus programas. Los dos tipos de identificadores son *nombres*, que se refieren a direcciones de datos, y *etiquetas*, que se refieren a la dirección de una instrucción.
- Una operación es usada, por lo común, para definir áreas de datos y codificar instrucciones. Un operando proporciona información para la información que actúa sobre él.
- Un programa consiste en uno o más segmentos, cada uno de los cuales empieza en un límite de párrafo.
- La directiva ENDS finaliza cada segmento, ENDP termina cada procedimiento y END termina un programa.
- La directiva ASSUME asocia los registros de segmentos CS, DS y SS con sus nombres de segmento apropiados. '
- Los programas .EXE (pero no los .COM) deben proporcionar al menos 32 palabras para el direccionamiento de la pila.
- Para un programa .EXE, por lo general se inicializa el registro DS con la dirección del segmento de datos.
- Para las directivas simplificadas de segmentos, antes de definir algún segmento, se inicializa el modelo de memoria. Las opciones son SMALL (un segmento de código y un segmento de datos), MÉDIUM (cualquier número de segmentos de código y un segmento de datos), COMPACT (un segmento de código y cualquier número de segmentos de datos) y LARGE (cualquier número de segmentos de datos y de código).
- INT 21H, función 4CH, es la instrucción estándar para la salida de programas.
- Los nombres de los elementos de datos deben ser únicos y descriptivos. Por ejemplo, un elemento para el salario de un empleado podría ser SALEMP.
- DB es el formato preferido para la definición de cadenas de caracteres, ya que permite cadenas de más de dos bytes de longitud y las convierte a la secuencia normal de izquierda a derecha.
- Constantes decimales y binarias (hexadecimales) generan diferentes valores. Considere el efecto de sumar el 25 decimal en contra de sumar 25 hex:

```
ADD  AX,25      ;Suma 25
ADD AX,2 5H     ,-Suma 37
```

- DW, DD y DQ almacenan valores numéricos en código objeto, con los bytes en orden inverso.
- Los elementos DB son usados para procesar la mitad de registros (AL, BL, etc.). DW para registros completos (AX, BX, etc.), y DD para registros extendidos (EAX, EBX, etc.). Elementos numéricos más largos necesitan de manejo especial.

## PREGUNTAS

4-1. Señale las diferencias entre un compilador y un ensamblador.

4-2. ¿Qué es una palabra reservada en un lenguaje ensamblador? Dé dos ejemplos.

4-3. ¿Cuáles son los dos tipos de identificadores?

4-4. Determine cuáles de los nombres siguientes son válidos: (a) PCAT; (b) $50; (c) @$_Z; (d) 34B7; (e) AX.

4-5. ¿Cuáles son las diferencias entre una directiva y una instrucción?

4-6. ¿Qué comandos hacen que el ensamblador (a) imprima un encabezado en la parte superior de una página en el listado de un programa y (b) salte a una nueva página?

4-7. ¿Cuál es el objetivo de cada uno de los tres segmentos descritos en este capítulo?

4-8. El formato de la directiva SEGMENT es \

```
nombre SEGMENT alineación combinar 'clase'
```

Explique el objetivo de (a) alineación; (b) combinar; (c) 'clase'.

4-9. (a) ¿Cuál es el objetivo de un procedimiento? (b) ¿Cómo define el inicio y el final de un procedimiento? (c) ¿Cuándo definiría un procedimiento como FAR y cuándo como NEAR?

4-10. Explique qué enunciados END particulares tratan la finalización de (a) un programa; (b) un procedimiento; (c) un segmento.

4-11. Establezca las diferencias entre los enunciados que finalizan un ensamblado y los enunciados que finalizan una ejecución.

4-12. Dé los nombres STKSEG, DATSEG y CDSEG a los segmentos de la pila, de los datos y del código, respectivamente, y codifique el ASSUME necesario.

4-13. Considere la instrucción MOV AX,4C00H utilizada con INT 21H. (a) ¿Qué hace la instrucción? (b) ¿Cuál es la finalidad del 4C y el 00?

4-14. Para las directivas simplificadas de segmentos, la directiva .MODEL proporciona los modelos TINY, SMALL, MÉDIUM, COMPACT y LARGE. ¿Bajo qué circunstancias se utilizaría cada uno de estos modelos?

4-15. Dé las longitudes, en bytes, generadas por las siguientes directivas de datos: (a) DD; (b) DW; (c) DT; (d) DQ; (e) DB.

4-16. Defina una cadena de caracteres con nombre TITLE1 que contenga la constante: RGB Electronics.

4-17. Defina los valores numéricos siguientes en elementos de datos FIELDA a FIELDE, respectivamente:

(a) Un elemento de cuatro bytes con el equivalente hexadecimal del 215 decimal.

**(b)** Un elemento de un byte con el equivalente hexadecimal del 35 decimal.

**(c)** Un elemento de dos bytes con un valor no definido.

**(d)** Un elemento de un byte con el equivalente binario del 25 decimal.

(e) Un DW con los valores consecutivos 17, 19, 21, 26 y 31.

4-18. Muestre el código objeto hexadecimal generado por (a) DB '28'; (b) DB 28.

4-19. Determine el código objeto hexadecimal ensamblado para (a) DB 28H; (b) DW 2845H; (c) DD 28733AH; (d) DQ 28733AH.

## CAPÍTULO 5

# Cómo ensamblar, enlazar y ejecutar un programa

OBJETIVO

**Analizar los pasos para ensamblar, enlazar y ejecutar un programa en lenguaje ensamblador.**

## INTRODUCCIÓN

Este capítulo explica el procedimiento para teclear un programa en lenguaje ensamblador y para ensamblarlo, enlazarlo y ejecutarlo. Las instrucciones simbólicas que codifica en lenguaje ensamblador, son conocidas como el *programa fuente.* Se utiliza el programa ensamblador para traducir el programa fuente en código de máquina, conocido como el *programa objeto.* Por último, se emplea un programa enlazador para completar el direccionamiento de máquina del programa objeto, generando un *módulo ejecutable.*

Las secciones sobre el ensamble explican cómo solicitar la ejecución del programa ensamblador, el cual provee de diagnósticos (incluyendo mensajes de error) y genera el programa objeto. También se explican los detalles del listado del ensamblador y, en términos generales, cómo el ensamblador procesa un programa fuente.

Las secciones sobre el enlace explican cómo solicitar la ejecución del programa enlazador de manera que pueda generar un módulo ejecutable. También son explicados los detalles del mapa de enlace generado, así como los diagnósticos. Por último, una sección explica cómo solicitar la ejecución de un módulo ejecutable.

## CÓMO PREPARAR UN PROGRAMA PARA SU EJECUCIÓN

La figura 4-2 sólo ilustró el código fuente de un programa, todavía no en formato ejecutable. Para teclear este programa, se puede usar un programa editor, tal como el proporcionado con el DOS. En los ejemplos siguientes de comandos DOS, sustituya lo apropiado para su sistema. También puede aumentar mucho la productividad cargando sus programas y archivos en un disco RAM (disco virtual). Llame a su programa editor, teclee los enunciados del programa en la figura 4-2 y al archivo resultante póngale por nombre P05ASM1.ASM.

Aunque para el ensamblador no es importante el espaciamiento, un programa será más legible si mantiene alineados por columnas y de manera consistente el nombre, operación, operandos y comentarios. La mayoría de los editores tienen marcas de tabulación cada ocho posiciones para facilitar la alineación de columnas.

Una vez que ha introducido todos los enunciados del programa, revise el código para ver si es correcto. La mayoría de los editores tiene una facilidad para imprimir, pero si no la tiene, encienda su impresora y utilice el programa PRINT del DOS:

```
PRINT n:P05ASMl.ASM  [Enter]
```

Tal como está, el programa es sólo un archivo de texto que no puede ejecutarse: primero debe ensamblarlo y enlazarlo.

1. El paso de *ensamble* consiste en la traducción del código fuente en código objeto y la generación de un archivo intermedio .OBJ (objeto), o módulo (en capítulos anteriores ya ha visto ejemplos de código de máquina y de código fuente). Una de las tareas del ensamblador es calcular el desplazamiento de cada elemento en el segmento de datos y de cada instrucción en el segmento de código. El ensamblador también crea un encabezado al frente del módulo .OBJ generado; parte del encabezado tiene información acerca de direcciones incompletas. El módulo .OBJ aún no está en forma ejecutable.
2. El paso de *enlace* implica convertir el módulo .OBJ en un módulo de código de máquina .EXE (ejecutable). Una de las tareas del enlazador es combinar los programas ensamblados en forma separada en un módulo ejecutable.
3. El último paso es *cargar* el programa para su ejecución. Ya que el cargador conoce en dónde está el programa a punto de ser cargado, puede completar las direcciones indicadas en el encabezado que estaban incompletas. El cargador desecha el encabezado y crea un PSP inmediatamente antes del programa cargado en memoria.

La figura 5-1 proporciona un diagrama de los pasos implicados al ensamblar, enlazar y ejecutar un programa.

## CÓMO ENSAMBLAR UN PROGRAMA FUENTE

El programa ensamblador de Microsoft (hasta la versión 5.x) es MASM.EXE, mientras que el programa de Borland es TASM.EXE. El ensamblador de Microsoft por lo general utiliza el comando ML, pero también acepta MASM por compatibilidad con versiones anteriores.

Puede teclear el comando para ejecutar MASM o TASM en una línea de comando o por medio de peticiones. Esta sección muestra cómo utilizar la línea de comando; véase en el apéndice D el método con indicación. El formato general para un comando de línea para ensamblar un programa es:

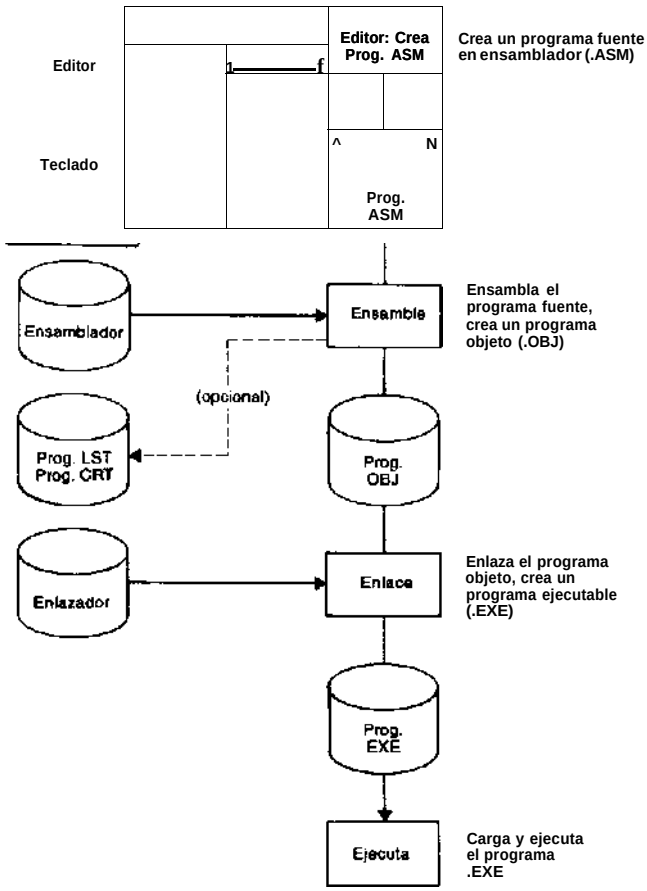



Figura 5-1 Pasos para ensamblar, enlazar y ejecutar

```
MASM/TASM [opciones] fuente [,objeto] [,listado] [,refcruzadas]
```

- *Opciones* estipula características como configuración del nivel de mensajes de advertencia y se explican en el apéndice D. Ya que los valores por omisión del ensamblador por lo regular son los adecuados, rara vez necesitará utilizar opciones.
- *Fuente* identifica el nombre del programa fuente, como P05ASM1. El ensamblador asume la extensión .ASM, de modo que no necesita introducirla. Si no quiere aceptar la unidad de disco por omisión, también puede dar la especificación de una unidad de disco.
- *Objeto* estipula un archivo .OBJ generado. La unidad, subdirectorio y nombre de archivo puede ser el mismo o diferente del fuente.
- *Listado* estipula un archivo .LST generado que contiene tanto el código fuente como el código objeto. La unidad, subdirectorio y nombre de archivo puede ser el mismo o diferente del fuente.

- *Refcruzadas* genera un archivo de referencias cruzadas con los símbolos usados en el programa, que puede usar para un listado de referencias cruzadas. Para MASM, la extensión es .CRF y para TASM la extensión es .XRF. La unidad, subdirectorio y nombre de archivo puede ser el mismo o diferente del fuente.

El nombre del archivo fuente siempre lo debe introducir, y por lo general solicita un archivo .OBJ, que es necesario para enlazar un programa en forma ejecutable. Tal vez en algunas ocasiones solicitará archivos .LST, en especial cuando quiera examinar el código de máquina generado. Un archivo .CRF es útil para programas grandes en donde quiera ver qué instrucciones hacen referencia a qué datos. También la petición de un .CRF hace que el ensamblador genere números de líneas para los enunciados en el archivo .LST a las cuales el archivo .CRF se refiera. Secciones posteriores cubren en detalle los archivos .LST y .CRF.

Ejemplo 1: Especifique el archivo fuente, P05ASM1, en la unidad D y genere archivos objeto, de listado y de referencias cruzadas. Si el nombre de un archivo es el mismo que el del archivo fuente, no necesita repetirlo; basta con la especificación de la unidad para solicitar un archivo:

```
MASM/TASM  D:P05ASM1,D:,C:,D:
```

Ejemplo 2: Sólo genere un archivo objeto. En este caso, puede omitir la referencia a los archivos de listado y de referencias cruzadas: sólo introduzca el comando

```
MASM/TASM  A:P05ASM1,D:
```

El ensamblador convierte sus enunciados fuente en código de máquina y despliega, si hay, errores en la pantalla. Los errores comunes incluyen un nombre que viola las convenciones para la formación de nombres, una operación que se escribió de forma incorrecta (como MOVE en lugar de MOV) y un operando con un nombre que no está definido. Existen alrededor de 100 mensajes de error, explicados en el manual del ensamblador. Ya que hay muchas versiones diferentes de ensamblador, no trataremos de listar los errores. El ensamblador intenta corregir algunos errores, pero de cualquier forma usted debe volver a cargar su editor, corregir el programa fuente .ASM y reensamblarlo.

## LISTADO DEL ENSAMBLADOR DE LAS DEFINICIONES CONVENCIONALES DE SEGMENTOS

La figura 5-2 proporciona el listado que produce el ensamblador con el nombre P05ASM1.LST. Por la entrada PAGE, el ancho de la línea es de 132 posiciones. Si su impresora puede comprimir la línea de impresión, entonces también puede imprimir este listado. Muchas impresoras de impacto tienen un interruptor que fuerza la impresión comprimida, o podría solicitar al editor o procesador de textos imprimir en modo comprimido. Otra manera es usar el comando MODE del DOS; encienda su impresora, teclee el comando MODE LPT1:132,6 para 132 caracteres por línea y seis líneas por pulgada y utilice PRINT del DOS.

Note cómo han actuado las directivas PAGE y TITLE en la parte superior del listado. Ninguna de las directivas, incluyendo SEGMENT, PROC, ASSUME y END, generan código de máquina, ya que sólo son mensajes al ensamblador.

En el extremo izquierdo está el número de cada línea. La segunda columna muestra, en hexadecimal, las direcciones de los campos de datos y de las instrucciones. La tercera columna muestra el código de máquina traducido en formato hexadecimal. A la derecha se encuentra el código fuente original.

```
P05ASM1  (EXE)   Operaciones de mover y sumar                Page     1-1

 1                                page 60,132
 2                    TITLE       P05ASM1  (EXE)  Operaciones de mover y sumar
 3
 4  0000              STACKSG     SEGMENT  PARA STACK 'Stack'
 5  0000   0020[                  DW       32 DUP(0)
 6         0000
 7             ]

 9  0040              STACKSG     ENDS
10
11  0000              DATASG      SEGMENT  PARA
12  0000   00FA        FLDA        DW       250
13  0002   007D        FLDB        DW       125
14  0004   0000        FLDC        DW       ?
15  0006               DATASG      ENDS
16
17  0000               CODESG      SEGMENT PARA 'Code'
18  0000               BEGIN       PROC    FAR
19                                 ASSUME  SS:STACKSG,DS:DATASG,CS:CODESG
20  0000   B8_______R              MOV     AX,DATASG     Establecer la dirección de
21  0003   8E D8                   MOV     DS,AX           DATASG en el registro DS
22
23  0005   Al 0000 R               MOV     AX,FLDA       Mover 0250 a AX
24  0008   03 06 0002              ADD     AX, FLDB      Sumar 0125 a AX
25  OOOC   A3 0004 R               MOV     FLDC, AX      Almacenar suma en FLDC
26  OOOF   B8 4C00                 MOV     AX,4C00H      Salida a DOS
27  0012   CD 21                   INT     21H
28  0014               BEGIN       ENDP                  Fin de procedimiento
29  0014               CODESG      ENDS                  Fin de segmento
30                                 END     BEGIN         Fin de programa


Segments and Groups:
          Ñ a m e                Length    Align     Combine     Class
CODF.SG. . . . . .               .0014     PARA       NONE       'CODE'
DATASG. . . . . .                .0006     PARA       NONE       'DATA'
STACKSG. . . . . .               .0040     PARA       STACK      'STACK'

Symbols:
          Ñ a m e                 Type      Valué      Attr
BEGIN. . . . . .                .F PROC   0000       CODESG     Length = 0014

FLDA. . . . . .                 .L WORD   0000       DATASG
FLDB. . . . . .                 .L WORD   0002       DATASG
FLDC. . . . . .                 .L WORD   0004       DATASG

©CPU. . . . . .                 .TEXT     OlOlh
©FILENAME. . . . . .            .TEXT     p05asml
©VERSIÓN. . . . . .             .TEXT     510

     27 Source   Lines
     27 Total    Lines
     15 Symbols
0 Warning Errors
0 Severe  Errors
```

Figura 5-2 Programa ensamblado con segmentos convencionales

Para cada uno de los tres segmentos, la directiva SEGMENT avisa al ensamblador alinee el segmento a una dirección que sea divisible entre 10 hex —el enunciado mismo no genera código de máquina. De forma teórica, cada dirección de segmento inicia en la localidad con desplaza-

miento 0000. En realidad, cuando el programa inicia su ejecución, el segmento es almacenado en memoria de acuerdo con una dirección que el DOS carga en el registro del segmento y es desplazado cero bytes a partir de esa dirección.

Note que la pila, el segmento de datos y el segmento de código son áreas separadas, cada una con su característico valor de desplazamiento para datos e instrucciones.

### Segmento de la pila

El segmento de la pila contiene una directiva DW (definir palabra) que define 32 palabras, que genera cada una un valor cero designado con (0). Esta definición de 32 palabras es un tamaño realista para una pila, ya que un programa grande puede necesitar muchas interrupciones para llamadas de entrada/salida a subprogramas, y todas implican el uso de la pila. El segmento de la pila termina en el desplazamiento 0040H, que es el equivalente al valor decimal 64 (32 palabras x 2 bytes).

Si el tamaño de la pila es demasiado pequeño para contener a todos los elementos que se guardan en ella, ni el ensamblador ni el enlazador le advertirán de esto, y la ejecución del programa puede sufrir una detención total de una manera impredecible.

### Segmento de datos

El programa define un segmento de datos, DATASG, con tres valores definidos, todos en formato DW (definir palabra). FLDA define una palabra (dos bytes) inicializada con el valor decimal 250, que el ensamblador traduce a 00FAH (mostrado a la izquierda). FLDB define una palabra inicializada con el valor decimal 125, ensamblada como 007DH. Los valores reales almacenados de estas dos constantes son FA00 y 7D00, respectivamente, lo cual puede verificar con DEBUG.

FLDC es codificada como una DW con ? en el operando para definir una palabra con una constante no inicializada.

### Segmento de código

El programa define un segmento de código, CODESG, que contiene el código del programa ejecutable, todo en un procedimiento (PROC).

Tres enunciados establecen el direccionamiento del segmento de datos:

```
                        ASUME  SS:STACKSG,DS:DATASG,CS;CODESG
0000  B8_____R          MOV AX, DATASDG
0003  8E D8             MOV DS, AX
```

- La directiva ASSUME relaciona DATASG con el registro DS. Note que el programa no requiere el registro ES, pero como práctica usual, algunos programadores lo definen. ASSUME sólo proporciona información al ensamblador, lo que no genera código de máquina.
- La primera instrucción MOV "almacena" DATASG en el registro AX. Ahora bien, en realidad una instrucción no puede almacenar un segmento en un registro —el ensamblador sólo reconoce un intento de cargar la dirección de DATASG. Observe el código de máquina a la izquierda: B8____R. Los cuatro guiones significan que en este punto el ensamblador no puede determinar la dirección de DATASG; el sistema determina esta dirección sólo cuando el programa objeto está enlazado y cargado para su ejecución. Ya que el cargador del sistema puede ubicar un programa en cualquier parte de la memoria, el ensamblador deja abierta la

dirección e indica este hecho con una R; el programa cargador DOS es para reemplazar (o reubicar) las direcciones incompletas con las reales.

- La segunda instrucción MOV mueve el contenido del registro AX al registro DS. Ya que no existe una instrucción válida para mover de forma directa de la memoria al registro DS, tiene que codificar dos instrucciones para inicializar el DS.

El cargador DOS inicializa de forma automática el SS y el CS cuando carga un programa para ejecución, pero es su responsabilidad inicializar el DS y, si se necesita, el ES.

Para las directivas simplificadas de segmentos, la inicialización del DS es como sigue:

```
MOV  AX,©datos
MOV  DS,AX
```

Aunque todo estas acciones parecen ser demasiado complicadas, en este momento en realidad no tiene que entenderlo. Todos los programas en este libro utilizan una definición e inicialización estándar, y usted sólo tiene que reproducir el código para cada uno de sus programas. Para este fin, almacene en disco una estructura de un programa ensamblado, y para cada programa nuevo que quiera crear, COPIE la estructura del programa en un archivo con su nombre correcto y use su editor para completar las instrucciones adicionales.

La primera instrucción después de inicializar el DS es MOV AX,FLDA, que empieza en la localidad con desplazamiento 0005 y genera el código de máquina Al 0000. El espacio entre Al (la operación) y 0000 (el operando) es sólo por legibilidad. La instrucción siguiente es ADD AX,FLDB que empieza en la localidad con desplazamiento 0008 y genera cuatro bytes de código de máquina. En este ejemplo, la longitud de las instrucciones de máquina son dos, tres y cuatro bytes.

El último enunciado en el programa, END, contiene el operando BEGIN, que relaciona al nombre del PROC en el desplazamiento 0000. Ésta es la localidad en el segmento de código a donde el cargador de programa transfiere el control para la ejecución.

A continuación del listado del programa están una tabla de segmentos y grupos y una tabla de símbolos.

### Tabla de segmentos y grupos

La primera tabla al final del listado del ensamblador muestra todos los grupos y segmentos definidos. Note que los segmentos no están listados en el mismo orden en que fueron codificados; el ensamblador los lista en orden alfabético por nombre (este programa no tiene grupos, que es un tema posterior). La tabla proporciona la longitud en bytes de cada segmento, la alineación (ambos son párrafos), el tipo combinar y la clase. El ensamblador ha convertido los nombres de clase a mayúsculas.

### Tabla de símbolos

La segunda tabla proporciona los nombres de los campos de datos en el segmento de datos (FLDA, FLDB y FLDC) y las etiquetas aplicadas a instrucciones en el segmento de código. Para BEGIN (la única entrada en el ejemplo), Type F PROC significa procedimiento lejano. La columna *valué* da el desplazamiento para el inicio del segmento de nombres, etiquetas y procedimientos. La columna encabezada con Attr (atributo) proporciona el segmento en el que el elemento está definido.

El apéndice D explica todas las opciones de estas tablas. Para que el ensamblador omita las tablas, codifique la opción /N después del comando MASM, esto es, MASM/N.

En cuanto a las últimas tres entradas, @CPU identifica al procesador, ©FILENAME da el nombre del programa y ©VERSIÓN muestra la versión del ensamblador en la forma n.nn.

## LISTADO ENSAMBLADOR DE DIRECTIVAS SIMPLIFICADAS DE SEGMENTOS

La figura 4-3 mostró cómo codificar un programa que usa las directivas simplificadas de segmentos. La figura 5-3 proporciona el listado ensamblado de ese programa. La primera parte de la tabla de símbolos bajo "Segments and Groups" muestra los tres segmentos renombrados por el ensamblador y listados de forma alfabética:

- _DATA, con una longitud de 6 bytes
- STACK, con una longitud de 40H (64 bytes)
- _TEXT, para el segmento de código, con una longitud de 14H (20 bytes)

Bajo el título "Symbols" hay nombres definidos en el programa o nombres por omisión. Las directivas simplificadas de segmentos proporcionan varias equivalencias predefinidas, que empiezan con el símbolo © y que usted tiene libertad de referenciar en un programa. Igual que ©datos, ellos son:

| | |
|---|---|
| ©CODE | Igualada al nombre del segmento de código TEXT |
| ©CODESIZE | Establece a cero para los modelos pequeño y mediano |
| ©CPU | Modelo de procesador |
| ©DATASIZE | Establece a cero para los modelos pequeño y mediano |
| ©FILENAME | Nombre del programa |
| ©VERSIÓN | Versión del ensamblador (n.nn) |

Puede usar ©código y ©datos en enunciados ASSUME y ejecutables, tal como MOV AX, ©datos.

## ENSAMBLADOR DE DOS PASADAS

Muchos ensambladores dan dos pasadas al programa fuente a fin de resolver referencias hacia adelante (o posteriores) a direcciones que aún no se encuentran en el programa. Durante la pasada 1, el ensamblador lee todo el código fuente y construye una tabla de símbolos de nombres y etiquetas usadas en el programa, esto es, nombres de campos de datos y etiquetas del programa y sus localidades relativas (desplazamiento) dentro del segmento. Usted puede ver tal tabla de símbolos a continuación del programa ensamblado en la figura 5-3, en donde los desplazamientos de FLDA, FLDB y FLDC son 0000, 0002 y 0004 bytes, respectivamente. Aunque el programa no define etiquetas de instrucciones, ellas aparecerían en el segmento de código con sus propios desplazamientos. La pasada 1 determina la cantidad de código que es generado por cada instrucción. MASM inicia la generación del código objeto en la pasada 1, mientras que TASM lo hace en la pasada 2.

Durante la pasada 2, el ensamblador usa la tabla de símbolos que construyó en la pasada 1. Ahora que "conoce" la longitud y posiciones relativas de cada campo de datos e instrucción,

```
05ASM2  (EXE)  Operaciones de mover y sumar                Page     l-1

                        page   60,132
               TITLE    P05ASM2  (EXE)  Operaciones de mover y sumar

                        .MODEL  SMALL
                        .STACK  64          ;Se define la pila
                        .DATA               ;Se definen datos
0000   OOFA     FLDA    DW      250
0002   007D     FLDB    DW      125
0004   0000     FLDC    DW      ?

                        .CODE               ;Se define segmento de código
0000            BEGIN   PROC    FAR
0000   B8 _____R        MOV     AX,@data    ;Establecer la dirección de
0003   8E D8            MOV     DS,AX       ;  DATASG en el registro DS

0005   Al 0000 R        MOV     AX,FLDA     ,-Mover 0250 a AX
0008   03 06 0002 R     ADD     AX,FLDB     ;Sumar 0125 a AX
OOOC   A3 0004 R        MOV     FLDC,AX     ,-Almacenar suma en FLDC

OOOF   B8 4C00          MOV     AX,4C00H    ;Salida a DOS
0012   CD 21            INT     21H
0014            BEGIN   ENDP                ,•Fin de procedimiento
                        END     BEGIN       ,•Fin de programa

Segments and Groups:
                Ñame            Length   Align     Combine  Class
DGROUP. . . . . .               .GROUP
  _DATA. . . . . .              .0006    WORD      PUBLIC   'DATA'
  STACK. . . . . .              .0040    PARA      STACK    'STACK•
_TEXT. . . . . .                .0014    WORD      PUBLIC   'CODE'
Symbols:
                Ñame            Type     Valué     Attr
BEGIN. . . . . .                .F PROC  0000      _TEXT    Length = 0014

FLDA. . . . . .                 .L WORD  0000      DATA
FLDB. . . . . .                 .L WORD  0002      DATA
FLDC. . . . . .                 .L WORD  0004      "DATA

©CODE. . . . . .                .TEXT    _TEXT
©CODESIZE. . . . . .            .TEXT    0
©CPU. . . . . .                 .TEXT    Ololh
©DATASIZE. . . . . .            .TEXT    0
@FILEÑAME. . . . . .            .TEXT    p05asm2

      0 Warning Errors
      0 Severe  Errors
```

Figura 5-3 Programa ensamblado con directivas simplificadas de segmentos

puede completar el código objeto para cada instrucción. Después produce, si se solicita, los diferentes archivos objeto (.OBJ), de listado (.LST) y de referencias cruzadas (.REF).

Un problema potencial en la pasada 1 es una *referencia hacia adelante:* Una instrucción de salto en el segmento de código puede referenciar a una etiqueta, pero el ensamblador aún no ha encontrado su definición. MASM construye el código objeto con base en lo que supone es la longitud de cada instrucción generada en lenguaje de máquina. Si existen diferencias entre la pasada 1 y la pasada 2, con respecto a la longitud de una instrucción, MASM envía un mensaje de

error "Phase error between passes". Tales errores son relativamente raros, y si aparecen usted debe buscar su causa y corregirla.

Desde la versión 6.0, MASM hace un manejo más eficaz de la longitud de las instrucciones, dando tantas pasadas al archivo como sean necesarias.

## CÓMO ENLAZAR UN PROGRAMA OBJETO

Una vez que su programa queda sin mensajes de error, el siguiente paso es enlazar el módulo objeto, P05ASM1.OBJ, que fue producido por el ensamblador y que contiene sólo código de máquina. El enlazador realiza las funciones siguientes:

- Si se pide, combina más de un módulo ensamblado de forma separada en un programa ejecutable, como dos o más programas en ensamblador o un programa en ensamblador con un programa en C.
- Genera un módulo .EXE y lo inicializa con instrucciones especiales para facilitar su subsecuente carga para ejecución.

Una vez que ha enlazado uno o más módulos .OBJ en un módulo .EXE, puede ejecutar el módulo .EXE cualquier número de veces. Pero siempre que necesite realizar un cambio al programa, debe corregir el programa fuente, ensamblarlo en otro módulo .OBJ y enlazar el módulo .OBJ en un módulo .EXE. Aunque al principio estos pasos no sean por completo claros, encontrará que con un poco de experiencia se vuelven automáticos.

Puede convertir muchos programas .EXE a programas .COM. Para detalles, véase el capítulo 7.

La versión del enlazador de Microsoft es LINK, mientras que la de Borland es TLINK. Puede teclear LINK o TLINK en una línea de comando o por medio de peticiones (a partir de MASM 6.0, el comando ML proporciona tanto el ensamble como el enlace). Esta sección muestra cómo enlazar usando la línea de comando; para el uso de peticiones véase el apéndice D. La línea de comando para enlazar es

```
LINK/TLINK archobj, archeje, [,archmapa] [,archbibl]
```

- *Archobj* identifica al archivo objeto generado por el ensamblador. El enlazador supone la extensión .OBJ, de modo que no tiene que introducirla. Unidad, subdirectorio y nombre de archivo pueden ser iguales o diferentes del archivo fuente.
- *Archeje* estipula que se genere un archivo .EXE. Unidad, subdirectorio y nombre de archivo pueden ser iguales o diferentes del archivo fuente.
- *Archmapa* estipula que se genere un archivo con extensión .MAP que indica la ubicación relativa y el tamaño de cada segmento y cualquier error que LINK haya encontrado. Un error común es el fallo al definir un segmento de pila. Introducir CON (por consola) le indica al enlazador que muestre el mapa en la pantalla (en lugar de escribirlo en un disco) de forma que se pueda ver el mapa inmediatamente para los errores.
- *Archbibl* estipula la opción de bibliotecas, que no necesita en estos primeros pasos de programación en lenguaje ensamblador.

Este ejemplo enlaza el archivo objeto P05ASM1.OBJ que fue generado por un ensamble anterior. Al enlazador se le pide escribir el archivo .EXE en la unidad D, desplegar el mapa e ignorar la opción de biblioteca:

```
LINK D:P05ASM1,D:,CON
```

Si el nombre del archivo es el mismo que el del fuente, no necesita repetirlo: basta con la identificación de la unidad para indicar la petición del archivo. El apéndice D porporciona otras opciones.

### Mapa del enlace para el primer programa

Para el programa P05ASM1, LINK produce este mapa:

| START | STOP | LENGTH | ÑAME | CLASS |
|---|---|---|---|---|
| OOOOOH | 0003FH | 0040H | STACKSG | STACK |
| 00040H | 00045H | 0006H | DATASG | DATA |
| 00050H | 00063H | 0014H | CODESG | CODE |

Punto de entrada del programa en 0005:0000

- La pila es el primer segmento e inicia con un desplazamiento de cero bytes desde el inicio del programa. Como está definida como 32 palabras, es de 64 bytes, como lo indica su longitud (40H).
- El segmento de datos inicia en el siguiente límite de párrafo, desplazamiento 40H.
- El segmento de código inicia en el siguiente límite de párrafo, desplazamiento 50H. Algunos ensambladores acomodan los segmentos en orden alfabético.
- El punto de entrada al programa es 0005:0000, que está en la forma "relativa (no absoluta) segmento desplazamiento", se refiere a la dirección de la primera instrucción ejecutable. En realidad, la dirección relativa de inicio es en el segmento 5[0], desplazamiento de 0 bytes, que corresponde al límite del segmento en 50H. El programa cargador utiliza este valor cuando carga el programa en memoria para ejecución.

En **esta** etapa el único error que puede encontrar es introducir de manera errónea los nombres de los archivos. La solución es reiniciar el comando de enlace.

### Mapa del enlace para el segundo programa

El mapa de enlace para el segundo programa, que utiliza las directivas simplificadas de segmentos, muestra una configuración un poco diferente a la del programa anterior. Primero, el ensamblador ha reacomodado de manera física los segmentos en orden alfabético, y segundo, los segmentos sucesivos están alineados por límites de palabras (no de párrafo):

| START | STOP | LENGTH | ÑAME | CLASS |
|---|---|---|---|---|
| OOOOOH | 00013H | 0014H | _TEXT | CODE |
| 00014H | 00019H | 0006H | _DATA | DATA |
| 00020H | 0005FH | 0040H | STACK | STACK |

Punto de entrada del programa en 0000:0000

- El segmento de código ahora es el primer segmento e inicia en un desplazamiento de cero bytes desde el inicio del programa.
- El segmento de datos inicia en el siguiente límite de palabra, desplazamiento 14H.
- La pila inicia en el siguiente límite de palabra, desplazamiento 20H.
- El punto de entrada al programa ahora es 0000:0000, lo cual significa que la ubicación relativa del segmento de código inicia en el segmento 0, desplazamiento 0.

## CÓMO EJECUTAR UN PROGRAMA

Una vez ensamblado y enlazado un programa, ahora puede (¡al fin!) ejecutarlo. Si el archivo .EXE está en la unidad por omisión, podría usar el DOS para cargarlo para su ejecución introduciendo:

```
P05ASM1.EXE  o  P05ASM1
```

Si omite la extensión del archivo, el DOS supone que es .EXE (o .COM). Sin embargo, ya que este programa no produce resultados visibles, se sugiere que lo ejecute con DEBUG y avance paso por paso en su ejecución con comandos de rastreo (T). Teclee lo siguiente, incluyendo la extensión .EXE:

```
DEBUG  D:P05ASM1.EXE
```

DEBUG carga el módulo del programa .EXE y muestra su indicación (un guión). Para ver el segmento de la pila, teclee

```
D SS : 0
```

La pila contiene sólo ceros ya que fue la forma de inicializarla. Para ver el segmento de datos, teclee

```
D DS : 0
```

La operación muestra tres elementos de datos FA 00 7D 00 00 00, con los bytes de cada palabra en orden inverso. Para ver el segmento de código, teclee

```
D CS : 0
```

Compare el código de máquina mostrado con el del segmento de código en el listado del ensamblado:

```
B8____8ED8A10000 . . .
```

En este caso, el listado del ensamblado no muestra de manera precisa el código de máquina, ya que el ensamblador no conoce la dirección del operando de la primera instrucción. Ahora puede determinar esta dirección examinando el código desplegado.

Teclee **R** para ver los registros, y rastree la ejecución del programa con sucesivos comandos T. A medida que avance por el programa, fíjese en el contenido de los registros. Cuando llegue a la última instrucción, puede utilizar L para volver a cargar y correr el programa o Q para salir de la sesión con DEBUG.

## LISTADO DE REFERENCIAS CRUZADAS

El ensamblador genera un archivo opcional .CRF o .XRF que puede usar para producir un *listado de referencias cruzadas* de los identificadores o símbolos del programa. Sin embargo, aún tiene usted que convertir este archivo a un archivo de referencias cruzadas, ordenado de manera adecuada. Esta función la realiza un programa en el disco del ensamblador: CREF para Microsoft o TCREF para Borland. Puede teclear CREF o TCREF con una línea de comando o por medio de indicaciones. Esta sección utiliza una línea de comando; véase el apéndice D para usar indicaciones. El comando para convertir el archivo de referencias cruzadas es

```
CREF/TCREF archivoxref, archivoref
```

- *archivorefx* identifica el archivo de referencias cruzadas generado por el ensamblador. El programa supone la extensión, así que no necesita introducirla. También puede dar una identificación de la unidad de disco.
- *archivoref* estipula que se genere un archivo .REF. Unidad, subdirectorio y nombre de archivo pueden ser iguales o diferentes del archivo fuente.

### El listado

La figura 5-4 contiene el listado de referencias cruzadas producido por CREF para el programa de la figura 5-2. Los símbolos en la primera columna están en orden alfabético. Los números en la segunda columna, mostrados como n#, indican la línea en que están definidos los símbolos en el archivo .LST. Los números a la derecha de esta columna son los números de línea en donde los símbolos están referenciados. Por ejemplo, CODESG está definido en la línea 17 y se hace referencia a él en las líneas 19 y 29. FLDC está definido en la línea 14 y referenciado en la línea 25 +, en donde " + " significa que su valor es modificado en esta línea.

```
P04ASM1  (EXE)   Operaciones de mover y sumar

Symbol Cross-Reference    (# definition,  + modification)

©CPU . . . . . .  1#
SVERSION . . . . . .  1#

BEGIN . . . . . .  18#     28      30

CODE . . . . . .
CODESG . . . . . .  17#     19      29

DATA . . . . . .
DATASG . . . . . .  11#     15      19

FLDA . . . . . .  12#     23
FLDB . . . . . .  13#     24
FLDC . . . . . .  14#     25 +

STACK . . . . . .
. . . . .  4#      9      19

 12 Symbols
```

Figura **5-4** Tabla de referencias cruzadas

### Archivos generados

Al ensamblar varios programas puede usar mucho espacio en disco. Es posible, de manera segura, borrar los archivos .OBJ, .CRF y .LST. Guarde los programas fuente .ASM en caso de cambios futuros y también guarde los archivos .EXE para la ejecución del programa.

## DIAGNÓSTICO DE ERRORES

El ensamblador proporciona un diagnóstico de cualquier error de programación que viole sus reglas. El programa en la figura 5-5 es el mismo que el de la figura 5-2, salvo que tiene insertados varios errores intencionales con fines ilustrativos. El programa fue corrido con MASM; TASM genera un listado parecido de errores. Aquí están los errores, como se codificaron:

| LÍNEA | EXPLICACIÓN |
|---|---|
| 14 | FLDC necesita un operando. |
| 19 | ASSUME no relaciona el SS a STACKSG, aunque el ensamblador no ha detectado esta omisión. |
| 20 | DATSEG debe ser escrito como DATASG. |

```
                                page 60,132
                        TITLE   P05ASM3  (EXE)  Ilustra errores de ensamblado

   0000                 STACKSG SEGMENT PARA STACK  'Stack'
   0000   0020[                 DW      32 DUP(0)
               0000
                    ]

 9 0040                 STACKSG ENDS
10
11 0000                 DATASG  SEGMENT PARA  'Data'
12 0000   00FA          FLDA    DW      250
13 0002   007D          FLDB    DW      125
14 0004                 FLDC    DW
p05asm3.ASM(11)  error A2027: Operand  expected
15 0004                 DATASG  ENDS
16
17 0000                 CODESG  SEGMENT PARA 'Code'
18 0000                 BEGIN   PROC    FAR
19                              ASSUME  CS:CODESG,DS:DATASG
20 0000  Al 0000 U              MOV     AX,DATSEG         ;Dirección de DATASG
p05asm3.ASM(17): error A2009: Symbol not defined:  DATSEG
21 0003  8B DO                  MOV     DX,AX               en el registro DS
22
23                              MOV     AS,FLDA           ;Mover 0250 a AX
p05asm3.ASM(20): error A2009: Symbol not defined:  AS
24 0005  03 06 0002 R           ADD     AX,FLDB           ;Sumar 0125 a AX
25 0009  A3 0000 U              MOV     FLDD,AX            Almacenar suma en FLDC
p05asm3 ASM(22): error A2009: Symbol not defined:  FLDD
26 000C  B8 4C00                MOV     AX,4C00H          ;Salida a DOS
27 000F  CD 21                  INT     21H
28 0011                 BEGIN   ENDP
p05asm3 ASM(25): error A2006: Phase error between passes
29 0011                 CODESG  ENDS
30                              END     BEGIN
```

**Figura 5-5** Diagnóstico del ensamblado

| | |
|---|---|
| 21 | DX debe ser codificado como DS, aunque el ensamblador no sabe que éste es un error. |
| 23 | AS debe se codificado como AX. |
| 25 | FLDD debe se codificado como FLDC. |
| 28 | La corrección de los otros errores hará que este diagnóstico desaparezca. |

El último mensaje de error, "Phase error between passes", ocurre cuando las direcciones generadas en la pasada 1 difieren de aquellas en la pasada 2 en un ensamblador de dos pasadas. Para aislar un error desconocido, utilice la opción /D para que MASM liste un archivo para la pasada 1 y otro archivo para la pasada 2, y compare los desplazamientos.

## PUNTOS CLAVE

- MASM y TASM proporcionan una línea de comando para ensamblar, incluyendo (al menos) el nombre del programa fuente. MASM también proporciona indicaciones para introducir opciones.
- El ensamblador convierte un programa fuente a un archivo .OBJ y genera archivos opcionales para el listado y las referencias cruzadas.
- La tabla de segmentos y grupos que sigue a un listado de ensamblador muestra los segmentos y grupos definidos en el programa. La tabla de símbolos muestra todos los símbolos (nombres de datos y etiquetas de instrucción).
- El enlazador (LINK o TLINK) convierte un archivo .OBJ en un archivo .EXE. Usted puede enlazar usando una línea de comando o por medio de indicaciones (sólo LINK).
- Las directivas simplificadas de segmentos generan los nombres _DATA para el segmento de datos, STACK para el segmento de la pila y TEXT para el segmento de código. También generan varias equivalencias predefinidas.
- El programa CREF (o TCREF) produce un útil listado de referencias cruzadas.

## PREGUNTAS

5-1. Codifique la línea de comandos para ensamblar el programa fuente llamado DISCOUNT.ASM con archivos .LST, .OBJ y .CRF. Suponga que el programa fuente y el ensamblador están en la unidad C.

5-2. Codifique la línea de comando en LINK o TLINK para enlazar DISCOUNT.OBJ de la pregunta 5-1.

5-3. Codifique los comandos para DISCOUNT.EXE de la pregunta 5-2 para hacer lo siguiente: (a) ejecución por medio de DEBUG; (b) ejecución directa desde el DOS.

5-**4.** Dar el objetivo de cada uno de los archivos siguientes: (a) archivo .ASM; (b) archivo .CRF; (c) archivo .LST; (d) archivo .EXE; (e) archivo .OBJ; (f) archivo .MAP.

5-5. Codifique las dos instrucciones para inicializar el registro DS. Suponga que el nombre del segmento de datos es DATSEG.

5-6. Escriba un programa en ensamblador usando las definiciones convencionales de segmentos para lo siguiente: (a) Mover el valor inmediato 40 hex al registro AL; (b) recorrer el contenido de AL un bit hacia la izquierda (código SHL AL,1); (c) mover el valor inmediato 22 hex al BL; (d) multiplicar AL por BL (código MUL BL). Recuerde las instrucciones necesarias para finalizar la ejecución de un programa. El programa no necesita definir o inicializar el segmento de datos. Asegúrese de COPIAR una estructura de programa y utilice su editor para desarrollar el programa. Ensámblelo y enlácelo. Utilice DEBUG para rastrear y verificar el segmento de código y los registros.

**5-7.** Corrija el programa de la pregunta 5-6 para directivas simplificadas de segmentos. Ensámblelo y enlácelo, y compare el código objeto, las tablas de símbolos y el mapa de enlace con aquellos del programa original.

**5-8.** Agregue un segmento de datos al programa de la pregunta 5-6, para lo siguiente:

- Defina un elemento de un byte (DB) llamado FIELDA con 40 hex y otro con nombre FIELDB con 22 hex.
- Defina un elemento de dos bytes (DW) con nombre FIELDC sin constante.
- Mueva el contenido de FIELDA al registro AL, y recórralo un bit a la izquierda.
- Multiplique el AL por FIELDB (código MUL FIELDB).
- Mueva el producto en el AX a FIELDC.

Ensamble, enlace y utilice DEBUG para probar el programa.

**5-9.** Corrija el programa de la pregunta 5-8 para directivas simplificadas de segmentos. Ensámblelo y enlácelo, y compare el código objeto, las tablas de símbolos y el mapa de enlace con aquellos del programa original.

## CAPÍTULO 6

# Instrucciones y direccionamiento del procesador

OBJETIVO

**Proporcionar los fundamentos del conjunto de instrucciones de lenguaje ensamblador y los requisitos para el direccionamiento de datos.**

### INTRODUCCIÓN

Este capítulo introduce el conjunto de instrucciones del procesador y enseguida describe los formatos básicos de direccionamiento que son usados en el resto del libro. Formalmente, las instrucciones que se tratan en este capítulo son MOV, MOVSX, MOVZX, XCHNG, LEA, INC, DEC e INT. También se puede definir como un valor inmediato una constante en el operando de una instrucción.

Por último, el capítulo explica la alineación de dirección y el prefijo que invalida el segmento.

### EL CONJUNTO DE INSTRUCCIONES DEL PROCESADOR

La siguiente es una lista de las instrucciones para la familia de procesadores 8086, clasificadas por categorías. Aunque la lista parece enorme, muchas de las instrucciones rara vez se necesitan.

#### Aritméticas

- ADC: Suma con acarreos
- ADD: Suma números binarios

- DEC: Decrementa en 1
- DIV: División sin signo
- IDIV: Divide con signo (enteros)
- IMUL: Multiplica con signo (enteros)
- INC: Incrementa en 1
- MUL: Multiplica sin signo
- NEG: Negación
- SBB: Resta con el bit prestado
- SUB: Resta valores binarios

**Conversión ASCII-BCD**

- AAA: Ajuste ASCII después de sumar
- AAD: Ajuste ASCII antes de dividir
- AAM: Ajuste ASCII después de multiplicar
- A AS: Ajuste ASCII después de restar
- DA A: Ajuste decimal después de sumar
- DAS: Ajuste decimal después de restar

**Corrimiento de bit**

- RCL: Rota a la izquierda a través del acarreo
- RCR: Rota a la derecha a través del acarreo
- ROL: Rota a la izquierda
- ROR: Rota a la derecha
- SAL: Corrimiento algebraico a la izquierda
- SAR: Corrimiento algebraico a la derecha
- SHL: Corrimiento lógico a la izquierda
- SHR: Corrimiento lógico a la derecha
- SHLD/SHRD: Corrimiento en doble precisión (80386 y posteriores)

**Comparación**

- BSF/BSR: Exploración de bit (80386 y posteriores)
- BT/BTC/BTR/BTS: Prueba bit (80386 y posteriores)
- CMP: Compara
- CMPS: Compara cadenas de caracteres
- TEST: Prueba bits

**Transferencia de datos**

- LDS: Carga el registro del segmento de datos
- LEA: Carga una dirección efectiva

- LES: Carga el registro de segmento extra
- LODS: Carga una cadena
- LSS: Carga el registro del segmento de la pila
- MOV: Mueve datos
- MOVS: Mueve cadenas
- MOVSX: Mueve con signo-extendido
- MOVZX: Mueve con cero-extendido
- STOS: Almacena una cadena
- XCHG: Intercambia
- XLAT: Traduce

**Operaciones con banderas**

- CLC: Limpia la bandera de acarreo
- CLD: Limpia la bandera de dirección
- CLI: Limpia la bandera de interrupción
- CMC: Complementa la bandera de acarreo
- LAHF: Carga AH de las banderas
- POPF: Remueve banderas de la pila
- PUSHF: Agrega banderas a la pila
- SAHF: Almacena el contenido de AH en las banderas
- STC: Establece la bandera de acarreo
- STD: Establece la bandera de dirección
- STI: Establece la bandera de interrupción

**Entrada/Salida**

- IN: Introduce un byte o una palabra
- OUT: Saca un byte o una palabra

**Operaciones lógicas**

- AND: Conjunción lógica (y)
- NOT: Negación lógica (no)
- OR: Disyunción lógica (o)
- XOR: Disyunción exclusiva

**Ciclos**

- LOOP: Repetir el ciclo hasta que se complete
- LOOPE/LOOPZ: Repetir el ciclo mientras sea igual/mientras sea cero
- LOOPNE/LOOPNZ: Repetir el ciclo mientras no sea igual/mientras no sea cero

**El conjunto de instrucciones del procesador**

Control del procesador

- ESC: Escape
- HLT: Introduce un estado de detención
- LOCK: Bloquea el bus
- NOP: No operar
- WAIT: Pone al procesador en estado de espera

Operaciones con la pila

- POP: Remueve una palabra de la pila
- POPA: Remueve todos los registros generales (80286 y posteriores)
- PUSH: Agrega a la pila
- PUSHA: Agrega todos los registros generales (80286 y posteriores)

Operaciones con cadenas

- CMPS: Compara cadenas
- LODS: Carga cadena
- MOVS: Mueve cadena
- REP: Repite una cadena
- REPE/REPZ: Repite mientras sea igual/mientras sea cero
- REPNE/REPNZ: Repite mientras no sea igual/mientras no sea cero
- SCAS: Explora una cadena
- STOS: Almacena una cadena

Transferencia (condicional)

- INTO: Interrumpe si hay desbordamiento
- JA/JNBE: Bifurca (salta) si es mayor o salta si no es menor o igual
- JAE/JNB: Salta si es mayor o igual o salta si no es menor
- JB/JNAE: Salta si es menor o salta si no es mayor o igual
- JBE/JNA: Salta si es menor o igual o salta si no es mayor
- JC/JNC: Salta si hay acarreo o salta si no hay acarreo
- JCXZ: Salta si CX es cero
- JE/JZ: Salta si es igual o salta si es cero
- JG/JNLE: Salta si es mayor o salta si no es menor o igual
- JGE/JNL: Salta si es mayor o igual o salta si no es menor
- JL/JNGE: Salta si es menor o salta si no es mayor o igual
- JLE/JNG: Salta si es menor o igual o salta si no es mayor
- JNE/JNZ: Salta si no es igual o salta si no es cero

- JNP/JPO: Salta si no hay paridad o salta si la paridad es impar
- JO/JNO: Salta si hay desbordamiento o salta si no hay desbordamiento
- JP/JPE: Salta si hay paridad o salta si la paridad es par
- JS/JNS: Salta si el signo es negativo o salta si el signo es positivo

**Transferencia (incondicional)**

- CALL: Llama a un procedimiento
- INT: Interrupción
- IRET: Interrupción de regreso
- JMP: Salto incondicional
- RET: Regreso
- RETN/RETF: Regreso cercano o regreso lejano

**Conversión de tipo**

- CBW: Convierte byte a palabra
- CDQ: Convierte palabra doble a palabra cuádruple (80386 y posteriores)
- CWD: Convierte palabra a palabra doble
- CWDE: Convierte una palabra a una palabra doble extendida

## OPERANDOS

Un operando es una fuente de datos para una instrucción. Algunas instrucciones, como CLC y RET, no necesitan un operando, mientras que otras pueden tener uno o dos operandos. Donde existan dos operandos, el segundo es el fuente, que contiene ya sea datos que serán entregados (inmediatos) o bien la dirección (de un registro o en memoria) de los datos. El dato fuente no es cambiado por la operación. El primer operando es el destino, que contiene datos en un registro o en memoria y que será procesado.

```
operación  operandol,  operando2
```

Examinemos ahora cómo los operandos pueden afectar el direccionamiento **de** datos.

### Operandos registro

Para este tipo, el registro proporciona el nombre de alguno de los registros de 8, 16 o 32 bits. Dependiendo de la instrucción, el registro puede codificarse en el primero o segundo operandos, o en ambos:

```
WORD  DW  ?

      MOV  CX,WORDX      ,-Registro en el primer operando
      MOV  WORDX,  BX    ,•Registro en el segundo operando
      MOV  CL,  AH       /Registros en ambos operandos
```

El procesamiento de datos entre registros es el tipo de operación más rápida, ya que no existe referencia a memoria.

**Operandos inmediatos**

En formato inmediato, el segundo operando contiene un valor constante o una expresión constante. El campo destino en el primer operando define la longitud de los datos y puede ser un registro o una localidad de memoria. A continuación se dan algunos ejemplos:

```
SAVE   DB    ?

       ADD   CX,  12    ;Suma 12 al CX
       MOV   SAVE,  25  ;Mueve 25 a SAVE
```

Una sección posterior estudia los operandos con mayor detalle.

**Operandos de memoria directa**

En este formato, uno de los operandos hace referencia a una localidad de memoria y el otro a un registro. Note que no existen instrucciones que permite que ambos operandos sean direcciones de memoria. Para el direccionamiento de datos en memoria, el registro DS es el registro por omisión. Aquí están algunos ejemplos:

```
WORDl  DW 0

BYTE1  DB 0


       MOV AX,WORDl          ;Carga WORDl en AX

       ADD BYTE1,  CL        ;Suma CL a BYTE1

       MOV BX.DS:  [38B0H]   ;Mueve una palabra desde la memoria al desplazamiento 38B0H

       INC BYTE PTR  [2F0H]  /Incrementa el byte en el desplazamiento 2F0H
```

Los últimos dos ejemplos utilizan corchetes como *especifi -adores de índice* para indicar una referencia a memoria (el desplazamiento es combinado con la dirección en el DS). La omisión de los corchetes, como en **MOV** BX,38B0H, indica un valor inmediato: note la gran diferencia.

El último ejemplo incrementa el byte en memoria en el desplazamiento 2F0H (el desplazamiento combinado con la dirección DS). Ya que el operando sólo indica la localidad inicial de memoria, aquí necesitamos el modificador BYTE PTR para definir la longitud.

A continuación, un elemento de dato actúa coma una dirección de desplazamiento en un operando de instrucción:

```
TABLEX  DB   25  DUP(?)

        MOV  AL,TABLEX[4]  /Obtiene el cuarto byte de TABLEX

        MOV  AL,TABLEX+4   ;La misma operación
```

El primer MOV usa un especificador de índice para accesar el cuarto byte de TABLEX. El segundo MOV usa un operador + para tener exactamente el mismo efecto.

**Operandos de memoria indirecta**

*Direccionamiento indirecto* es una técnica sofisticada que hace uso de las capacidades de la computadora para el direccionamiento de segmento:desplazamiento. Los registros utilizados para este propósito son BX, DI, SI y BP, codificados con corchetes como un operador de índice. BX, DI y SI están asociados con el registro DS como DS:BX, DS:DI y DS:SI, para procesamiento de datos en el segmento de datos. El BX, DI y SI están asociados con el registro DS como DS:BX, DS:DI y DS:SI para procesamiento de datos en el segmento de datos. El BP está asociado con el registro SS como SS:BP, para manejo de datos en la pila, lo cual haremos en el capítulo 23 cuando llamemos subprogramas y pasemos parámetros.

Cuando el primer operando contiene una dirección indirecta, el segundo se refiere a un registro o a un valor inmediato; cuando el segundo operando contiene una dirección indirecta, el primero se refiere a un registro. Una dirección indirecta tal como [BX] le indica al ensamblador que la dirección de memoria a usar estará en el registro BX cuando el programa la ejecute posteriormente.

En el ejemplo siguiente, el primer MOV inicializa el BX con la dirección con desplazamiento de DATAFLD. El segundo MOV utiliza la dirección en el BX para almacenar cero en la localidad de memoria a la cual apunta, en este caso, DATAFLD:

```
DATAFLD DB    ?


        MOV   BX,OFFSET DATAFLD ;Carga BX con el desplazamiento
        MOV   [BX] ,  0         ,-Mueve 0 a DATAFLD
```

El efecto de los dos MOV es el mismo que codificar MOV DATAFLD,0, aunque el uso de direccionamiento indexado por lo común no es tan trivial. La siguiente instrucción mueve cero a la localidad que se encuentra dos bytes después de DATAFLD:

```
        MOV   [BX+2],0          ,-Mueve 0 a DATAFLD+2
```

También puede combinar registros en un direccionamiento indirecto. Así [BX+SI] significa la dirección en BX más la dirección en el SI.

Note que cualquier referencia encorchetes a los registros BX, DI, SI o BP implican un operando indirecto, y el sistema trata los contenidos de los registros como una desplazamiento de dirección. A continuación están algunos ejemplos más:

```
MOV BL,  [BX]                   ;DS:BX
SUB BYTE PTR [DI],  [SI]   ,-DS:DI y DS:SI
MOV [BP] ,AL                    ;SS:BP
```

**Desplazamiento de dirección.** Este método utiliza un desplazamiento de dirección para un operando. El código siguiente mueve el contenido del CL a TABLEX (una tabla 26 bytes); exactamente en donde TABLEX está determinada por el contenido de DI cuando el programa está en ejecución:

```
TABLEX DB   25 DUP(?)

MOV  TABLEX[DI], CL
```

**Indexación en el 80386** y **procesadores posteriores.** Estos procesadores permiten una dirección que sea generada a partir de cualquier combinación de uno o más registros generales, un desplazamiento y un factor de escala (1, 2, 4 u **8)** asociado con el contenido de uno de los registros. Por ejemplo, la instrucción

```
MOV EBX,  [ECX*2+ESP+4]
```

mueve una dirección al EBX, la dirección consiste en el contenido de (el ECX por 2) más el contenido de (el ESP más 4).

## LA INSTRUCCIÓN MOV

La instrucción MOV transfiere (esto es, copia) los datos referenciados por la dirección del segundo operando a la dirección del primer operando. El campo que se envía permanece sin cambios. Los operandos que hacen referencia a memoria o registros deben coincidir en tamaño (es decir, ambos deben ser bytes, ambos deben ser palabras o ambos deben ser palabras dobles). El formato general para MOV es

```
[etiqueta:]  MOV  {registro/memoria},{registro/memoria/inmediato}
```

Aquí están cuatro ejemplos de operaciones MOV válidas, por categorías, dados los siguientes elementos de datos:

```
BYTEVAL  DB  ?
WORDVAL  DW  ?
```

**1.** Mueve datos inmediatos

```
MOV  AX,25            ,-Inmediato a registro
MOV  BYTEVAL,  25     /Inmediato a memoria,  directo
MOV  WORDVAL  [BX],  25  /Inmediato a memoria,  indirecto
```

**2.** Mueve registros

```
MOV   EAX,ECX         Registro a registro
MOV   DS, AX          Registro a registro de segmento
MOV   BYTEVAL,  BH    Registro a memoria,  directo
MOV   [SI] , AX       Registro a memoria,  indirecto
```

3. Mueve memoria directa

```
MOV  BH,BYTEVAL        /Memoria a registro, directo

MOV AX,WORDVAL [BX]    ,-Memoria a registro, indirecto
```

**4.** Mueve registro de segmento

```
MOV  AX,DS             ,-Registro de segmento a registro

MOV  WORDVAL, DS       /Registro de segmento a memoria
```

Puede mover a un registro un byte (MOV AH,BYTEVAL), una palabra (MOV AX, WORDVAL) o una palabra doble (MOV EAX, DWORD VAL). El operando sólo afecta la parte del registro referenciado; por ejemplo, mover un byte al AH no afecta el AL.

Las operaciones MOV que no son permitidas son de memoria a memoria (tenga esto en mente), inmediato a registro de segmento y de registro de segmento a registro de segmento. Para manejar estas operaciones, tiene que codificar más de una instrucción.

## INSTRUCCIONES PARA MOVER Y LLENAR

Una limitación de la instrucción MOV es que el destino debe ser de la misma longitud que el fuente, tal como un byte a byte y una palabra a una palabra. En el 80386 y procesadores posteriores, las instrucciones MOVSX y MOVZX (mover y llenar) facilitan la transferencia de datos de un byte o palabra fuente a una palabra o palabra doble de destino. Aquí está el formato general de MOVSX y MOVZX:

```
[etiqueta]   MOVSX/MOVZX  {registro/memoria},{registro/memoria/inmediato}
```

MOVSX, para uso con valores aritméticos con signo, mueve un byte o palabra a una palabra o palabra doble de destino y llena con el bit de signo (el último bit a la izquierda del origen) los bits de más a la izquierda del destino. MOVZX, para uso con valores numéricos sin signo, mueve un byte o palabra a una palabra o palabra doble de destino y llena con bits cero los bits de más a la izquierda del destino. Como ejemplo, considere mover un byte con 1011 0000 a una palabra; el resultado en la palabra destino depende de la elección de la instrucción:

```
MOVSX: 1111  1111  1011 0000

MOVZX: 0000  0000  1011 0000
```

Aquí están algunos ejemplos del uso de MOVSX y MOVZX:

```
BYTEVAL  DB     ?
WORDVAL  DW     ?

         MOVSX AX,  BYTEVAL   /Byte a palabra
```

```
MOVSX EAX, WORDVAL   ;Palabra a palabra doble
MOVZX WORDVAL, AH    ;Byte a palabra
MOVZX EAX, WORDVAL   /Palabra a palabra doble
```

Los capítulos 8 y 13 cubren con todo detalle los datos con y sin signo.

## OPERANDOS INMEDIATOS

En el ejemplo siguiente de un operando inmediato, la instrucción

```
MOV  AX,0123H
```

mueve la constante inmediata 0123H al registro AX. El código de tres bytes para esta instrucción es B82301, en donde B8 significa "mueve un valor inmediato al registro AX" y los dos bytes siguientes contienen el valor (2301H, en orden inverso de bytes). Muchas instrucciones estipulan dos operandos; el primero puede ser un registro o localidad de memoria y el segundo puede ser una constante inmediata.

El uso de un operando inmediato da procesamiento más eficiente que definir una constante numérica en el segmento de datos y referenciarla en el operando del **MOV,** como en el ejemplo siguiente:

```
Segmento de datos:    AMT1  DW  0123H    /Define AMT1 como palabra

Segmento de código:   MOV   AX, AMT1     /Mueve a AMT1 a AX
```

### Longitud de los operandos inmediatos

La longitud de una constante inmediata no puede exceder la longitud definida por el primer operando. En el ejemplo siguiente, no válido, el operando es de dos bytes, pero el registro AL es de sólo un byte:

```
MOV  AL, 0123H       / Longitud- no válida
```

Sin embargo, si un operando inmediato es más corto que el operando receptor, como en

```
ADD  AX.25H      /Longitud válida
```

el ensamblador expande el operando a dos bytes, 0025H, y almacena el código objeto como 2500H.

El 80386 y procesadores posteriores permiten operandos inmediatos de cuatro bytes (palabra doble), tal como en

```
MOV  EAX,12345678H      /Mueve palabra doble
```

### Formatos inmediatos

Una constante inmediata puede estar en cualquier formato definido válido. Aquí están algunos ejemplos:

```
        PAGE   50,132
TITLE   P06IMMED (EXE)  Ejemplos de operandos inmediatos
;       (Coded for assembly only, NOT for execution)
        .MODEL SMALL
        .STACK 64              ,-Se define la pila
        .DATA                  ;Se definen los datos
FLDA    DB      7
FLDB    DW      ?
.386
        .CODE
BEGIN   PROC    FAR
        MOV     AX,275         ,•Mover inmediato
        ADD     AX,125         ;Suma inmediata
        SUB     AX,200         ;Resta inmediata
        MOV     EBX,0          ;Mover inmediato (80386)
        ADD     BX,20H         ,•Suma inmediata (hex)
BEGIN   ENDP
        END
```

Figura 6-1 Operaciones inmediatas

```
Hexadecimal:   0123H

Decimal:       291 (que el ensamblador convierte en 0123H)

Binario:       100100011B (que convierte en 0123H)
```

MOV, ADD y SUB son tres de las muchas instrucciones que permiten operandos inmediatos. La figura 6-1 da ejemplos de estas instrucciones. La directiva .386 permite al ensamblador reconocer la referencia al registro EBX. No se necesita un 80386 o procesador posterior para ensamblar este enunciado, pero sí para ejecutarlo. Ya que el ejemplo no tiene la intención de ejecutarse, no se define una pila ni se inicializa el registro DS.

Procesar elementos más largos que la capacidad de un registro exige codificación adicional, tratada en capítulos posteriores.

## LA INSTRUCCIÓN XCHG

La instrucción XCHG realiza otro tipo de transferencia de datos, pero en lugar de copiar los datos de una localidad a otra, XCHG intercambia los datos. El formato general para XCHG es

```
[etiqueta:]  XCHG  {registro/memoria},  {registro/inmediato}
```

Operaciones válidas con XCHG implican intercambio de datos entre dos registros y entre un registro y la memoria. Aquí están ejemplos:

```
WORD   DW  ?


       XCHG AL, AH       /Intercambia los contenidos de los dos registros

       XCHG AX, WORDX    /Intercambia los contenidos del registro y la memoria
```

## LA INSTRUCCIÓN LEA

La instrucción LEA es útil para inicializar un registro con una dirección de desplazamiento. De hecho, un nombre más descriptivo para esta instrucción sería "Load Offset Address, carga una dirección de desplazamiento". El formato general para LEA es

```
[etiqueta:]  LEA  {registro/memoria}
```

Un uso común de LEA es para inicializar un desplazamiento en el registro BX, DI o SI para indexar una dirección de memoria. Haremos mucho de esto a lo largo de este libro. Aquí está un ejemplo:

```
DATABLK DB  2 0 D0P (?)

SAVBYTE DB


          LEA BX,  DATABLK     ,-Carga la dirección del desplazamiento

          MOV SAVBYTE,  [BX]    /Mueve el primer byte de DATABLK
```

Una operación equivalente a LEA o MOV con desplazamiento, se codifica así:

```
MOV BX,  OFFSET DATABLK    /Carga la dirección del desplazamiento
```

## LAS INSTRUCCIONES INC Y DEC

INC y DEC son instrucciones adecuadas para aumentar y disminuir en 1 los contenidos de registros y localidades de memoria. El formato general para INC y DEC es

```
[etiqueta:]  INC/DEC  {registro/memoria}
```

Note que esas instrucciones sólo necesitan de un operando. Dependiendo del resultado, la operación apaga o prende las banderas OF, SF y ZF, a las que las instrucciones de salto condicional pueden verificar para menos, cero o más.

## INSTRUCCIONES DE MOVIMIENTO EXTENDIDO

Los programas anteriores movieron datos inmediatos a un registro, movieron datos de una localidad de memoria definida a un registro, movieron contenidos de registros a memoria y movieron el contenido de un registro a otro. En todos los casos, la longitud de los datos estaba limitada a uno o dos bytes y ninguna operación movió datos de un área de memoria directamente a otra área de memoria. Esta sección explica cómo mover datos que exceden los dos bytes. Otro método, el uso de instrucciones de cadenas de caracteres, es estudiado en el capítulo 12.

En el programa de la figura 6-2, el segmento de datos contiene dos campos de nueve bytes definidos como NAME1 y NAME2. El objetivo del programa es mover el contenido de NAME1 a NAME2:

```
        page   60,132
TITLE   P06MOVE (EXE)   Operaciones de movimiento extendidos

        .MODEL SMALL
        .STACK 64

        .DATA
NAME1   DB      'ABCDEFGHI'
ÑAME2   DB      'JKLMNOPQR'

        .CODE
BEGIN   PROC    FAR
        MOV     AX,@data        Inicia registros
        MOV     DS,AX             de segmento
        MOV     ES,AX

        MOV     CX, 09          Iniciación para mover 9 caracteres
        LEA     SI,NAMEl        Iniciación de direcciones para NAME1
        LEA     DI,NAME2             Y ÑAME2
B20:
        MOV     AL, [SI]        Obtener carácter de NAME1,
        MOV     [DI] , AL       Moverlo a ÑAME2
        INC     SI              Incrementar siguiente carácter en NAME1
        INC     DI              Incrementar, a siguiente posición, en NAME2
        DEC     CX              Decrementar contador de iteraciones
        JNZ     B20             ¿Contador diferente de cero? Si, iterar

        MOV     AX,4C00H        •Salida a DOS
        INT     21H
BEGIN   ENDP
        END     BEGIN
```

Figura 6-2 Operaciones de movimiento extendido

```
NAMEl:    A B C D E F G H I
          I I I I I I I I I
NAME2:    J K L M N O P Q R
```

Ya que cada uno de los campos es de nueve bytes, se necesita más de una instrucción MOV. El programa contiene varias características nuevas.

A fin de pasar NAMEl a NAME2, la rutina inicializa el registro CX a 9 (la longitud de los dos campos) y utiliza los registros índice SI y DI. Dos instrucciones LEA cargan las direcciones de desplazamiento de NAMEl y NAME2 en SI y DI como sigue:

```
LEA   SI,NAMEl      ;Carga  desplazamientos

LEA   DI,NAME2      ; de NAMEl y ÑAME2
```

El programa utiliza las direcciones de los registros SI y DI para mover el primer byte de NAMEl al primer byte de NAME2. Los corchetes alrededor de SI y DI en los operandos de MOV significan que la instrucción es para usar el desplazamiento en el registro dado, a fin de accesar la localidad de memoria. Así

```
MOV  AL,  [SI]
```

significa "Utilice el desplazamiento en SI (NAMEl +0) para mover el byte referenciado al registro AL". Y la instrucción

```
MOV  [DI] , AL
```

significa "Mueva el contenido de AL al desplazamiento referenciado por DI(NAME2+0)". El programa tiene que repetir estas dos instrucciones MOV nueve veces, una vez para cada carácter en los campos respectivos. Para este fin, utiliza una instrucción que aún no hemos explicado: JNE (Salta si no es igual).

Dos instrucciones INC incrementan los registros SI y DI en 1, _y DEC decrementa el CX en 1. DEC también pone a 1 o a 0 la bandera de cero (ZF), dependiendo del resultado en CX; si el contenido no es cero, aún existen caracteres por mover, y JNE regresa a la etiqueta B20 para repetir las instrucciones MOVE. Y como el SI y DI han sido incrementados en 1, el siguiente MOV hace referencia a NAME1 +1 y NAME2+1. El ciclo continúa de esta manera hasta que ha movido nueve caracteres en total, hasta mover NAME1 + 8 a NAME2 + 8.

(Tal vez quiera teclear este programa, ensamblarlo y enlazarlo y utilizar DEBUG para rastrearlo. Observe el resultado en los registros, el apuntador de instrucción y la pila. Utilice D DS:0 para ver los cambios en NAME2.)

## LA INSTRUCCIÓN INT

En ejecución, una instrucción INT interrumpe el procesamiento y accesa la tabla de servicios de interrupción en memoria baja para determinar la dirección de la rutina solicitada. Después, la operación transfiere al DOS o al BIOS para una acción especificada y regresa a su programa para continuar el procesamiento. Con más frecuencia, una interrupción tiene que realizar los pasos complejos de una operación de entrada o salida. Las interrupciones necesitan de un camino que facilite la salida de un programa y, tras una terminación exitosa, el regreso al programa. Para este objetivo, INT realiza lo siguiente:

- Decrementa en 2 el apuntador de la pila y mete en la pila el contenido del registro de banderas.
- Limpia (pone a 0) las banderas de interrupción y de trampa (IF y TF).
- Decrementa en 2 el apuntador de la pila y mete en la pila el registro CS.
- Decrementa en 2 el apuntador de la pila y mete en la pila el apuntador de instrucción.
- Hace que la operación solicitada sea realizada.

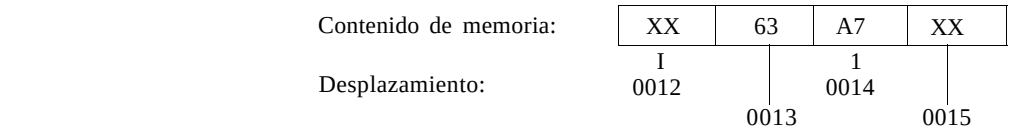
Para regresar de una interrupción, la rutina emite un IRET (regreso de interrupción), el cual saca los registros de la pila y regresa a la instrucción inmediata posterior al INT en su programa.

Ya que el proceso anterior es automático por completo, sus únicas preocupaciones son definir una pila suficientemente grande, para las operaciones necesarias de agregar y remover información de ella y utilizar las operaciones INT adecuadas. A partir del capítulo 9, haremos uso considerable de la instrucción INT.

## ALINEACIÓN DE DIRECCIONES

Como el 8086 y el 80286 tienen un bus de datos de 16 bits (una palabra), ejecutan (trabajan) más rápido si accesan palabras que empiezan en una dirección (palabra) con número par. Considere una situación en la que los desplazamientos 0012H y 0013H contienen la palabra 63 A7H. El

procesador puede accesar la palabra completa en el desplazamiento 0012H de forma directa a un registro. Pero la palabra pudo empezar en una dirección con número impar, tal como 0013H:




En este caso, el procesador tiene que realizar *dos* accesos. Primero, accesa los bytes en 0012H y 0013H y envía el byte de 0013H (63) al registro AL. Después accesa los bytes en 0014H y 0015H y envía el byte de 0014H (A7) al registro AH. Ahora el AX contiene A763H.

Usted no tiene que realizar ninguna programación especial para localidades pares o impares, ni tiene que saber si una dirección es par o impar. La operación de acceso invierte de forma automática una palabra de memoria en un registro, de manera que retome su orden correcto.

El 80386 y procesadores posteriores tienen un bus de datos de 32 bits. De acuerdo con esto, se prefiere la alineación de elementos referenciados en direcciones que sean divisibles entre cuatro (una dirección de palabra doble). (Técnicamente, los procesadores 486 y Pentium prefieren alineación en un límite de 16 bytes [párrafo].)

El lenguaje ensamblador tiene una directiva ALIGN que se puede usar para alinear elementos en límites. Por ejemplo, ALIGN 2 alinea en un límite de palabra y ALIGN 4 alinea en un límite de palabra doble. También, como el inicio del segmento de datos siempre está en un límite de párrafo, podría organizar sus primeros datos con valores de palabras dobles, después con valores de palabra y por último con valores de byte. Sin embargo, el 80386 y procesadores posteriores ejecutan a velocidad tan rápida que usted probablemente nunca notará los efectos de forzar el alineamiento.

## DIRECCIONES CERCANA Y LEJANA

En un programa, una dirección puede ser cercana o lejana. Una dirección *cercana* sólo consiste en la parte de desplazamiento de una dirección. Una instrucción que hace referencia a una dirección cercana supone al segmento actual —a saber, el DS para el segmento de datos y el CS para el segmento de código.

Una dirección *lejana* consta de dos partes, la del segmento y la del desplazamiento, en la forma segmento desplazamiento. Una instrucción puede referenciar una dirección lejana desde cualquier segmento (incluyendo el actual).

Casi toda la programación en ensamblador hace uso de direcciones cercanas, las cuales genera el ensamblador a menos que se le instruya de otra manera. Programas grandes en los que los segmentos ocupan más de 64K de memoria pueden necesitar de direcciones lejanas.

## PREFIJO QUE INVALIDA EL SEGMENTO

Para la mayoría de los propósitos, una referencia a un área de datos en un programa es a localidades en el segmento de datos, manejados por medio del registro DS. Sin embargo, existen ocasiones —en especial para programas grandes— cuando usted puede tener que manejar datos que están

en otro registro de segmento, tal como el ES o, en el 80386 y procesadores posteriores, el FS o GS. Un buen ejemplo sería una tabla grande de datos cargados del disco a la memoria.

Puede utilizar cualquier instrucción para procesar datos en los otros segmentos, pero debe identificar el registro de segmento apropiado. Digamos que la dirección del otro segmento está en el registro ES y que el BX contiene el desplazamiento dentro del segmento. Suponga que el requisito es mover dos bytes (una palabra) desde esa localidad al registro CX:

```
MOV CX,ES:  [BX]   ;Mueve a CX desde ES: [BX]
```

La codificación de ES: indica un operador de invalidación que significa "Reemplace el uso normal del registro de segmento DS con el de ES".

El ejemplo siguiente mueve un valor de un byte desde el AL a este otro segmento, en un desplazamiento formado por el valor en el DI más 24:

```
MOV ES:  [DI+24] ,  AL  ,-Mueve a ES:[DI+24]  desde AL
```

El ensamblador genera el código en lenguaje de máquina con el operador de invalidación insertado como un prefijo de un byte (26H), precediendo a la instrucción, igual que si hubiera codificado la instrucción como

```
ES:  MOV CX,  [BX]     ,-Mueve a CX desde ES:  [BX]

ES:  MOV  [DI+24]      ,-Mueve a ES: [DI+24]  desde AL
```

## PUNTOS CLAVE

- Un operando proporciona una fuente de datos para una instrucción. Algunas instrucciones no necesitan operandos, mientras que otras pueden tener uno o dos operandos.
- En donde existan dos operandos, el segundo es el fuente, que contiene ya sea datos inmediatos o la dirección (de un registro o de memoria) de los datos. El primer operando es el destino, que contiene datos en un registro o en la memoria que serán procesados.
- En formato inmediato, el segundo operando contiene un valor constante o una expresión. Los operandos inmediatos deben coincidir con el tamaño de un registro: una constante de un byte con un registro de un byte (AL, BH) y una constante de una palabra con un registro de una palabra (AX, BX).
- En formato de memoria directa, uno de los operandos hace referencia a una localidad de memoria y el otro a un registro.
- El direccionamiento indirecto utiliza la capacidad de la computadora para direccionamiento segmento desplazamiento. Los registros usados son BX, DI, SI y BP, codificados dentro de corchetes como un operador de índice. BX, DI y SI están asociados con el DS como DS:BX, DS:DI y DS:SI, respectivamente, para procesamiento de datos en el segmento de datos. El BP está asociado con el SS como SS:BP para manejo de datos en la pila.
- Puede combinar los registros en un direccionamiento indirecto como [BX + SI], lo que significa la dirección en BX más la dirección en el SI.

- La instrucción MOV transfiere (o copia) datos referenciados por la dirección en el segundo operando a la dirección en el primer operando.
- La instrucción LEA es útil para inicializar un registro con un desplazamiento.
- INC y DEC son instrucciones adecuadas para incrementar y decrementar en 1 los contenidos de registros y de localidades de memoria.
- La instrucción INT interrumpe el procesamiento de su programa, transfiere al DOS o al BIOS para una acción específica y regresa su programa para continuar el procesamiento.

PREGUNTAS

**6-1.** Para una instrucción con dos operandos, ¿cuál es el fuente y cuál el destino?

**6-2.** (a) ¿De qué manera significativa difieren las siguientes instrucciones en su ejecución?

```
MOV  AX, 325AH

MOV   AX, [325AH]
```

**(b)** Para el segundo MOV, un operando está entre corchetes. ¿Cuál es el nombre de esta característica?

**6**-3. (a) ¿De que manera significativa difieren las siguientes instrucciones en su ejecución?

```
MOV   BX,0

MOV  [BX] , 0
```

**(b)** Para el segundo MOV, ¿qué tipo de direccionamiento está involucrado con el primer operando?

**6-4.** Explique la operación de la instrucción

```
MOV  CX,  [BX+SI+4]
```

**6**-5. El enunciado siguiente tiene un error; esto es, se necesita algo para que el ensamblador lo traduzca:

```
MOV  [BX] , [SI]
```

(a) ¿Cuál es el error?

**(b)** ¿Cómo corregiría el error?

**6-6.** Dada la siguiente definición de datos, encuentre los errores en los enunciados y codifique las instrucciones necesarias para corregirlos:

```
BYTEl  DB ?

BYTE2  DB ?

WORD1  DW ?

  (a)   MOV BYTEl,  BYTE2

  (b)   MOV AL,  WORD1     ;El operando 1 es correcto

  (c)   MOV BL,  034AH     ;El operando 2 es correcto
```

**6**-7. Codifique lo siguiente como instrucciones con operandos inmediatos: (a) almacenar 320 en el AX; (b) comparar FLDB con cero; (c) sumar 40 hex al BX; (d) restar 40 hex del CX; (e) recorrer FLDB un bit a la izquierda; (f) recorrer el CH un bit a la derecha.

**6-8.** Codificar una instrucción que intercambie los contenidos de una palabra llamada WORDl con el CX.

**6-9.** Codifique instrucciones para establecer BX con la dirección (desplazamiento) de un elemento llamado TABLEX.

**6-10.** En términos generales, ¿cuál es el objetivo de la instrucción INT?

**6-11.** (a) ¿Cómo afecta la instrucción INT a la pila? (b) ¿Cómo afecta la instrucción IRET a la pila?

**6-12.** Codifique, ensamble, enlace y utilice DEBUG para probar el programa siguiente:

- Defina elementos byte llamados BYTEA y BYTEB (con cualquier valor) y una palabra llamada WORDC (con cero).
- Mueva el contenido de BYTEA al AL.
- Sume el contenido de BYTEB al AL.
- Mueva el valor inmediato 25H al BL.
- Intercambie los contenidos del AL y BL.
- Multiplique el contenido de BL por el de AL (MUL BL).
- Almacene el producto en AX y envíelo a WORDC.

## CAPÍTULO 7

# Escritura de programas .COM

OBJETIVO

**Explicar el objetivo y los usos de programas .COM y cómo preparar un programa en lenguaje ensamblador para ese formato.**

### INTRODUCCIÓN

Hasta ahora sólo hemos escrito, ensamblado y ejecutado programas .EXE. De forma automática, el enlazador genera un formato particular para un programa .EXE y, cuando se almacena en disco, es precedido por un bloque especial de encabezado que al menos es de 512 bytes (el capítulo 24 proporciona detalles de los bloques de encabezado).

También puede generar un programa .COM para ejecución. Un ejemplo de uso común de programa .COM es el COMMAND.COM. Las ventajas de programas .COM están en que son más pequeños que programas .EXE comparables y son más fáciles de adaptar para actuar como programas residentes en memoria. El formato .COM tiene sus raíces en los días anteriores al DOS, cuando el tamaño de los programas estaba limitado a 64K.

### DIFERENCIAS ENTRE PROGRAMAS .COM Y .EXE

Algunas diferencias importantes entre un programa que es para ejecutarse como .EXE y uno que es para ejecutarse como .COM implica el tamaño del programa, la segmentación y la inicialización.

### Tamaño del programa

En la práctica, un programa .EXE puede ser de cualquier tamaño, mientras que un programa .COM está restringido a un segmento y a un máximo de 64K, incluyendo el PSP. EL PSP es un bloque de 256 bytes (100H) que el DOS inserta antes de los programas .COM y .EXE cuando los carga en memoria. El límite de 64K es una regla general; puede darle la vuelta codificando enunciados SEGMENT AT adicionales, una característica que está fuera del alcance de este capítulo. Un programa .COM siempre es más pequeño que su contraparte .EXE; una razón es que el bloque de encabezado de 512 bytes a un programa .EXE no precede a un programa .COM. (No confunda el bloque de encabezado con el PSP.) Un programa .COM es una imagen absoluta del programa ejecutable, pero sin información de direcciones reubicables.

### Segmentos

El uso de segmentos para programas .COM es muy diferente (y más fácil) que para programas .EXE.

**Segmento de la pila.** Usted define un programa .EXE con un segmento de pila, mientras que un programa .COM genera de manera automática una pila. Así, cuando escribe un programa en lenguaje ensamblador que será convertido a formato .COM, omite la definición de la pila. Si los 64K del tamaño del programa no es suficiente, el ensamblador establece la pila fuera del programa, en memoria superior.

**Segmento de datos.** Un programa .EXE por lo común define un segmento de datos e inicializa el registro DS con la dirección de ese segmento. Ya que los datos para un programa .COM están definidos dentro del segmento de código, tampoco tiene que definir el segmento de datos. Como verá, existen formas sencillas de manejar esta situación.

**Segmento de código.** Un programa .COM completo combina el PSP, la pila, el segmento de datos y el segmento de código en un segmento de código de un máximo de 64K.

### Inicialización

Cuando el DOS carga un programa .COM para ejecución, inicializa de forma automática todos los registros de segmentos con la dirección del PSP. Ya que los registros CS y DS contendrán la dirección de segmento inicial correcta, su programa no tiene que cargarlos.

Puesto que el direccionamiento comienza en un desplazamiento de 100H bytes desde el inicio del PSP, codifique una directiva ORG como ORG 100H inmediatamente después de SEGMENT (segmento de código) o el enunciado .CODE. La directiva ORG le indica al ensamblador que empiece la generación del código objeto en un desplazamiento de 100H bytes pasando el inicio del PSP, en donde el programa .COM real inicia.

## CONVERSIÓN A FORMATO .COM

Si su programa fuente ya está escrito en formato .EXE, puede utilizar un editor para convertir las instrucciones a formato .COM. Los formatos de codificación de MASM y TASM para programas .COM son idénticos, aunque sus métodos de conversión difieren. Cuando la conversión a formato .COM está completa, puede borrar los archivos .OBJ y .EXE.

### Conversión con Microsoft

Para ambos programas, .EXE y .COM, con MASM de Microsoft se ensambla y produce un archivo .OBJ y después se enlaza para producir un programa .EXE. Si escribió el programa para ejecutarse como un programa .EXE, ahora puede ejecutarlo. Si escribió el programa para ejecutarse como un programa .COM, el enlazador produce un mensaje:

```
Advertencia:  No existe segmento de la pila (STACK)
```

Puede ignorar este mensaje, ya que se supone que no debe existir definida una pila. Un programa con nombre EXE2BIN convierte programas .EXE a programas .COM. (En realidad, convierte programas .EXE a un archivo .BIN binario; el nombre del programa significa "convierte EXE a BIN", pero debe poner a su archivo de salida la extensión .COM.) Suponiendo que EXE2BIN está en la unidad por omisión, y que el archivo enlazado llamado CALC.EXE está en la unidad D, teclee.

```
EXE2BIN  D:CALC D:CALC.COM  [ENTER]
```

Ya que el primer operando del comando siempre se refiere a un archivo .EXE, no codifique la extensión .EXE. El segundo operando puede ser un nombre diferente a CALC.COM. Si omite la extensión, EXE2BIN supone que es BIN, que después tendría que renombrar como .COM a fin de ejecutar el programa (alguien, en algún lugar, debió pensar que esta forma era una buena idea).

### Conversión con Borland

Con tal de que su programa fuente esté codificado de acuerdo con los requisitos .COM, usted puede convertir en forma directa su programa objeto en programa .COM. Utilice la opción *IT* para TLINK:

```
TLINK /T D:CALC
```

## EJEMPLO DE UN PROGRAMA .COM

El programa de la figura 7-1, llamado EXC0M1, es el mismo de la figura 5-2, pero ahora está corregido para ajustarse a los requisitos .COM. Note los cambios siguientes de la figura 5-2.

- No existen definidos una pila o un segmento de datos.
- Un enunciado ASSUME le indica al ensamblador que inicie los desplazamientos desde el inicio del segmento de código. El registro CS también tiene esta dirección, que es la del PSP. Sin embargo, la directiva ORG hace que el programa empiece 100H bytes desde este punto, inmediatamente a continuación del PSP.
- ORG 100H establece un desplazamiento para el inicio de ejecución. El cargador de programa almacena esta dirección en el apuntador de instrucción.
- Una instrucción JMP transfiere el control a la ejecución pasando los datos definidos. Algunos programadores codifican los datos después de las instrucciones, de manera que la instrucción inicial JMP no es necesaria. Codificando primero los datos puede acelerar ligeramente el proceso de ensamble, pero no da ninguna otra ventaja.

```
        page 60,132
TITLE   P07COM1   Programa .COM para mover y sumar
CODESG  SEGMENT PARA 'Code'
        ASSUME  CS:CODESG,DS  CODESG,SS:CODESG,ES:CODESG
        ORG     100H          Inicio al final de PSP
BEGIN:  JMP     MAIN          Salto pasando los datos

FLDA    DW      250           ;Definiciones de datos
FLDB    DW      125
FLDC    DW      ?

MAIN    PROC    NEAR
        MOV     AX,FLDA       Mover 0250 a AX
        ADD     AX, FLDB      Sumar 0125 a AX
        MOV     FLDC,AX       Almacenar suma en FLDC
        MOV     AX,4C00H      Salida a DOS
        INT     21H
MAIN    ENDP
CODESG  ENDS
        END     BEGIN
```

Figura 7-1 Programa fuente .COM con segmentos convencionales

- INT 21H, función 4CH, finaliza el procesamiento y sale al DOS. Para este propósito, también puede usar la instrucción RET.

Aquí están los pasos para convertir el programa para MASM y TASM:

| MASM | TASM |
|---|---|
| MASM D:EXCOMl,D: | TASM D:EXCOMÍ,D: |
| LINK D:EXCOMl,D: | TLINK /T D:EXCOMl,D: |
| EXE2BIN D:EXCOMl D:EXCOMl.COM | |

Los programas .EXE y .COM son de 792 bytes y de 24 bytes, respectivamente. La diferencia es en gran parte causada por el bloque de encabezado de 512 bytes almacenado al inicio de los módulos .EXE. Teclee DEBUG D:EXCOMl .COM para rastrear la ejecución del programa .COM hasta (pero no incluyendo) la última instrucción.

Cuando codifique un programa .COM, también puede utilizar directivas simplificadas de segmentos, como se muestra en la figura 7-2. Una vez más, sólo define un segmento de código, no una pila ni un segmento de datos.

## LA PILA DE .COM

Para un programa .COM, el DOS define de manera automática una pila y establece la misma dirección de segmento en los cuatro registros de segmento. Si el segmento de 64K para el programa es suficientemente grande, el DOS establece la pila al final del segmento y carga el registro SP con FFFEH, la parte superior de la pila (el tope de la pila).

Si el segmento de 64K no contiene espacio suficiente para una pila, el DOS establece la pila al final de la memoria. En cualquier caso, el DOS mete después una palabra con cero a la pila, la cual actúa como un desplazamiento para el IP, si usted utiliza RET para terminar la ejecución del programa.

Si su programa es grande, o si la memoria está limitada, debe tener cuidado al enviar palabras a la pila. El comando DIR indica el tamaño de un archivo y le dará una idea del espacio

```
        page 60 132
TITLE   P07COM2  Programa  .COM para mover y sumar datos
        .MODEL   SMALL
        .CODE
        ORG      100H          ,• Inicio al final de PSP
BEGIN:  JMP      MAIN          ,- Salto pasando los datos

FLDA    DW       250           ,-Definiciones de datos
FLDB    DW       125
FLDC    DW

MAIN    PROC     NEAR
        MOV      AX, FLDA      ,-Mover 0250 a AX
        ADD      AX,FLDB       /Sumar 0125 a AX
        MOV      FLDC,AX       /Almacenar suma en FLDC
        MOV      AX,4C00H      /Volver a DOS
        INT      21H
MAIN    ENDP
        END      BEGIN
```

Figura 7-2 Programa fuente .COM con directivas simplificadas de segmento

disponible para una pila. La mayoría de los programas más pequeños en este libro están en formato .COM, que deben ser distinguidos con facilidad de los de formato .EXE.

## SUGERENCIAS PARA LA DEPURACIÓN

La omisión de un solo requisito .COM puede provocar que un programa falle. Si EXE2BIN encuentra un error, sólo le notifica que no puede convertir el archivo, pero no da la razón. Verifique los enunciados SEGMENT, ASSUME y END. Si omite ORG 100H, de forma incorrecta el programa se refiere a los datos en el PSP, con resultados impredecibles.

Si ejecuta un programa .COM con DEBUG, utilice D CS:100 para ver los datos e instrucciones. No siga el programa hasta su terminación; en lugar de eso, utilice el comando Q de DEBUG.

Un intento de ejecutar un módulo .EXE de un programa escrito como .COM fallará.

## PUNTOS CLAVE

- Un programa .COM está restringido a un segmento de 64K.
- Un programa .COM es más pequeño que su programa .EXE contraparte.
- Un programa escrito para correr como .COM no define una pila o un segmento de datos ni inicializa el registro DS.
- Un programa escrito para correr como .COM utiliza ORG 100H inmediatamente después del enunciado SEGMENT. El enunciado establece la dirección de desplazamiento al inicio de la ejecución que sigue al PSP.
- Para MASM de Microsoft, el programa EXE2BIN convierte un archivo .EXE a formato .COM. TLINK de Borland puede convertir un programa objeto directamente a formato .COM.
- El DOS define una pila para un programa .COM al final del programa.

## PREGUNTAS

**7-1.** ¿Cuál es el tamaño máximo de un programa .COM?

**7-2.** Para un programa fuente que será convertido a formato .COM, ¿qué segmentos puede definir?

**7-3.** ¿Por qué debe codificar ORG 100H al inicio de un programa que será convertido a formato .COM?

**7-4.** ¿Cómo maneja el sistema el hecho de que usted no define una pila para un programa .COM?

**7-5.** Un programa fuente tiene por nombre SAMPLE.ASM. Proporcione los comandos para convertir a formato .COM bajo (a) MASM; (b) TASM.

**7-6.** Corrija el programa de la pregunta 6-12 para formato .COM. Ensámblelo, enlácelo y ejecútelo con DEBUG.

# CAPITULO 8

# Lógica y control de programas

OBJETIVO

**Cubrir los requisitos para control de programas (ciclos y transferencia de control [saltos]), para comparaciones lógicas, para operaciones lógicas entre bits y para organización del programa.**

## INTRODUCCIÓN

Hasta este capítulo los programas que hemos examinado han sido ejecutados en forma lineal, esto es, con una instrucción secuencialmente a continuación de otra. Sin embargo, rara vez un problema programable es tan sencillo. La mayoría de los programas constan de varios ciclos en los que una serie de pasos se repite hasta alcanzar un requisito específico y varias pruebas para determinar qué acción se realiza de entre varias posibles. Una práctica común es verificar si un programa está al final de su ejecución.

Requisitos como éstos implican la transferencia de control a la dirección de una instrucción que no sigue de inmediato de la que se está ejecutando actualmente. Una transferencia de control puede ser *hacia adelante,* para ejecutar una serie de pasos nuevos, o *hacia atrás,* para volver a ejecutar los mismos pasos.

Ciertas instrucciones pueden transferir el control fuera del flujo secuencial normal añadiendo un valor de desplazamiento al IP. A continuación están las instrucciones introducidas en este capítulo, por categorías:

| OPERACIONES DE COMPARACIÓN | OPERACIONES DE TRANSFERENCIA | OPERACIONES LÓGICAS | CORRIMIENTO Y ROTACIÓN |
|---|---|---|---|
| CMP | CALL | AND | SAR/SHR |
| TEST | JMP | NOT | SAL/SHL |
| Jnnn | OR | RCR/ROR | |
| LOOP | XOR | RCL/ROL | |

## DIRECCIONES CORTA, CERCANA Y LEJANA

Una operación de salto alcanza una dirección *corta* por medio de un desplazamiento de un byte, limitado a una distancia de -128 a 127 bytes. Una operación de salto alcanza una dirección *cercana* por medio de un desplazamiento de una palabra, limitado a una distancia de -32,768 a 32,767 bytes dentro del mismo segmento. Una dirección *lejana* puede estar en otro segmento y es alcanzada por medio de una dirección de segmento y un desplazamiento; CALL es la instrucción normal para este propósito.

La tabla siguiente indica las reglas sobre distancias para las operaciones JMP, LOOP y **CALL.** Hay poca necesidad de memorizar esta reglas, ya que el uso normal de estas instrucciones en rara ocasión causa problemas.

| | Corta | Cercana | Lejana |
|---|---|---|---|
| Instrucciones | Mismo segmento -128 a 127 | Mismo segmento -32,768 a 32,767 | Otro segmento |
| JMP | sí | sí | sí |
| Jnnn | sí | sí: 80386 y posteriores | no |
| LOOP | sí | no | no |
| CALL | N/A | sí | sí |

## ETIQUETAS DE INSTRUCCIONES

Las instrucciones JMP, Jnnn (salto condicional) y LOOP requieren un operando que se refiere a la etiqueta de una instrucción. El ejemplo siguiente salta a A90, que es una etiqueta dada a una instrucción MOV:

```
        JMP     A90

A90     MOV.    AH,00
```

La etiqueta de una instrucción, tal como A90:, terminada con dos puntos (:) para darle el atributo de cercana —esto es, la etiqueta está dentro de un procedimiento en el mismo segmento de código. Cuidado: Un error común es la omisión de los dos puntos. Note que una etiqueta de dirección en un operando de instrucción (como JMP A20) no tiene un carácter de dos puntos.

También puede codificar una etiqueta en una línea separada como

```
A90

        MOV.  AH,0  0
```

En ambos casos, la dirección A90 se refiere al primer byte de la instrucción MOV.

## LA INSTRUCCIÓN JMP

Una instrucción usada comúnmente para la transferencia de control es la instrucción JMP (jump, salto, bifurcación). Un salto es incondicional, ya que la operación transfiere el control bajo cualquier circunstancia. También, JMP vacía el resultado de la instrucción previamente procesada; por lo que, un programa con muchas operaciones de salto puede perder velocidad de procesamiento. El formato general para JMP es

```
[etiqueta:]  JMP  dirección corta, cercana o lejana
```

Una operación JMP dentro del mismo segmento puede ser corta o cercana (o de manera técnica, lejana, si el destino es un procedimiento con el atributo FAR). En su primer paso por un programa fuente, el ensamblador genera la longitud de cada instrucción. Sin embargo, una instrucción JMP puede ser de dos o tres bytes de longitud. Una operación JMP a una etiqueta dentro de -128 a +127 bytes es un salto corto. El ensamblador genera un byte para la operación (EB) y un byte para el operando. El operando actúa como un valor de desplazamiento que la computadora suma al registro IP cuando se ejecuta el programa. Los límites son de OOH hasta FFH, o de -128 hasta +127. El ensamblador ya puede haber encontrado el operando designado (un salto hacia atrás) dentro de -128 bytes, como en

```
A50:

    JMP     A50
```

En este caso, el ensamblador genera una instrucción de máquina de dos bytes. Una JMP que excede -128 a +127 bytes se convierte en un salto cercano, para el que el ensamblador genera un código de máquina diferente (E9) y un operando de dos bytes (8086/80286) o un operando de cuatro bytes (80386 y procesadores posteriores). En un salto hacia adelante, el ensamblador aún no ha encontrado el operando designado:

```
    JMP     A90

A90:
```

Ya que algunas versiones del ensamblador no saben en este punto si el salto es corto o cercano, generan de forma automática una instrucción de tres bytes. Sin embargo, estipulando que en realidad el salto es corto se puede utilizar el operador SHORT para forzar un salto corto y una instrucción de dos bytes codificando

```
    JMP  SHIRT A90

A90:
```

### Ejemplo de un programa que utiliza JMP

El programa .COM de la figura 8-1 ilustra el uso de la instrucción JMP. El programa inicializa los registros AX, BX y CX con el valor de 1, y un ciclo realiza lo siguiente:

```
                        page 60,132
                TITLE   P08JUMP (COM)  Uso de JMP para iterar
                        .MODEL SMALL
                        .CODE
0100                    ORG     100H
0100            MAIN    PROC    NEAR
0100  B8 0001           MOV     AX,01   |Iniciación de AX,
0103  BB 0001           MOV     BX,01   | BX y
0106  B9 0001           MOV     CX,01   | CX a 01
0109            A20:
0109  05 0001           ADD     AX,01   |Sumar 01 a AX
010C  03 D8             ADD     BX,AX   |Sumar AX a BX
010E  DI El             SHL     CX,1    |Multiplicar por dos a CX
0110  EB F7             JMP     A20     |Saltar a la etiqueta A20
0112            MAIN    ENDP
                        END     MAIN
```

Figura 8-1 Uso de la instrucción JMP

- Suma 1 a AX
- Suma AX a BX
- Duplica el valor en CX

Al final del ciclo, la instrucción JMP A20 transfiere el control a la instrucción etiquetada con A20. El efecto de repetir el ciclo hace que AX se incremente como 1,2,3,4,...; BX aumente de acuerdo a la suma de los primeros números naturales, obteniéndose 1, 3, 6, 10, ...; y CX se duplique como 1, 2, 4, 8, ... Ya que este ciclo no tiene salida, el procesamiento es infinito —por lo común no es una buena idea.

En el programa, A20 es -9 bytes desde el JMP. Puede confirmar esta distancia examinando el código objeto para JMP:EBF7. EB es el código de máquina para un JMP cercano y F7 hex es la notación en complemento a dos del -9. El IP contiene el desplazamiento (0112H) de la siguiente instrucción a ejecutarse. La operación JMP suma el F7 (técnicamente, FFF7, ya que el IP es de tamaño de una palabra) al IP, que contiene el desplazamiento 0112H de la siguiente instrucción al JMP:

| | DECIMAL | HEXADECIMAL | |
|---|---|---|---|
| Apuntador de instrucción: | 274 | 0112 | |
| Operando de JMP: | -9 | FFF7 | (complemento a dos) |
| Dirección de salto: | 265 | (1)0109 | |

La dirección de salto es calculada 0109H, en donde se ignora el acarreo externo de 1 (como lo muestra una revisión del listado del programa para la dirección de desplazamiento de A20). La operación cambia el valor del desplazamiento en el IP y salta la instrucción de la cola. Como éste es un salto hacia atrás, el operando FFF7 es negativo, mientras que para un salto hacia adelante será un valor positivo.

Como una experiencia útil, teclee el programa, ensámblelo, enlácelo y conviértalo a formato .COM. No se necesitan definiciones de datos, ya que los operandos inmediatos generan todos los datos. Utilice DEBUG para rastrear el módulo .COM para varias iteraciones. Una vez que el AX contenga 08, el BX y CX serán incrementados a 24H (36 decimal) y 80H (128 decimal), repectivamente. Teclee Q para salir de DEBUG.

## LA INSTRUCCIÓN LOOP

Como se mencionó en la figura 8-1, la instrucción JMP provoca un ciclo infinito. Pero es más probable que una rutina realice un ciclo un número específico de veces o hasta que se alcance una condición particular. La instrucción LOOP, que sirve para este propósito, requiere un valor inicial en el registro CX. En cada iteración, LOOP de forma automática disminuye 1 de CX. Si el valor en el CX es cero, el control pasa a la instrucción que sigue; si el valor en el CX no es cero, el control pasa a la dirección del operando. La distancia debe ser un salto corto, desde -128 hasta +127 bytes. Para una operación que exceda este límite, el ensamblador envía un mensaje como "salto relativo fuera de rango". El formato general para LOOP es

```
[etiqueta:]  LOOP  dirección corta
```

El programa en la figura 8-2 ilustra el uso de LOOP y realiza la misma operación que la del programa de la figura 8-1, salvo que termina después de 10 vueltas. Una instrucción MOV inicializa el CX con el valor 10. Como LOOP utiliza el CX, este programa usa ahora DX en lugar de CX para duplicar el valor inicial de 1. La instrucción LOOP reemplaza JMP A20 y, para un procesamiento más rápido, INC AX (incrementa el AX en 1) reemplaza ADD AX,01.

Igual que para JMP, el operando del código de máquina contiene la distancia desde el final de la instrucción LOOP a la dirección de A20, la cual es sumada al IP.

Como un ejercicio útil, modifique su copia de la figura 8-1 con estos cambios y ensamble, enlace y convierta el programa a .COM. Utilice DEBUG para rastrear a lo largo de los 10 ciclos. Una vez que CX es reducido a cero, los contenidos de AX, BX y DX son, 000BH, 0042H y 0400H, respectivamente. Presione Q para salir de DEBUG.

Existen dos variaciones de la instrucción LOOP, ambas también decrementan el CX en 1. LOOPE/LOOPZ (repite el ciclo mientras sea igual o repite el ciclo mientras sea cero) continúa el ciclo mientras que el valor en el CX es cero o la condición de cero está establecida. LOOPNE/LOOPNZ (repite el ciclo mientras no sea igual o repite el ciclo mientras sea cero) continúa el ciclo mientras el valor en el CX no es cero o la condición de cero no está establecida.

```
                        page 60,132
                TITLE   P08LOOP (COM)   Ilustración de LOOP
                        .MODEL SMALL
                        .CODE
0100                    ORG     100H
0100            BEGIN   PROC    NEAR
0100 B8 0001            MOV     AX, 01          Iniciar AX,
0103 BB 0001            MOV     BX, 01            BX, y
0106 BA 0001            MOV     DX, 01            DX con 01
0109 B9 000A            MOV     CX, 10          Iniciar
010C            A20:                              número de iteraciones
010C 40                 INC     AX              Sumar 01 a AX
010D 03 D8              ADD     BX, AX          Sumar AX a BX
010F DI E2              SHL     DX, 1           Multiplicar por dos a DX
0111 E2 F9              LOOP    A20             Decrementar CX,
                                                  iterar si es diferente de cero
0113 B8 4C00            MOV     AX,4C00H        Salida a DOS
0116 CD 21              INT     21H
0118            BEGIN   ENDP
                        END     BEGIN
```

Figura 8-2 Uso de la instrucción LOOP

Ni LOOP ni sus variantes LOOPxx afectan ninguna bandera en el registro de banderas, que serían cambiados por otras instrucciones dentro de la rutina del ciclo. Como consecuencia, si la rutina no tiene instrucciones que afecten la bandera ZF (cero) entonces el uso de LOOPNE/ LOOPNZ sería equivalente a usar LOOP.

## REGISTRO DE BANDERAS

El resto del material de este capítulo necesita de un conocimiento más detallado del registro de banderas. Este registro tiene 16 bits, los cuales varias instrucciones ponen a 1 para indicar el estado de una operación. En todos los casos, una bandera permanece en 1 hasta que otra instrucción lo cambia. El registro de banderas para modo real tiene los siguientes bits usados comúnmente:

| Bit no.: | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Señalizador: | | | | | 0 | D | I | T | S | Z | | A | | P | | C |

**CF (Bandera de acarreo).** Contiene un acarreo (0 o 1) del bit de orden alto (el más a la izquierda) después de operaciones aritméticas y algunas operaciones de corrimiento y rotación.

**PF (Bandera de paridad).** Contiene una verificación de los ocho bits de orden bajo de operaciones de datos. La bandera de paridad no debe ser confundida con el bit de paridad y rara vez interesa en programación convencional. Un número impar de bits en 1 limpian la bandera a cero (lo ponen en 0), y un número par de bits en 1 lo establecen en 1 (lo ponen en 1).

**AF (Bandera de acarreo auxiliar).** Tiene que ver con aritmética en campos ASCII y BCD empacados. Una operación que provoca un acarreo externo en el bit 3 (el cuarto bit desde la derecha) de un registro de un byte pone en 1 esta bandera.

**ZF (Bandera de cero).** Como resultado de una operación aritmética o de comparación, esta bandera se pone en 1 o en 0. De modo inesperado, un resultado no cero pone en 0 la bandera y un resultado cero lo pone en 1. Sin embargo, la configuración, que en apariencia no es correcta, es correcta lógicamente: 0 significa no (el resultado no es igual a cero) y 1 significa sí (el resultado es igual a cero). JE y JZ prueban esta bandera.

**SF (Bandera de signo).** Se establece de acuerdo con el signo (el bit de orden más alto o de más a la izquierda) después de una operación aritmética: Positivo pone la bandera en 0 y negativo lo pone en 1. JG y JL prueban esta bandera.

**TF (Bandera de trampa).** Cuando está en 1, hace que el procesador ejecute en modo de un solo paso, esto es, una instrucción a la vez bajo el control del usuario. Ya estableció esta bandera cuando ingresó el comando T en DEBUG, y ése es casi el único lugar en donde esperaría encontrar su uso.

**IF (Bandera de interrupción).** No permite interrupción cuando está en 0 y permite interrupción cuando está en 1. En programación convencional, esta bandera rara vez es utilizada.

**DF (Bandera de dirección).** Utilizado en operaciones de cadenas para determinar la dirección de transferencia de datos. Cuando la bandera es 0, la operación incrementa los registros SI y DI, haciendo que la transferencia de datos sea de izquierda a derecha; usando la bandera en 1, la operación decrementa el SI y DI haciendo que la transferencia de datos sea de derecha a izquierda.

**OF (Bandera de desbordamiento).** Indica un acarreo interno y uno externo en el bit de signo de alto orden (de más a la izquierda) después de una operación aritmética con signo.

## LA INSTRUCCIÓN CMP

La instrucción CMP por lo común es utilizada para comparar dos campos de datos, uno o ambos de los cuales están contenidos en un registro. El formato general para CMP es

```
[etiqueta:]  CMP  {registro/memoria},{registro/memoria/inmediato}
```

El resultado de una operación CMP afecta las banderas AF, CF, OF, PF, SF y ZF, aunque no tiene que probar estas banderas de forma individual. El código siguiente prueba el registro BX por un valor cero:

```
X       CMP    BX,00    ,-Compara BX con cero
        JZ     B50      ,-Si es cero salta a B50
               (acción si es diferente de cero)

B50:    ...             /Destino del salto, si BX es cero
```

Si el BX tiene cero, CMP establece el ZF a 1 y puede o no cambiar la configuración de otras banderas. La instrucción JZ (salta si es cero) sólo prueba la bandera ZF. Ya que ZF tiene 1 (que significa una condición cero), JZ transfiere el control (salta) a la dirección indicada por el operando B50.

Observe que la operación compara el primer operando con el segundo; por ejemplo, ¿el valor del primer operando es mayor que, igual a o menor que el valor del segundo operando? La sección siguiente porporciona las diferentes formas de transferencia de control con base en condiciones probadas.

## INSTRUCCIONES DE SALTO CONDICIONAL

El ensamblador permite usar una variedad de instrucciones de salto condicional que transfieren el control dependiendo de las configuraciones en el registro de banderas. Por ejemplo, puede comparar dos campos y después saltar de acuerdo con los valores de las banderas que la comparación establece. El formato general para el salto condicional es

```
[etiqueta:]  Jnnn  dirección corta
```

Como ya se explicó, la instrucción LOOP disminuye el registro CX; si es diferente de cero, transfiere el control a la dirección del operando. Podría reemplazar el enunciado LOOP A20 de la figura 8-2 con dos enunciados —uno que decremente el CX y otro que realice un salto condicional:

```
DEC     CX          ;Equivalente a LOOP
JNZ     A20
```

DEC y JNZ realizan exactamente lo que hace LOOP. DEC decrementa el CX en 1 y pone a 1 o a 0 la bandera de cero (ZF) en el registro de banderas. Después JNZ prueba la configuración de la bandera de cero; si el CX es diferente de cero, el control pasa a A20, y si el CX es cero el control pasa a la siguiente instrucción hacia abajo. (La operación de salto también brinca la línea de espera cola de instrucción de prebúsqueda del procesador.) Aunque LOOP tiene usos limitados, en este ejemplo es más eficaz que el uso de las instrucciones DEC y JNZ.

Al igual que para JMP y LOOP, el operando en código de máquina contiene la distancia desde el final de la instrucción JNZ a la dirección de A20, la cual es sumada al apuntador de instrucción. Para el 8086/286, la distancia debe ser un salto corto, desde -128 hasta +127 bytes. Si una operación excede este límite, el ensamblador envía un mensaje "salto relativo fuera de rango". El 80386 y procesadores posteriores proporcionan desplazamientos de 8 bits (corto) o de 32 bits (cercano) que permiten alcanzar cualquier dirección dentro del segmento.

### Datos con signo y sin signo

Distinguir el propósito de los saltos condicionales debe clarificar su uso. El tipo de datos (sin signo o con signo) sobre los que se realizan las comparaciones o la aritmética puede determinar cuál es la instrucción a utilizar. Un dato *sin signo* trata todos los bits como bits de datos; ejemplos típicos son las cadenas de caracteres, tal como nombres o direcciones, y valores numéricos tal como números de cliente. Un dato *con signo* trata el bit de más a la izquierda como un signo, en donde 0 es positivo y 1 es negativo. Muchos valores numéricos pueden ser positivo o negativo.

En el ejemplo siguiente, el AX contiene 11000110 y el BX contiene 00010110. La siguiente instrucción

```
CMP AX,BX
```

compara el contenido del AX con el contenido del BX. Para datos sin signo, el valor AX es mayor; sin embargo, para datos con signo el valor AX es menor a causa del signo negativo.

### Saltos con base en datos sin signo

Las instrucciones siguientes de salto condicional se aplican a datos sin signo:

| SÍMBOLO | DESCRIPCIÓN | BANDERA EXAMINADA |
|---|---|---|
| JE/JZ | Salta si es igual o salta si es cero | ZF |
| JNE/JNZ | Salta si no es igual o salta si no es cero | ZF |
| JA/JNBE | Bifurca si es mayor o salta si no es menor o igual | CF, ZF |
| JAE/JNB | Salta si es mayor o igual o salta si no es menor | CF |
| JB/JNAE | Salta si es menor o salta si no es mayor o igual | CF |
| JBE/JNA | Salta si es menor o igual o salta si no es mayor | CF, AF |

Cada una de estas pruebas las puede expresar en uno de dos códigos simbólicos de operación. Seleccione aquel que sea más claro y más descriptivo. Por ejemplo, aunque JB y JNAE generan el mismo código objeto, la prueba afirmativa JB es más fácil de entender que la prueba negativa JNAE.

### Saltos con base en datos con signo

Las instrucciones siguientes de salto condicional se aplican a datos con signo:

| SÍMBOLO | DESCRIPCIÓN | BANDERA EXAMINADA |
|---|---|---|
| JE/JZ | Salta si es igual o salta si es cero | ZF |
| JNE/JNZ | Salta si no es igual o salta si no es cero | ZF |
| JG/JNLE | Salta si es mayor o salta si no es menor o igual | ZF, SF, OF |
| JGE/JNL | Salta si es mayor o igual o salta si no es menor | SF, OF |
| JL/JNGE | Salta si es menor o salta si no es mayor o igual | SF, OF |
| JLE/JNG | Salta si es menor o igual o salta si no es mayor | ZF, SF, OF |

Los saltos para la prueba de igual o cero (JE/JZ) y para la prueba de no igual o cero (JNE/JNZ) están incluidos en las listas de datos sin signo y datos con signo, ya que una condición de igual o cero ocurre sin importar la presencia de signo.

### Pruebas aritméticas especiales

Las siguientes instrucciones de salto condicional tienen usos especiales:

| SÍMBOLO | DESCRIPCIÓN | BANDERA PROBADA |
|---|---|---|
| JS | Salta si el signo es negativo | SF |
| JNS | Salta si el signo es positivo | SF |
| JC | Salta si hay acarreo (igual que JB) | CF |
| JNC | Salta si no hay acarreo | CF |
| JO | Salta si hay desbordamiento | OF |
| JNO | Salta si no hay desbordamiento | OF |
| JP/JPE | Salta si hay paridad o salta si la paridad es par | PF |
| JNP/JPO | Salta si no hay paridad o salta si la paridad es impar | PF |

JC y JNC con frecuencia son usados para probar el éxito de operaciones en disco. Otro salto condicional, JCXZ, prueba el contenido del registro CX contra cero. Esta instrucción no necesita ser colocada inmediatamente después de una operación aritmética o de comparación. Un uso de JCXZ podría ser al inicio de un ciclo, para asegurarse de que en realidad CX contiene un valor diferente de cero.

No espere memorizar todas estas instrucciones; sin embargo, como recordatorio note que un salto para datos *sin signo* es igual, superior o inferior, mientras que un salto para datos *con signo* es igual, mayor que o menor. Los saltos que prueban las banderas de acarreo, de desbordamiento y de paridad tienen propósitos únicos. El ensamblador traduce el código simbólico en

código objeto, sin importar qué instrucción utilice, pero, por ejemplo JAE y JGE aunque en apariencia son similares, no prueban las mismas banderas.

El 80386 y procesadores posteriores permiten saltos condicionales lejanos. Puede indicar un salto corto o lejano, por ejemplo,

```
JNE  dirección corta  (SHORT)
JAE  dirección lejana  (FAR)
```

## LLAMADA A PROCEDIMIENTOS

Hasta ahora los segmentos de código han consistido sólo en un procedimiento, codificado como

```
BEGIN  PROC  FAR


BEGIN  ENDP
```

En este caso el operando FAR informa al sistema que la dirección indicada es el punto de entrada para la ejecución del programa, mientras que la directiva ENDP define el final del procedimiento. Sin embargo, un segmento de código puede tener cualquier número de procedimientos, todos distinguidos por PROC y ENDP. Un procedimiento llamado (o subrutina) es una sección de código que realiza una tarea definida y clara (tal como ubicar el cursor o bien obtener entrada del teclado). La organización de un programa en procedimientos proporciona los beneficios siguientes:

- Reduce la cantidad de código, ya que un procedimiento común puede ser llamado desde cualquier lugar en el segmento de código.
- Fortalece la mejor organización del programa.
- Facilita la depuración del programa, ya que los errores pueden ser aislados con mayor claridad.
- Ayuda en el mantenimiento progresivo de programas, ya que los procedimientos son identificados de forma rápida para su modificación.

### Operaciones CALL y RET

La instrucción CALL transfiere el control a un procedimiento llamado, y la instrucción RET regresa del procedimiento llamado al procedimiento original que hizo la llamada. RET debe ser la última instrucción en un procedimiento llamado. Los formatos generales para CALL y RET son:

| | | |
|---|---|---|
| [etiqueta:] | CALL | procedimiento |
| [etiqueta:] | RET | [inmediato] |

El código objeto particular que CALL y RET generan depende de si la operación implica un procedimiento NEAR (cercano) o un procedimiento FAR (lejano).

Llamada y regreso cercanos. Una llamada (CALL) a un procedimiento dentro del mismo segmento es cercana y realiza lo siguiente:

- Disminuye el SP en 2 (una palabra).
- Mete el IP (que contiene el desplazamiento de la instrucción que sigue al CALL) en la pila.
- Inserta la dirección del desplazamiento del procedimiento llamado en el IP (esta operación vacía el resultado de la instrucción previamente procesada).

Un RET que regresa desde un procedimiento cercano realiza lo siguiente:

- Saca el antiguo valor de IP de la pila y lo envía al IP (lo cual también vacía el resultado de la instrucción previamente procesada).
- Incrementa el SP en 2.

Ahora el CS:IP apunta a la instrucción que sigue al CALL original en la llamada del procedimiento, en donde se reasume la ejecución.

**Llamada** y **regreso lejanos.** Una llamada (CALL) lejana llama a un procedimiento etiquetado con FAR, tal vez en un segmento de código separado. Un CALL lejano mete a la pila al CS y al IP, y RET los saca de la pila. Las llamadas y regresos lejanos son tema del capítulo 23.

### Ejemplo de una llamada y regreso cercanos

Una organización común de llamadas y regreso cercanos aparece en la figura 8-3. Advierta las características siguientes:

- El programa está dividido en un procedimiento lejano, BEGIN y dos procedimientos cercanos, B10 y CIO. Cada procedimiento tiene un nombre único y contiene su propio ENDP para finalizar su definición.

```
                        page 60,132
                TITLE   P08CALLP (EXE)   Llamada a procedimientos
                        .MODEL   SMALL
                        .STACK   64
                        .DATA

                        .CODE
0000            BEGIN   PROC     FAR
0000 E8 0008 R          CALL     B10             /Llamada a B10

0003 B8 4C00            MOV      AX,4C00H        ,-Salida a DOS
0006 CD 21              INT      21H
0008            BEGIN   ENDP
-------------------------------------------------------------------
0008            B10     PROC     NEAR
0008 E8 000C R          CALL     CÍO             /Llamada a CÍO

000B C3                 RET                      /De regreso a
000C            B10     ENDP                     /  quien llama
-------------------------------------------------------------------
000C            CÍO     PROC     NEAR

000C C3                 RET                      /De regreso a
000D            CÍO     ENDP                        quien llama
-------------------------------------------------------------------
                        END      BEGIN
```

Figura 8-3 Llamada a procedimientos

- Las directivas PROC para B10 y CIO tienen el atributo NEAR para indicar que estos procedimientos están dentro del segmento de código actual. Puesto que la omisión del atributo hace que el ensamblador por omisión tome NEAR, muchos ejemplos subsiguientes lo omiten.
- En el procedimiento BEGIN, la instrucción CALL transfiere el control del programa al procedimiento B10 e inicia su ejecución.
- En el procedimiento B10, la instrucción CALL transfiere el control al procedimiento CIO e inicia su ejecución.
- En el procedimiento CIO, la instrucción RET hace que el control regrese a la instrucción que sigue a CALL CIO.
- En el procedimiento B10, la instrucción RET hace que el control regrese la instrucción que sigue a CALL B10.
- Entonces el procedimiento BEGIN reasume el procesamiento desde ese punto.
- RET siempre regresa a la rutina que llama. Si B10 no termina con una instrucción RET, las instrucciones se ejecutarían pasando B10 e irían directamente a CIO. De hecho, si CIO no contiene un RET el programa ejecutaría, pasando el final de CIO, todas las instrucciones (si hay) que estuvieran ahí, con resultados impredecibles.

Técnicamente, puede transferir el control a un procedimiento cercano por medio de una instrucción de salto o incluso por código normal en línea. Pero por claridad y consistencia, utilice CALL para transferir el control a un procedimiento y utilice RET para terminar la ejecución de un procedimiento.

## EFECTOS EN LA PILA DE LA EJECUCIÓN DE PROGRAMAS

Hasta este punto nuestros programas han tenido poca necesidad de meter datos en la pila y, en consecuencia, hemos definido una pila muy pequeña. Sin embargo, una llamada a procedimiento puede llamar (CALL) a otro procedimiento, el cual a su vez aun puede llamar (CALL) a otro procedimiento, de manera que la pila debe ser los suficientemente grande para contener las direcciones guardadas. Todo esto llega a ser más fácil de lo que parece a primera vista, y para la mayoría de nuestros propósitos una definición de la pila de 32 palabras es suficiente.

CALL y PUSH almacenan una dirección de una palabra o valor en la pila. RET y POP sacan de la pila y accesan la palabra previamente guardada. Todas estas operaciones cambian la dirección del desplazamiento en el rigistro de SP para la palabra siguiente. A causa de esta característica, las operaciones RET y POP deben coincidir con sus operaciones originales CALL y PUSH.

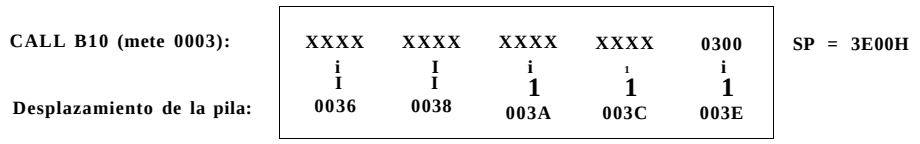
Como un recordatorio, al cargar un programa .EXE para ejecución el cargador de sistema establece los valores siguientes en los registros:

- DS y ES: La dirección del PSP, un área de 256 bytes (100H) que precede un módulo de programa ejecutable en memoria.
- CS: La dirección del segmento de código —el punto de entrada a su programa.
- IP: Cero, si la primera instrucción ejecutable está en el inicio del segmento de código.
- SS: La dirección del segmento de la pila.
- SP: Desplazamiento del tope de la pila. Por ejemplo, para una pila definida como .STACK 64 (64 bytes o 32 palabras), en un inicio SP contiene 64, o 40H.

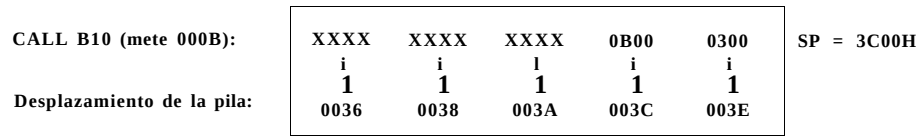
Rastree el programa sencillo de la figura 8-3 a lo largo de su ejecución. En la práctica, las llamadas procedimientos tendrían cualquier número de instrucciones.

La localidad disponible actual para agregar o remover es el tope de la pila. Para este ejemplo, el cargador tendría establecido el SP al tamaño de la pila, 64 bytes (40H). El programa realiza las operaciones siguientes:

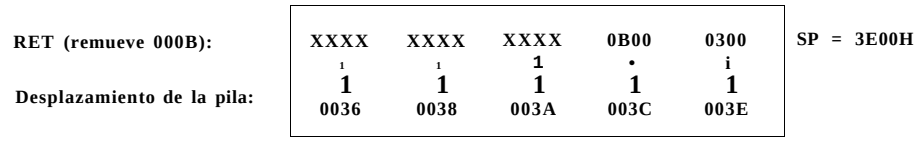
- CALL B10 disminuye en 2 el SP, de 40H a 3EH. Después mete el IP (con 0003) en el tope de la pila en el desplazamiento 3EH. Éste es el desplazamiento de la instrucción que sigue a la instrucción CALL. El procesador utiliza la dirección formada por CS:IP para transferir el control a B10. Las palabras en memoria contienen bytes en orden inverso; por ejemplo 0003 se convierte en 0300.



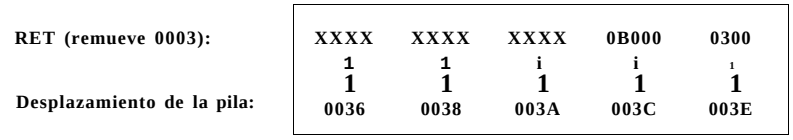

- En el procedimiento B10, CALL CÍO disminuye en 2 el SP, a 3CH. Después agrega el IP (con 000B) en el tope de la pila en el desplazamiento 3CH. El procesador utiliza las direcciones CS:IP para transferir el control a CÍO.




- Para regresar de CÍO, la instrucción RET remueve el desplazamiento (000B) del tope de la pila a 3CH, la inserta en el IP e incrementa el SP en 2 a 3EH. Esto provoca un regreso automático al desplazamiento 000BH en el procedimiento B10.




- El RET al final del procedimiento B10 saca la dirección (0003) del tope de la pila en 3EH y la envía al IP e incrementa en 2 el SP a 40H. Esto provoca un regreso automático al desplazamiento 0003H, en donde el programa termina su ejecución.




Si utiliza DEBUG para ver la pila, puede encontrar datos irrelevantes a la izquierda de un programa previamente ejecutado.

## OPERACIONES BOOLEANAS

La lógica booleana es importante en el diseño de circuitos y tiene un paralelo en la lógica de programación. Las instrucciones para lógica booleana son AND, OR, XOR, TEST y NOT, que pueden usarse para poner bits en 0 o 1 y para manejar datos ASCII con propósitos aritméticos (capítulo 13). El formato general para las operaciones booleanas es

```
[etiqueta:]  operación  {registro/memoria},{registro/memoria/inmediato}
```

El primer operando se refiere a un byte o palabra en un registro o memoria y es el único valor que es cambiado. El segundo operando hace referencia a un registro o a un valor inmediato. La operación compara los bits de los dos operandos referenciados y de acuerdo con esto establece las banderas CF, OF, PF, SF y ZF (AF está indefinido).

- AND. Si ambos bits comparados son 1, establece el resultado en 1. Las demás condiciones dan como resultado 0.
- OR. Si cualquiera (o ambos) de los bits comparados es 1, el resultado es 1. Si ambos bits están en 0, el resultado es 0.
- XOR. Si uno de los bits comparados es 0 y el otro 1, el resultado es 1. Si ambos bits comparados son iguales (ambos 0 o ambos 1), el resultado es 0.
- TEST. Establece las banderas igual que lo hace AND, pero no cambia los bits de los operandos.

Las operaciones siguientes AND, OR y XOR ilustran los mismos valores de bits como operandos:

| | AND | OR | XOR |
|---|---|---|---|
| | 0101 | 0101 | 0101 |
| | 0011 | 0011 | 0011 |
| Resultado: | 0001 | 0111 | 0110 |

Es útil recordar la siguiente regla: el empleo de AND con bits 0 es 0 y el de OR con bits 1 es 1.

### Ejemplos de operaciones booleanas

Para los siguientes ejemplos independientes, suponga que el AL contiene 1100 0101 y el BH contiene 0101 1100:

```
1.  AND   AL, BH      ;Establece AL a 0100 0100
2.  AND   AL,00H      ;Establece AL a 0000 0000
3.  AND   AL,0FH      ;Establece AL a 0000 0101
4.  OR    BH, AL      ;Establece BH a 1101 1101
5.  OR    CL, CL      ,-Pone en uno SF y ZF
6.  XOR   AL, AL      ;Establece AL a 0000 0000
7.  XOR   AL,0FFH     ;Establece AL a 0011 1010
```

Los ejemplos 2 y 6 muestran formas de limpiar un registro, y ponerlo a cero. El ejemplo 3 pone a cero los cuatro bits más a la izquierda de AL. Aunque el uso de CMP puede ser más claro, puede utilizar OR para los siguientes fines:

```
1. OR   CX,CX       ¿Verifica CX contra cero
   JZ   ...         ,•Salta si es cero
2. OR   CX,CX       /Verifica el signo de CX
   JS   ...          Salta si es negativo
```

TEST actúa igual que AND, pero sólo establece las banderas. Aquí están algunos ejemplos:

```
1.  TEST   BL,11110000B      ;¿Alguno de los bits de más a la
    JNZ                      r  izquierda es BL en diferente de cero?
2.  TEST   AL,00000001B      ;¿AL contiene
    JNZ                      ;  un número impar?
3.  TEST   DX,OFFH           ;¿El DX contiene
    JZ                       ;  un valor cero?
```

### La instrucción NOT

La instrucción NOT sólo invierte los bits en un byte o palabra en un registro o en memoria: esto es, convierte los ceros en unos y los unos en ceros. El formato general para NOT es

```
[etiqueta:]  NOT  {registro/memoria}
```

Por ejemplo, si el AL contiene 1100 0101, la instrucción NOT AL cambia el AL a 0011 1010 (el resultado es el mismo de XOR AL,OFFH del anterior ejemplo 7). Las banderas no son afectadas. NOT no es lo mismo que NEG, que cambia un valor binario de positivo a negativo y viceversa, inviniendo los bits y sumando 1.

## CAMBIO DE MINÚSCULAS A MAYÚSCULAS

Existen varias razones para realizar la conversión entre letras mayúsculas y minúsculas. Por ejemplo, puede haber recibido un archivo de datos de un sistema que procesa sólo letras mayúsculas. O un programa tiene que permitir a los usuarios ingresar de forma indistinta mayúsculas o minúsculas (como 'YES' o 'yes') y para facilitar las comparaciones, convertirlas a mayúsculas. Las letras mayúsculas de A a la Z son desde 41H hasta 5AH, y las letras minúsculas de a hasta la z son desde 61H a 7AH. La única diferencia es que el bit 5 de una mayúscula está en 0 y para minúsculas está en 1, como se muestra a continuación:

| | MAYÚSCULAS | | MINÚSCULAS |
|---|---|---|---|
| Letra A: | 01000001 | Letra a: | 01100001 |
| Letra Z: | 01011010 | Letra z: | 01111010 |
| Bit: | 76543210 | Bit: | 76543210 |

```
TITLE   P08CASE (COM)   Cambio de minúsculas a mayúsculas
        .MODEL SMALL
        .CODE
        ORG    100H
BEGIN:  JMP    MAIN

TITLEX  DB     'Change to uppercase letters'

MAIN    PROC   NEAR
        LEA    BX.TITLEX+l        Primer carácter a cambiar
        MOV    CX,26             ,-No. de caracteres a cambiar
B20:
        MOV    AH,[BX]           |Carácter de TITLEX
        CMP    AH,61H            |Es
        JB     B30               |  letra
        CMP    AH, 7AH           |  minúscula
        JA     B30               |  ¿letra?
        AND    AH,11011111B      |Si - convertirla
        •MOV    [BX], AH         |Restaurar en TITLEX
B30:
        INC    BX                |Establecer para siguiente carácter
        LOOP   B20               |Iterar 26 veces
        MOV    AX,4C00H •        |Hecho -- salida
        INT    21H
MAIN    ENDP
        END     BEGIN
```

Figura 8-4 Cambio de minúsculas a mayúsculas

El programa .COM de la figura 8-4 convierte el contenido de un dato, TITLEX, de minúscula a mayúscula, empezando en TITLEX+ 1. El programa inicializa el BX con la dirección de TITLEX +1 y utiliza la dirección para mover cada carácter al AH, iniciando en TITLEX + 1. Si el valor está entre 61H y 7AH, una instrucción AND establece el bit 5 en 0:

```
AND  AH,11011111B
```

Todos los demás caracteres distintos de a hasta z permanecen sin cambio. Después la rutina mueve el carácter cambiado de regreso a TITLEX, incrementa el BX para el siguiente carácter y repite el ciclo.

Usado de esta manera, el registro BX funciona como un registro de índice para las localidades de memoria direccionadas. Para el mismo fin, también se puede utilizar SI y DI.

## CORRIMIENTO DE BITS

Las instrucciones de corrimiento, que son parte de la capacidad lógica de la computadora, pueden realizar las siguientes acciones:

- Hacer referencia a un registro o dirección de memoria.
- Recorre bits a la izquierda o a la derecha.
- Recorre hasta 8 bits en un byte, 16 bits en una palabra y 32 bits en una palabra doble (80386 y procesadores posteriores).
- Corrimiento lógico (sin signo) o aritmético (con signo).

El segundo operando contiene el valor del corrimiento, que es una constante (un valor inmediato) o una referencia al registro CL. Para los procesadores 8088/8086, la constante inme-

diata sólo puede ser 1; un valor de corrimiento mayor que 1 debe estar contenido en el registro Cl. Procesadores posteriores permiten constantes de corrimiento inmediato hasta de 31. El formato general para el corrimiento es

```
[etiqueta:]          (registro/memoria),{CL/inmediato)
```

### Corrimiento de bits hacia la derecha

Los corrimientos hacia la derecha (SHR y SAR) mueven los bits hacia la derecha en el registro designado. El bit recorrido fuera del registro mete la bandera de acarreo. Las instrucciones de corrimiento a la derecha estipulan datos lógicos (sin signo) o aritméticos (con signo):

**SHR: desplazamiento lógico a la derecha** o —»

**SAR: desplazamiento aritmético a la derecha** —>

Las siguientes instrucciones relacionadas ilustran SHR y datos sin signo:

```
INSTRUCCIÓN              AL                COMENTARIO
MOV   CL,03
MOV   AL,10110111B     10110111
SHR   AL,01            01011011   Un corrimiento a la derecha
SHR   AL,CL            00001011   Tres corrimientos adicionales a la derecha
SHR   AX,03            Válido para 80186 y procesadores posteriores
```

El primer SHR desplaza el contenido de AL un bit hacia la derecha. El bit de más a la derecha es enviado a la bandera de acarreo, y el bit de más a la izquierda se llena con un cero. El segundo SHR desplaza tres bits más al AL. La bandera de acarreo contiene de manera sucesiva 1, 1 y 0; además, tres bits 0 son colocados a la izquierda del AL.

SAR difiere de SHR en un punto importante: SAR utiliza el *bit de signo* para llenar el bit vacante de más a la izquierda. De esta manera, los valores positivos y negativos retienen sus signos. Las siguientes instrucciones relacionadas ilustran SAR y datos con signo en los que el signo es un bit 1:

```
INSTRUCCIÓN              AL                COMENTARIO
MOV  CL,03
MOV AL,10110111B     ; 10110111
SAR AL,01            ; 11011011    Un corrimiento a la derecha
SAR AL,CL            ; 11111011    Tres corrimientos a la derecha
SAR AX,03            ; Para 80186 y procesadores posteriores
```

En especial, los corrimientos a la derecha son útiles para (dividir entre dos) *obtener mitades* de valores y son mucho más rápidas que utilizar una operación de división. En los ejemplos de corrimientos de tres bits a la derecha, el primer corrimiento de un bit a la derecha en realidad divide entre dos, y el segundo y tercer corrimientos a la derecha en realidad dividen entre 8.

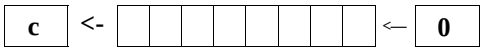
Al obtener la mitad de números impares tales como 5 y 7 se genera 2 y 3, respectivamente, y la bandera de acarreo se pone en 1. También, si tiene que desplazar dos bits, es más eficaz la codificación de dos instrucciones de corrimiento que almacenar 2 en el CL y codificar un corrimiento.

Al terminar una operación de corrimiento, puede utilizar la instrucción JC (salta si hay acarreo) para examinar el bit desplazado a la bandera de acarreo.

### Corrimiento de bits hacia la izquierda

Los corrimientos hacia la izquierda (SHL y SAL) mueven los bits a la izquierda, en el registro designado. SHL y SAL son idénticos en su operación. El bit desplazado fuera del registro ingresa a la bandera de acarreo. Las instrucciones de corrimiento hacia la izquierda estipulan datos lógicos (sin signo) y aritméticos (con signo):

SHL: desplazamiento lógico a la izquierda SAL: desplazamiento aritmético a la izquierda




Las siguientes instrucciones relacionadas ilustran SHL para datos sin signo:

```
INSTRUCCIÓN          AL                COMENTARIO
MOV  CL,03
MOV AL,10110111B    ; 10110111
SHL AL,01           ; 01101110    Un corrimiento a la izquierda
SHL AL,CL           ; 01110000    Tres corrimientos más
SHL AX,03           ; Para 80186 y procesadores posteriores
```

El primer SHL desplaza el contenido del AL un bit hacia la izquierda. El bit desplazado de más a la izquierda ahora se encuentra en la bandera de acarreo, y el último bit a la derecha del AL se llena con cero. El segundo SHL desplaza tres bits más el AL. La bandera de acarreo contiene en forma sucesiva 0, 1 y 1, y se rellena con tres ceros a la derecha del AL.

Los corrimientos a la izquierda llenan con cero el bit de más a la derecha. Como resultado de esto, SHL y SAL son idénticos. Los corrimientos a la izquierda en especial son útiles para duplicar valores y son mucho más rápidos que usar una operación de multiplicación. En los ejemplos de la operación de corrimiento a la izquierda, el primer corrimiento de un bit a la izquierda en realidad multiplica por 2, y el segundo y tercer corrimientos de tres bits a la izquierda en realidad multiplican por 8. También, si tiene que desplazar dos bits, es más eficaz la codificación de dos instrucciones de corrimiento que almacenar 2 en el CL y codificar un corrimiento.

Al finalizar una operación de corrimiento, puede utilizar la instrucción JC (salta si hay acarreo) para examinar el bit que ingresó a la bandera de acarreo.

## ROTACIÓN DE BITS (desplazamiento circular)

Las instrucciones de rotación, que son parte de la capacidad lógica de la computadora, pueden realizar las siguientes acciones:

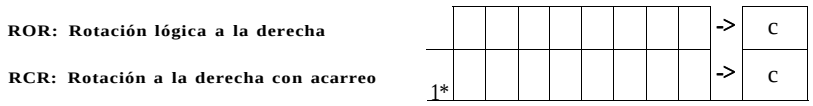
- Hacer referencia a un byte o a una palabra.
- Hacer referencia a un registro o a memoria.
- Realizar rotación a la derecha o a la izquierda. El bit que es desplazado fuera llena el espacio vacante en la memoria o registro y también se copia en la bandera de acarreo. Véanse las figuras de las dos secciones siguientes.
- Realizar rotación hasta de 8 bits en un byte, 16 bits en una palabra y 32 bits en una palabra doble (80386 y procesadores posteriores).
- Realizar rotación lógica (sin signo) o aritmética (con signo).

El segundo operando contiene un valor de rotación, el cual es una constante (un valor inmediato) o una referencia al registro CL. Para los procesadores 8088/8086, la constante inmediata sólo puede ser 1; un valor de rotación mayor que 1 debe estar en el registro CL. Procesadores más recientes permiten constantes inmediatas hasta el 31. El formato general para la rotación es:

```
[etiqueta:]   rotación   {registro/memoria},{CL/inmediato}
```

### Rotación a la derecha de bits

Las rotaciones a la derecha (ROR y RCR) desplazan a la derecha los bits en el registro designado. Las instrucciones de rotación a la derecha estipulan datos lógicos (sin signo) o aritméticos (con signo):




Las siguientes instrucciones relacionadas ilustran ROR:

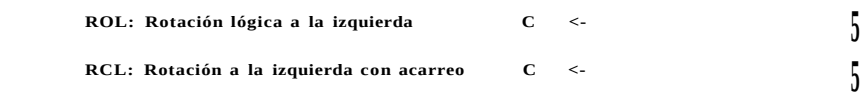
```
INSTRUCCIÓN              BH              COMENTARIO
MOV CL,03
MOV  BH,10110111B      10110111
RORBH,01               11011011    Una rotación a la derecha
ROR  BH,CL             01111011    Tres rotaciones a la derecha
ROR  BX,03             Para 80186 y procesadores posteriores
```

El primer ROR desplaza el bit de más a la derecha del BH a la posición vacante de más a la izquierda. La segunda y tercera operaciones ROR realizan la rotación de los tres bits de más a la derecha.

RCR provoca que la *bandera de acarreo* participe en la rotación. Cada bit que se desplaza fuera por la derecha se mueve al CF y el bit del CF se mueve a la posición vacante de la izquierda.

### Rotación a la izquierda de bits

Las rotaciones a la izquierda (ROL y RCL) desplazan a la izquierda los bits del registro designado. Las instrucciones de rotación a la izquierda estipulan datos lógicos (sin signo) y aritméticos (con signo):



Las siguientes instrucciones relacionadas ilustran ROL:

```
INSTRUCCIÓN          BL              COMENTARIO
MOV  CL,03
MOV BL,10110111B     ; 10110111
ROR BL,01            ; 11011011    Una rotación a la izquierda
ROR BL,CL            ; 01111011    Tres rotaciones a la izquierda
ROR BX,03            ; Para 80186 y procesadores posteriores
```

El primer ROL desplaza el bit de más a la izquierda del BL a la posición vacante de más a la derecha. La segunda y tercera operaciones ROL realizan la rotación de los tres bits de más a la izquierda.

De manera similar a RCR, RCL también provoca que la bandera de acarreo participe en la rotación. Cada bit que se desplaza fuera por la izquierda se mueve al CF, y el bit del CF se mueve a la posición vacante de la derecha.

Puede usar la instrucción JC (salte si hay acarreo) para comprobar el bit rotado hacia la CF en el extremo de una operación de rotación.

### Desplazamiento y rotación de palabras dobles

También puede utilizar las instrucciones de rotación y para desplazar a fin de multiplicar y dividir entre múltiplos de 2, valores en palabras dobles. Considere un valor en 32 bits en el que los 16 bits de más a la izquierda están en el DX y los 16 bits de más a la derecha están en el AX, como DX:AX. Las instrucciones para "multiplicar" ese valor por dos podría ser:

```
SHL  AX, 1      ;Usa desplazamiento a la izquierda para
RCL  DX, 1      ; multiplicar por dos el par DX:AX
```

EL SHL desplaza a la izquierda todos los bits del AX, y el bit de más a la izquierda lo envía a la bandera de acarreo. El RCL desplaza el DX a la izquierda e inserta el bit del CF en el bit vacante de más a la derecha. Para multiplicar por 4, haga seguir a la pareja SHL-RCL por otra pareja SHL-RCL.

Para división, otra vez considere un valor en 32 bits en DX:AX. Las instrucciones para "dividir" entre dos el valor serían

```
SAR  DX,1       ,-Usa desplazamiento a la derecha para
RCR  AX, 1      ,• dividir entre dos el par DX:AX
```

Para dividir entre cuatro, haga seguir a la pareja SAR-RCR por otra pareja SAR-RCR.

Los desplazamientos de doble precisión para el 80386 y procesadores posteriores son SHRD y SHLD.

## TABLAS DE BIFURCACIÓN

Un programa puede tener una rutina para probar varias condiciones relacionadas, de las que cada una necesita un salto a otra rutina. Por ejemplo, considere un sistema para una compañía que ha establecido códigos especiales para los clientes con base en su nivel de crédito y volumen de ventas. Los códigos indican la cantidad de descuento ofrecido y otros procesos especiales que pueden necesitarse para el cliente. Los códigos de los clientes son 0, 1, 2, 3 y 4.

Una manera convencional de manejar códigos es comparar de manera sucesiva contra cada código de cliente:

```
CMP    CUSCODE,0    /¿Código = 0?
JE     DOODSCT
CMP    CUSCODE,1    ;¿Código = 1?
JE     D10DSCT
CMP    CUSCODE,2    ¡¿Código = 2?
JE     D20DSCT
CMP    CUSCODE,3    ;¿Código = 3?
JE     D30DSCT
CMP    CUSCODE,4    ;¿Código = 4?
JE     D40NSCT
```

Con este enfoque, es grande la ocasión para errores: sólo considera la comparación de los códigos correctos contra sus valores y salta a la rutina correcta. Una solución más elegante involucra una tabla de direcciones de salto. Como se muestra en el programa parcial de la figura 8-5, CUSTTBL define de manera sucesiva las cinco direcciones en palabras (dos bytes cada una). La rutina en D10JUMP accesa los códigos (como valores hexadecimales 00-04) en el registro BX. El valor es duplicado, de manera que 0 permanece como 0, 1 se convierte en 2, 2 se convierte en 4, y así sucesivamente. El valor duplicado proporciona un desplazamiento en la tabla: CUSTTBL4-0 es la primera dirección, CUSTTBL+2 es la segunda, CUSTTBL+4 es la tercera, y así sucesivamente. El operando de la instrucción JMP, [CUSTTBL+BX], forma una dirección con base en el inicio de la tabla más un desplazamiento en la tabla. Después la operación salta de manera directa a la rutina apropiada.

Una restricción importante en el programa es que los códigos sólo pueden ser valores hexadecimales 00-04; ¡cualquier otro valor causaría terribles resultados! Si utiliza DEBUG para ejecutar este programa, para verificar el resultado de la lógica ingrese valores hexadecimales válidos (00-04) en CUSCODE.

Para el 80386 y procesadores posteriores podría reemplazar las dos instrucciones en D10JUMP, esto es;

```
MOV     BL,CUSCODE      ¡Obtiene el código de descuento
XOR     BH,BH           ¡Limpia la parte superior de BX
```

con una instrucción:

```
MOVZX     BX,CUSCODE     ¡Obtiene código de descuento
```

## ORGANIZACIÓN DE UN PROGRAMA

Lo siguiente son los pasos comunes al escribir un programa en lenguaje ensamblador:

1. Tenga una idea clara del problema que el programa va a resolver.
2. Esboce sus ideas en términos generales y planee la lógica general. Por ejemplo, si un problema es examinar las operaciones de movimiento de múltiples bytes, inicie definiendo los campos

```
                        PAGE   60,132
                TITLE   P08JMPTB (EXE)  Uso de una tabla de saltos
                        .MODEL SMALL
                        .STACK 64
                ;-------------------------------------------------
                        .DATA
0000 001B R     CUSTTBL DW     DOONODSC   ;Tabla de direcciones
0002 001E R             DW     D10DSCT
0004 0021 R             DW     D20DSCT
0006 0024 R             DW     D30DSCT
0008 0027 R             DW     D40DSCT
000A 04         CUSC0DE DB     04         ;Código de descuento
                ;-------------------------------------------------
                        .CODE
0000            BEGIN   PROC   FAR
0000 B8_____R           MOV    AX,©data      ;Iniciar
0003 8E D8              MOV    DS,AX           registros
0005 8E CO              MOV    ES,AX         ;  de segmento

0007 E8 OOOF R          CALL   D10JUMP       ;Invocar rutina de sal

000A B8 4C00            MOV    AX,4C00H      ,•Salida a dos
OOOD CD 21              INT    21H
OOOF            BEGIN   ENDP

OOOF            D10JUMP PROC   NEAR
OOOF 8A 1E OOOA R       MOV    BL,CUSCODE    /Obtener código de des
0013 32 FF              XOR    BH, BH        ;Limpiar parte alta de
0015 DI E3              SHL    BX, 01        ,-Multiplicar por dos e
0017 FF A7 0000 R       JMP    [CUSTTBL+BX]  ;A rutina de tabla


001B            DOONODSC:                    ;Rutina código 0

001B EB OD 90           JMP    D90RET
001E            D10DSCT:                     ;Rutina código 1

001E EB OA 90           JMP    D90RET
0021            D20DSCT:                     ,Rutina código 2

0021 EB 07 90           JMP    D90RET
0024            D30DSCT:                     /Rutina código 3

0024 EB 04 90           JMP    D90RET
0027            D40DSCT:                     /Rutina código 4

0027 EB 01 90           JMP    D90RET
002A C3         D90RET: RET
002B            D10JUMP ENDP
                        END    BEGIN
```

Figura 8-5 Tabla de bifurcaciones

que serán movidos. Después planee la estrategia para las instrucciones: rutinas de inicialización, para uso de salto condicional y para uso de LOOP. Lo siguiente muestra la lógica principal: es seudocódigo que muchos programadores utilizan para planear un programa:

- Inicializar los registros de segmento
- Llamar a la rutina de bifurcación
- Llamar a la rutina del ciclo
- Regresar al DOS

La rutina de bifurcación podría ser planeada como:

- Inicializar los registros del conteo, para direcciones de nombres
- Salto 1:

- Mover un carácter del nombre
- Incrementar para pasar al siguiente carácter de nombre
- Decrementar el contador: si no es cero, Salto 1
- Si es cero, Regresar

La rutina del ciclo podría ser esbozada de una manera semejante.

3. Organice el programa en unidades lógicas tales que rutinas relacionadas se sigan una a otra. Procedimientos de alrededor de 25 líneas (el tamaño de la pantalla) son más fáciles de depurar que procedimientos más largos.
4. Utilice como guías otros programas. Intentos de memorizar todo el material técnico y codificar "sin pensarlo bien" con frecuencia tienen como resultado más errores en el programa.
5. Utilice comentarios para clarificar lo que se supone hace un procedimiento, qué operaciones aritméticas y de comparación son realizadas y lo que está haciendo una instrucción rara vez usada. (Un ejemplo de lo anterior es LOOPNE: ¿el ciclo se efectúa mientras no sea igual o hasta que no sea igual?)
6. Para teclear el programa, utilice una estructura de programa que pueda copiar en un archivo con>un nuevo nombre.

El resto de los programas en este texto hacen uso considerable de JMP, LOOP, saltos condicionales, CALL y llamadas a procedimientos. Ya cubierto lo básico de lenguaje ensamblador, ahora está en posición para programación más avanzada y realista.

## PUNTOS CLAVE

- Una dirección corta es alcanzada por medio de un desplazamiento y está limitada a una distancia de -128 a 127 bytes. Una dirección cercana es alcanzada por medio de un desplazamiento y está limitada a una distancia de -32,768 a 32,767 bytes dentro del mismo segmento. Una dirección lejana está en otro segmento y es alcanzada por medio de una dirección de segmento y un desplazamiento.
- Una etiqueta como "B20:" dentro de un procedimiento necesita dos puntos (:) para indicar que es una etiqueta cercana.
- Las etiquetas para instrucciones de salto condicional y LOOP deben ser cortas. El operando genera un byte de código objeto: 01H a 7FH que cubre el rango desde el +1 hasta el +127 decimales, y FFH a 80H cubre el rango desde -1 hasta -128. Ya que las instrucciones de máquina varían en longitud desde uno hasta cuatro bytes, el rango no es obvio, pero una guía práctica es alrededor de dos pantallas completas de código.
- Inicialice CX con un valor positivo cuando utilice LOOP, ya que LOOP disminuye el CX y verifica por un valor cero.
- Cuando una instrucción establece una bandera en 1, ésta permanece en 1 hasta que otra instrucción la cambia.
- Seleccione la instrucción apropiada de salto condicional, dependiendo de si la operación procesa datos con signo o sin signo.
- Utilice CALL para accesar un procedimiento e incluya RET al final del procedimiento para el regreso. Un procedimiento llamado puede llamar a otros procedimientos, y si usted sigue

las convenciones, RET hace que salga la dirección correcta de la pila. Los únicos ejemplos de este libro que saltan a un procedimiento están al inicio de los programas .COM.

- Utilice corrimiento (desplazamiento) a la izquierda para duplicar un valor y corrimiento a la derecha para dividirlo entre dos. Asegúrese de seleccionar la instrucción correcta de corrimiento para datos sin signo y para datos con signo.

## PREGUNTAS

**8-1.** Explique estos términos: (a) dirección corta; (b) dirección cercana; (c) dirección lejana.

**8-2.** (a) ¿Cuál es el número máximo de bytes que una instrucción JMP cercana, un LOOP y un salto condicional pueden saltar? (b) ¿Qué características del operando de código de máquina provocan este límite?

**8-3.** Una instrucción JMP empieza en la localidad con desplazamiento 0624H. Determine la dirección de transferencia con base en el siguiente código objeto para el operando de JMP: (a) 27H; (b) 6BH; (c) C6H.

**8-4.** Codifique una rutina usando LOOP que calcule la sucesión de Fibonacci: 1, 1, 2, 3, 5, 8, 13, . . . (Salvo por los dos primeros números en la sucesión, cada número es la suma de los dos números que le preceden.) Establezca el límite de 12 vueltas. Ensámblela, enlácela y utilice DEBUG para rastrear la rutina.

**8-5.** Suponga que AX y BX contienen datos con signo y que CX y DX contienen datos sin signo. Determine las instrucciones CMP (en donde sea necesaria) y de salto condicional para lo siguiente:
(a) ¿El valor de DX excede la de CX? (b) ¿El valor de BX excede al de AX? (c) El CX contiene cero? (d) ¿Existe un desbordamiento? (e) ¿El BX es igual o menor que el AX? (f) ¿El DX es igual o menor que el CX?

**8-6.** ¿Qué banderas son afectadas y qué contendrían en los siguientes sucesos?: (a) ocurrió un desbordamiento ; (b) un resultado es negativo; (c) un resultado es cero; (d) el procesamiento está en modo de avance paso por paso; (e) una transferencia de cadena se hace de derecha a izquierda.

**8-7.** Refiérase a la figura 8-3. Si el procedimiento B10 no contiene un RET, ¿cuál sería el efecto sobre la ejecución del programa?

**8-8.** ¿Cuál es la diferencia entre la codificación de un operando PROC con FAR y con NEAR?

**8-9.** ¿Cuáles son las formas en que un programa puede iniciar la ejecución de un procedimiento?

**8-10.** En un programa .EXE, A10 llama a B10, B10 llama a CÍO y CÍO llama a DIO. Como resultado de estas llamadas, ¿cuántas direcciones contiene la pila?

**8-11.** Suponga que el BL contiene 1110 0011 y que la localidad llamada BOONO contiene 0111 1001. Determine el efecto sobre el BL para lo siguiente: (a) XOR BL,BOONO; (b) AND BL.BOONO; (c) OR BL,BOONO; (d) XOR BL.llllllllB; (e) AND BL,0O0OOOO0B.

**8-12.** Corrija el programa de la figura 8-4 como sigue: Defina el contenido de TITLEX como letras mayúsculas y codifique las instrucciones que conviertan mayúsculas a minúsculas.

**8-13.** Suponga que el DX contiene 10111001 10111001 binario y que el CL contiene 03. Determine el contenido hexadecimal de DX después de la ejecución de las siguientes instrucciones no relacionadas (independientes): (a) SHR DX, 1; (b) SHR DX.CL; (c) SHL DX.CL; (d) SHL DL, 1; (e) ROR DX.CL; (0 ROR DL,CL; (g) SAL DH,1.

**8-14.** Utilice instrucciones para recorrer, mover y sumar para multiplicar el contenido de AX por 10.

**8-15.** Una rutina al final de la sección titulada "Rotación de bits" multiplica el DX:AX por 2. Corrija la rutina para (a) multiplicar por 4; (b) dividir entre 4; (c) multiplicar los 48 bits en el DX:AX:BX por dos.

## PARTE C — Operaciones para la pantalla y el teclado

## CAPÍTULO 9

# Introducción al procesamiento en pantalla y del teclado

OBJETIVO

**Introducir los requisitos para desplegar información en la pantalla y recibir información desde el teclado.**

### INTRODUCCIÓN

Hasta este punto, nuestros programas han definido datos ya sea en el área de datos o como datos inmediatos en un operando de instrucción. Sin embargo, la mayoría de los programas necesitan entradas desde un teclado, disco, ratón o módem y proporcionan salidas en un formato útil en la pantalla, impresora o disco. Este capítulo cubre los requisitos básicos para mostrar información en la pantalla y aceptar entradas desde el teclado.

Existen varios requisitos para especificar un dispositivo al sistema y solicitar una operación de entrada o salida. La instrucción INT (interrupción), para la mayoría de los propósitos, maneja entrada y salida. Los dos tipos de interrupciones tratados en este capítulo son las funciones de INT 10H del BIOS para manejar la pantalla y las funciones de INT 21H del DOS para mostrar salidas en pantalla y aceptar entrada desde el teclado. Estas *funciones* (o servicios) solicitan una acción; para identificar el tipo de operación que la interrupción va a realizar, inserte un número de función en el registro AH.

Las operaciones de bajo nivel del BIOS, como INT 10H transfieren el control de manera directa al BIOS. Sin embargo, para facilitar algunas de las operaciones más complejas, la INT 21H del DOS proporciona un servicio de interrupción que transfiere primero el control al DOS. Por ejemplo, la entrada desde un teclado puede consistir en un conteo de caracteres que se ingre-

san y verifican contra un número máximo. La operación INT 21H del DOS maneja gran parte de este procesamiento adicional de alto nivel y después transfiere el control de manera automática al BIOS, que maneja la parte de bajo nivel de la operación.

Como convención, este libro se refiere al número ODH como el carácter Enter para el teclado y como retorno de carro para la pantalla y la impresora.

Las operaciones introducidas en este capítulo son:

| FUNCIONES DE LA INT 10H DEL BIOS | | FUNCIONES DE LA INT 21H DEL DOS | |
|---|---|---|---|
| 02H | Fija el cursor | 02H | Despliega en pantalla |
| 06H | Recorre la pantalla | 09H | Despliega en pantalla |
| | | OAH | Entrada desde el teclado |
| | | 3FH | Entrada desde el teclado |
| | | 40H | Despliega en pantalla |

Los capítulos 10 y 11 cubren las características avanzadas para manejo de la pantalla y el teclado.

## LA PANTALLA

La pantalla es una malla de posiciones direccionables, en cualquiera de las cuales se puede colocar el cursor. Por ejemplo, un monitor común de video tiene 25 renglones (numerados del 0 hasta el 24) y 80 columnas (numeradas desde 0 hasta 79). A continuación se muestran varios ejemplos de ubicaciones del cursor:

| | Formato decimal | | Formato hexadecimal | |
|---|---|---|---|---|
| Ubicación en pantalla | Renglón | Columna | Renglón | Columna |
| Esquina superior izquierda | 00 | 00 | 00H | 00H |
| Esquina superior derecha | 00 | 79 | 00H | 4FH |
| Centro de la pantalla | 12 | 39/40 | 0CH | 27H/28H |
| Esquina inferior izquierda | 24 | 00 | 18H | 00H |
| Esquina inferior derecha | 24 | 79 | 18H | 4FH |

El sistema proporciona espacio en la memoria para un *área de despliegue de video,* o búfer. El área de despliegue monocromático inicia en la localidad de BIOS B000[0]H y permite utilizar 4K bytes de memoria: 2K disponibles para caracteres y 2K para atributos para cada carácter, como video inverso, intermitencia, intensidad y subrayado. El despliegue básico de video gráfico en color permite utilizar 16K bytes iniciando en la localidad de BIOS B800[0]H. Se puede procesar ya sea en modo de texto para carácter normal o en modo gráfico. Para modo de texto, el área de despliegue ofrece para la pantalla "páginas" numeradas desde la cero hasta la tres para una pantalla de 80 columnas, con bytes para cada carácter y su atributo.

Las interrupciones que manejan los despliegues en pantalla transfieren sus datos de forma directa al área de despliegue de video, dependiendo del tipo de adaptador de video instalado, como EGA o VGA. Aunque técnicamente sus programas pueden transferir datos en forma directa al área de despliegue de video, no existe seguridad de que las direcciones de memoria serán las mismas en todos los modelos, de modo que la escritura directa de datos en el área de despliegue, si bien rápida, puede ser riesgosa. La práctica recomendada es utilizar las instrucciones de interrupción adecuadas: las funciones de la INT 10H para despliegue, ubicar el cursor en cualquier posición y limpiar la pantalla, y las funciones de INT 21H para diferentes tipos de despliegue.

## COLOCACIÓN DEL CURSOR

La colocación del cursor es un requisito común en modo de texto, ya que su posición determina en dónde será desplegado el siguiente carácter. (El modo gráfico no permite el uso del cursor.) La INT 10H es la operación del BIOS para manejo de la pantalla, y la función 02H en el AH indica la operación que coloca al cursor. Se carga el número de página (o pantalla), por lo común 0, en el registro BH y en el DX el renglón y columna requeridos. Los contenidos de los otros registros no son importantes.

Las instrucciones siguientes colocan el cursor en el renglón 05, columna 12:

```
MOV    AH,02H      ,•Petición para colocar el cursor
MOV    BH, 00      ;Número de página 0
MOV    DH, 05      ;Renglón 05
MOV    DL, 12      ,•Columna 12
INT    10H         /Interrupción que llama al BIOS
```

Para establecer el renglón y columna en el DX también puede utilizar una instrucción MOV con un valor hexadecimal inmediato, como

```
MOV    DX,050CH    /Renglón 05, columna 12
```

## LIMPIAR LA PANTALLA

La función 06H de la INT 10H del BIOS maneja el borrado o recorrido de la pantalla. Puede limpiar todo o parte de un despliegue iniciando en cualquier localidad de la pantalla y terminando en cualquier localidad con número mayor. Por ejemplo, para limpiar toda la pantalla especifique el renglón:columna iniciales como 00:00H y el renglón:columna finales como 18:4FH. Cargue estos registros:

- AH = función 06H
- AL = OOH para la pantalla completa
- BH = número del atributo
- CX = renglón:columna iniciales
- DX = renglónxolumna finales

En el ejemplo siguiente el atributo 71H establece toda la pantalla con fondo blanco (7) con primer plano azul (1):

```
MOV    AX,0600H    ,-AH 06 (recorrido), AL 00 (pantalla completa)
MOV    BH,71H      ;Atributo; blanco (7) sobre azul (1)
MOV    CX,0000H    /Esquina superior izquierda renglón: columna
MOV    DX,184FH    ;Esquina inferior derecha renglón: columna
INT    10H         /Interrupción que llama al BIOS
```

Si de modo equivocado establece usted la ubicación de la esquina inferior derecha de la pantalla en algo mayor que 184FH, la operación da vuelta a la pantalla y limpia dos veces algunas

localidades. Esto puede causar un error en algunos sistemas. El capítulo siguiente describe el recorrido con mayor detalle.

Con frecuencia un programa tiene que desplegar mensajes al usuario que solicita datos o le indica que ejecute una acción. Primero examinaremos los métodos de las versiones originales del DOS, que son útiles para ejercicios y programas pequeños, y más adelante examinaremos los métodos con manejadores de archivo. Las operaciones del DOS original trabajan con todas las versiones y en algunos aspectos son más sencillas y más fáciles de usar, aunque se recomienda utilizar operaciones más recientes para el desarrollo de software.

## FUNCIÓN 09H DEL DOS PARA DESPLIEGUE EN PANTALLA

La simplicidad de la función 09H del DOS original para el despliegue es lo que la mantiene en uso común. Requiere la definición de una cadena de despliegue en el área de datos. La cadena es seguida inmediatamente por un delimitador de signo de pesos ($, o 24H), el cual utiliza la operación para finalizar el despliegue. El ejemplo siguiente lo ilustra:

```
NAMPRMP DB 'Customer name?','$'  ,-Cadena de despliegue
```

Puede codificar el signo de pesos inmediatamente después de la cadena de despliegue como se mostró, como parte de la cadena como en '¿Nombre del cliente?$', o en la línea siguiente como en DB '$'. Sin embargo, el resultado es que no puede utilizar esta función para desplegar en la pantalla un carácter $.

Coloque la función 09H en el registro AH, utilice LEA para cargar la dirección de la cadena de despliegue en el DX, y emita una instrucción INT 21H. La operación despliega los caracteres de izquierda a derecha y reconoce el final de los datos al encontrar el delimitador de signo de pesos ($). El código en lenguaje ensamblador es:

```
MOV   AH, 09H          ,-Petición para desplegar
LEA   DX,NAMPRMP       ;Carga la dirección de la indicación
INT   21H              ;Llama al DOS
```

La operación INT no cambia el contenido de los registros. Una cadena desplegada que excede la columna de la extrema derecha de la pantalla continúa de forma automática en el siguiente renglón, recorriendo la pantalla tanto como sea necesario. Si al final de la cadena se omite el signo de pesos, la operación despliega caracteres de la memoria hasta que encuentre un signo así, si existe alguno.

### Uso de la función 09H de la INT 21H para desplegar caracteres ASCII

La mayor parte de los 256 caracteres ASCII están representados por símbolos que pueden ser desplegados en una pantalla de video. Algunos valores, como 00H y FFH, pueden no tener un símbolo desplegable y aparecen como un espacio en blanco, aunque el verdadero carácter ASCII de espacio en blanco es 20H.

El programa .COM de la figura 9-1 despliega grupo completo de caracteres ASCII. El programa llama a tres procedimientos:

- B10CLR utiliza la función 06H de la INT 10H para limpiar la pantalla.
- C10SET utiliza la función 02H de la INT 10H para inicializar el cursor en 00,00H.

```
        page  60,132
TITLE   P09DOSAS (COM)   Exhibe los caracteres ASCII 00H-FFH
        .MODEL SMALL
        .CODE
        ORG     100H
BEGIN:  JMP     SHORT MAIN
CHAR    DB      00,'$'

;               Procedimiento principal:
;-----------------------------------------------------------------
MAIN    PROC    NEAR
        CALL    B10CLR          ;Limpiar pantalla
        CALL    C10SET          ;Fijar cursor
        CALL    D10DISP         ;Exhibir caracteres
        MOV     AX.4C00H        ;Salir a DOS
        INT     21H
MAIN    ENDP
;
;               Despejar pantalla:
;-----------------------------------------------------------------
B10CLR  PROC    NEAR
        MOV     AX,0600H        Recorrer toda la pantalla
        MOV     BH, 07          Atributo: blanco sobre negro
        MOV     CX,0000         Posición izquierda superior
        MOV     DX,184FH        Posición derecha interior
        INT     10H
        RET
B10CLR  ENDP
;               Fijar cursor en 00,00:
;-----------------------------------------------------------------
C10SET  PROC    NEAR
        MOV     AH,02H          ,-Petición de fijar cursor
        MOV     BH,00           ;Página No. 0
        MOV     DX,0000         /Hilera 0, columna 0
        INT     10H
        RET
C10SET  ENDP
;               Exhibir caracteres ASCII
;-----------------------------------------------------------------
D10DISP PROC
        MOV     CX,256          Iniciar 256 iteraciones
        LEA     DX, CHAR        Iniciar dirección de carácter
D20:
        MOV     AH,09H          ,-Exhibir carácter ASCII
        INT     21H
        INC     CHAR            ;Incrementar para el siguiente carácter
        LOOP    D20             ;Decrementar CX, ciclo diferente de cero
        RET                     ;Regresar
D10DISP ENDP
        END     BEGIN
```

Figura 9-1 Función del DOS para mostrar el conjunto de caracteres ASCII.

- D10DISP utiliza la función 09H de la INT 21H para desplegar el contenido de CHAR que es inicializado en OOH y de manera sucesiva es incrementado en uno para desplegar cada carácter hasta alcanzar FFH.

La primera línea desplegada inicia con un blanco (OOH), dos "caritas felices" (01H y 02H) y después un corazón (03H), un diamante (04H), un trébol (05H). El carácter 06H tendría que mostrar una pica, pero es borrada por caracteres de control posteriores. El carácter 07H hace que suene la bocina, 08H provoca un carácter de retroceso, 09H ocasiona un tabulador, OAH provoca un

avance de línea y ODH (Enter) causa un "retorno de carro" para el inicio de la línea siguiente. Y, por supuesto, con esta operación, el símbolo de pesos, 24H, no se despliega. (Como verá en el capítulo 10, los servicios del BIOS pueden desplegar símbolos apropiados para estos caracteres especiales.) El símbolo de la nota musical es OEH, y 7FH hasta FFH son caracteres ASCII extendidos.

Puede corregir el programa para librar el intento de desplegar los caracteres de control. Las instrucciones siguientes evitan todos los caracteres entre 08H y ODH; puede querer experimentar con esta desviación, digamos, sólo para 08H (Retroceso) y ODH (Retorno de carro).

```
        CMP   CHAR,08H        ;¿Menor a 08H?
        JB    D30             ,-Sí, entonces aceptar
        CMP   CHAR,ODH        ;¿Menor o igual a ODH?
        JBE   D40             ;Sí, entonces evitarlo
D30:
        MOV AH.09H            Desplegar los menores que 08H
                              y los mayores que ODH
        INT   21H             Llama al DOS
D40:
        INC   CHAR
```

Aunque este ejercicio los evita, el despliegue de los caracteres de retroceso, tabulador, avance de línea y retorno de carro es la forma normal de realizar estas operaciones.

Sugerencia: Reproduzca el programa anterior, ensámblelo, enlácelo y conviértalo en un archivo .COM.

## FUNCIÓN OAH DEL DOS PARA ENTRADA DEL TECLADO

En particular, la función OAH de la INT 21H para aceptar datos desde el teclado es poderosa. El área de entrada para los caracteres tecleados requiere de una *lista de parámetros* que contenga los campos especificados que la operación INT va a procesar. Primero, la interrupción necesita conocer la longitud máxima de los datos de entrada. El propósito es advertir a los usuarios que tecleen caracteres en demasía; la operación envía sonidos por la bocina y no acepta caracteres adicionales. Segundo, la operación envía a la lista de parámetros el número de bytes que realmente se introdujeron.

El código que sigue define una lista de parámetros para un área de entrada. (Si ha trabajado en un lenguaje de alto nivel, puede ser que haya utilizado el término *registro* o *estructura.)* LABEL es una directiva con el tipo de atributo de BYTE, que sólo provoca alineación en un límite (o frontera) de byte. El primer byte contiene su límite del número máximo de caracteres de entrada. El mínimo es cero y, ya que es un campo de un byte, el máximo es FFH, o 255. Usted decide sobre el máximo, con base en la clase de datos que espera que los usuarios introduzcan. El segundo byte es para la operación que almacena el número real de caracteres introducidos como un valor binario. El tercer byte inicia un campo que contiene los caracteres tecleados, de izquierda a derecha. El código en lenguaje ensamblador es:

```
NAMEPAR  LABEL  BYTE          Inicio de la lista de parámetros
MAXLEN   DB     20            Número máximo de caracteres de entrada
ACTLEN   DB     ?             Número real de caracteres de entrada
NAMEFLD  DB     20 DUP(")     Caracteres introducidos del teclado
```

En la lista de parámetros, la directiva LABEL indica al ensamblador que alinee en un límite de byte y dé a la localidad el nombre NAMEPAR. Puesto que LABEL no ocupa espacio, NAMEPAR y MAXLEN se refieren a la misma localidad de memoria.

Para solicitar una entrada, establezca la función OAH en el AH, cargue la dirección de la lista de parámetros (en el ejemplo NAMEPAR), en el DX, y emita INT 21H:

```
MOV  AH, OAH          Petición de la función de entrada
LEA  DX, NAMEPAR      Carga la dirección de la lista de parámetros
INT  21H              Llama al DOS
```

La operación INT espera que el usuario introduzca caracteres y verifica que no excedan el máximo (20 en MAXLEN en la lista de parámetros). La operación repite cada carácter en la pantalla y avanza el cursor. El usuario presiona la tecla Enter para señalar el final de la entrada. La operación también transfiere el carácter Enter (ODH) al campo de entrada (en el ejemplo, NAMEFLD) pero no lo cuenta en la longitud real. Si teclea un nombre como BROWN (Enter), la lista de parámetros es como lo siguiente:

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ASCII: | 20 | 5 | B | R | O | W | N | # | | | | |
| HWX: | 14 | 05 | 42 | 52 | 4F | 57 | 4E | OD | 20 | 20 | 20 | 20 |

La operación envía la longitud del nombre de entrada, 05H, al segundo byte de la lista de parámetros, llamado en el ejemplo ACTLEN. El carácter Enter (ODH) está en NAMEFLD+5. (Aquí el símbolo # indica este carácter, ya que ODH no es un símbolo imprimible.) Puesto que la longitud máxima es de 20, incluyendo el ODH, el nombre introducido sólo puede ser de hasta 19 caracteres.

La operación acepta y actúa sobre el carácter de retroceso, pero no lo agrega a la cuenta. La operación no acepta más que el número máximo de caracteres. Si en el ejemplo anterior un usuario teclea 20 caracteres sin presionar Enter, la operación provoca que suene la bocina; en este punto, sólo acepta el carácter Enter.

La operación pasa por alto las teclas de función ampliada, como F1, Inicio, RePág y las teclas de dirección del cursor (flechas). Si usted espera que el usuario introduzca alguna de ellas, utilice la INT 16H del BIOS o función 01H de la INT 21H del DOS, ambas estudiadas en el capítulo 11.

## CÓMO ACEPTAR Y DESPLEGAR NOMBRES

El programa de la figura 9-2 pide al usuario que introduzca un nombre y después lo despliega er el centro de la pantalla y emite un sonido la bocina. Por ejemplo, si el usuario introduce el nombre Pat Brown, el programa realiza lo siguiente:

1. Divide la longitud 09 entre dos: 9/2 = 4, ignorando la fracción.
2. Resta este resultado de 40: 40 -4 = 36.

En F10CENT, la instrucción SHR corre la longitud 09 un bit a la derecha dividiendo de hecho la longitud entre 2. Los bits 00001001 se convierten en 00000100, o 4. La instrucción NEG invierte el signo, cambiando +4 a -4. ADD suma el valor 40, dando en el registro DL la posición inicial de la columna, 36. Con el cursor colocado en el renglón 12, columna 36, el nombre aparece en la pantalla como sigue:

```
          page   60,132
TITLE     P09CTRNM (EXE)  Acepta nombres y los centra en la pantalla

          .MODEL SMALL
          .STACK 64

          .DATA
ÑAME PAR  LABEL  BYTE              Lista de parámetros nombre:
MAXNLEN   DB     20                 longitud máxima de nombre
NAMELEN   DB                        no. de caracteres introducidos
NAMEFLD   DB     21 DUP(' '         nombre introducido
PROMPT    DB     'Ñame? '

          .CODE
BEGIN     PROC   FAR
          MOV    AX,@data          ;Iniciar registros
          MOV    DS,AX             ; de segmento
          MOV    ES,AX
          CALL   Q10CLR            ,-Despejar pantalla
A20LOOP:
          MOV    DX,0000           ;Fijar cursor en 00,00
          CALL   Q20CURS
          CALL   B10PRMP           /Exhibir indicación
          CALL   D10INPT           ;Proporciona entradas del nombre
          CALL   Q10CLR            ,-Despejar pantalla
          CMP    NAMELEN,00        ;¿Se ingresó el nombre?
          JE     A30               ; no, salida
          CALL   El0CODE           ,-Fijar campana y '$'
          CALL   FIOCENT           ,-Centra y exhibe el nombre
          JMP    A20LOOP
A30:
          MOV    AX,4C00H          ;Salir a DOS
          INT    21H
BEGIN     ENDP
                 Exhibe indicador:

B10PRMP   PROC   NEAR
          MOV    AH,09H            ,-Petición de exhibición
          LEA    DX,PROMPT
          INT    21H
          RET
B10PRMP   ENDP
                 Acepta entrada de nombre:

D10INPT   PROC   NEAR
          MOV    AH, OAH           ;Petición de teclado
          LEA    DX, ÑAME PAR      ,- entrada
          INT    21H
          RET
          ENDP
```

Figura 9-2 Cómo aceptar y mostrar nombres

```
                    Fijar campana y delimitador '$'

El 0CODE    PROC    NEAR
            MOV     BH, 00              .•Reemplaza carácter Enter  (OD)
            MOV     BL,NAMELEN          ; con el de la campana  (07)
            MOV     NAMEFLD[BX],07
            MOV     NAMEFLD[BX+1],'$'   ;Pone el delimitador de exhibición
            RET
El0CODE     ENDP
                    Centrar y exhibir nombre

FIO-CENT    PROC    NEAR
            MOV     DL,NAMELEN          Localiza columna central:
            SHR     DL,1                 divide longitud en 2,
            NEG     DL                   invierte el seguro
            ADD     DL,40                suma 40
            MOV     DH,12               Centra hilera
            CALL    Q2 0CURS            Fija cursor
            MOV     AH,09H
            LEA     DX, NAMEFLD         ;Exhibe nombre
            INT     21H
            RET
FIOCENT     ENDP
                    Despejar pantalla

Q10CLR      PROC    NEAR
            MOV     AX,0600H            Petición de recorrido de pantalla
            MOV     BH, 30              Color  (07 para Blanco y Negro)
            MOV     CX,0000             De  00,00
            MOV     DX,184FH            A  24,79
            INT     10H
            RET
Q10CLR      ENDP
                    Fijar hilera/columna de cursor

Q2 0CURS    PROC    NEAR                DX fija en entrada
            MOV     AH,02H              Petición de ubicar cursor
            MOV     BH, 00              Página 0
            INT     10H
            RET
Q2 0CÚRS    ENDP

            END
```

Figura 9-2 (continuación)

```
Renglón 12:  Pat   Brown

                !      I

Columna:      36     40
```

Observe que la instrucción en E10CODE que inserta el carácter campana (07H) en el área de entrada sigue de manera inmediata al nombre:

```
MOV   BH,00           ,-Reemplaza el carácter Enter  (ODH)

MOV   BL,NAMELEN      / con el carácter campana  (07H)

MOV    NAMEFLD  [BX],07H
```

Los dos primeros MOV establecen el BX con la longitud. El tercer MOV hace referencia a un especificador de índice en corchetes, que significa que el BX actúa como un registro especial de índice para facilitar el direccionamiento extendido. El MOV combina la longitud en el BX con la

dirección de NAMEFLD y mueve el 07H a la dirección calculada. Así, para una longitud de 05 la instrucción inserta 07H en NAMEFLD+5 (reemplazando el carácter Enter) a continuación del nombre. La última instrucción en E10CODE inserta un delimitador '$' después del 07H, de manera que la función 09H del DOS pueda desplegar el nombre y sonar la bocina.

### Respuesta con sólo la tecla Enter

El programa continúa aceptando y desplegando nombres hasta que el usuario presione sólo la tecla Enter como respuesta a una petición. La función 09H del DOS la acepta e inserta una longitud de 00H en la lista de parámetros, como:

```
Lista de parámetros (hexadecimal):  |14j00|OD|  ...
```

Si la longitud es cero, el programa determina que la entrada ha finalizado, como lo muestra por la instrucción CMP NAMELEN,00 en A20LOOP.

### Cómo borrar el carácter Enter

Usted puede utilizar caracteres de entrada para diferentes propósitos, como imprimir un reporte, almacenar en una tabla o escribir en un disco. Para ello, tiene que haber reemplazado el carácter Enter (ODH) con un espacio en blanco (20H) siempre que éste aparezca en NAMEFLD. El campo que contiene la longitud real de los datos de entrada, NAMELEN, proporciona la posición relativa del carácter Enter. Por ejemplo, si NAMELEN contiene 05, entonces el carácter Enter está en NAMEFLD+5. Puede mover esta longitud al registro BX para indexar la dirección de NAMEFLD como sigue: <

```
MOV  BH,00              ,-Establece el BX
MOV  BL,NAMELEN         ; a 00 05
MOV  NAMEFLD [BX]  , 20H ;Borra el carácter Enter
```

Las dos primeras instrucciones MOV establecen el BX con la longitud 05. El tercer MOV mueve un espacio en blanco (20H) a la dirección especificada en el primer operando: la dirección de NAMEFLD más el contenido de BX —en realidad, NAMEFLD+5.

### Cómo limpiar el área de entrada

Los caracteres introducidos reemplazan a los anteriores que están en un área de entrada y permanecen hasta que otros caracteres los reemplazan. Considere las siguientes entradas sucesivas:

```
  ENTRADA              ÑAMEPAR  (HEX)

1. PAINE       |14|05|50|41|49|4E|45|0D|20|20|20|...|20|
2. HAMILTON    |14|08|48|41|4D|49|4C|54|4F|4E|0D|...|20|
3. ADAMS       |14|05|41|44|41¡4D|53|0D|45|5A|0D|...|20|
```

El nombre HAMILTON reemplaza al nombre más corto PAINE. Pero ya que el nombre ADAMS es más corto que HAMILTON, reemplaza HAMIL y el carácter Enter reemplaza a la T. Las letras restantes, ON, aún siguen a ADAMS. Puede querer borrar NAMEFLD antes de solicitar un nombre, como sigue:

```
        MOV    CX,20               ;Inicializa para realizar 20 ciclos

        MOV    SI,0000             ;Inicia la posición del nombre
B30:
        MOV    NAMEFLD [SI]  ,20H  Un espacio en blanco al nombre
        INC    SI                  Incrementa para el siguiente carácter
        LOOP   B30                 20 veces
```

En lugar del registro SI puede utilizar el DI o el BX. Un método más eficaz que mueve una palabra de dos espacios en blanco necesita 10 ciclos. Sin embargo, como NAMEFLD está definido como DB (byte), tendría que invalidar su longitud con un operando WORD y PTR (apuntador), como se indica a continuación:

```
        MOV    CX,10               ,-Inicializa para 10 ciclos

        LEA    SI,NAMEFLD          ;Inicializa el principio del nombre
B30:
        MOV    WORD PTR [SI]  ,2020H ;Dos espacios en blanco para el nombre
        INC    SI                  /Incrementa dos lugares
        INC    SI                 /  en el nombre
        LOOP   B30                 /Repite 10 veces
```

Interprete el MOV en B30 como "Mover una palabra en blanco a la localidad de memoria a donde apunta el registro SI". Este ejemplo utiliza LEA para inicializar el borrado de NAMEFLD y utiliza un método ligeramente diferente para el MOV en B30 porque ya no puede codificar una instrucción como

```
MOV WORD PTR[NAMEFLD],2020H  /No válido
```

El borrado del área de entrada resuelve el problema de nombres más cortos que siguen a datos anteriores. Una práctica más efectiva es borrar sólo las posiciones a la derecha del nombre que ha sido ingresado de manera más reciente.

## USO DE CARACTERES DE CONTROL PARA DESPLEGAR

Una manera de hacer más eficaz el uso de despliegues es utilizar los caracteres de control Retorno de carro, Avance de línea y el Tabulador. Puede codificarlos como valores ASCII o números hexadecimales, así:

| CARÁCTER DE CONTROL | ASCII | HEX | EFECTO EN EL CURSOR |
|---|---|---|---|
| Retorno de carro | 13 | 0DH | Restablece a la posición de la extrema izquierda |
| Avance de línea | 10 | 0AH | Avanza a la línea siguiente |
| Tabulador | 09 | 09H | Avanza a la siguiente marca de tabulador |

Siempre que despliegue salidas o acepte entradas, utilice estos caracteres de control para el manejo del cursor. Aquí está un ejemplo que despliega el contenido de una cadena de caracteres llamada MESSAGE, seguida por un retorno de carro y un avance de línea para colocar el cursor en la línea siguiente:

```
MESSAGE  DB   09, 'PC Users Group Annual Report', 13, 10, '$'


         MOV  AH,09H          Petición de despliegue
         LEA  DX,MESSAGE      Carga la dirección del título
         INT  21H             Llama al DOS
```

El uso de EQU para redefinir los caracteres de control puede hacer que un programa sea más legible:

```
CR       EQU  13     ; (o EQU ODH)
LF       EQU  10     ;(o EQU OAH)
TAB      EQU  09     ;(o EQU 09H)
MESSAGE  DB   TAB, "PC Users Group Annual Report', CR, LF, '$'
```

## FUNCIÓN 02H DEL DOS PARA DESPLIEGUE EN PANTALLA

Puede encontrar que la función 02H de la INT 21H, sea útil para despliegue de un solo carácter. Cargue en el DL el carácter que será desplegado en la posición actual del cursor, y solicite la INT 21H. Los caracteres de Tabulador, Retorno de carro y Avance de línea actúan normalmente, y la operación avanza de manera automática el cursor. El código en lenguaje ensamblador es:

```
MOV     AH,02H     ;Petición de desplegar un carácter
MOV     DL.char    /Carácter desplegado
INT     21H        /Llama al DOS
```

El ejemplo siguiente muestra cómo utilizar este servicio para desplegar una cadena de caracteres. La cadena para desplegar está definida en CONAME. El programa carga la dirección de CONAME en el registro DI y su longitud en el CX. El ciclo implica el incremento de DI (en INC) para cada carácter sucesivo y la disminución del CX (en LOOP) para el número de caracteres desplegados. El código es como sigue:

```
CONAME   DB     "Software Services', 13, 10


         MOV    AH,02H        /Petición para desplegar un carácter
         MOV    CX, 19        /Longitud de la cadena de caracteres
         LEA    DI,CONAME     /Dirección de la cadena de caracteres
         MOV    DL, [DI]      /Carácter que se despliega
         INT    21H           /Llama al DOS
```

```
INC   DI          ;Incrementa para el siguiente carácter
LOOP  A30         ,-Si aún no termina repite el ciclo
...               /Terminación
```

## MANEJADORES DE ARCHIVOS

Ahora examinaremos el uso de los *manejadores de archivos* para operaciones con la pantalla y el teclado, que está más en el estilo de UNIX o del OS/2. Un manejador de archivo sólo es un número que hace referencia a un dispositivo específico. Ya que los manejadores de archivo siguiente están preestablecidos, no tiene que definirlos:

| MANEJADOR | DISPOSITIVO |
|---|---|
| 00 | Entrada, por lo regular el teclado (CON), pero puede ser redireccionado |
| 01 | Salida, por lo regular la pantalla (CON), pero puede ser redireccionado |
| 02 | Error en la entrada, pantalla (CON), no puede ser redireccionado |
| 03 | Dispositivo auxiliar (AUX) |
| 04 | Impresora (LPT1 o PRN) |

Como puede verse, los manejadores de archivo normales son 00 para entrada del teclado ) 01 para despliegue en pantalla. Otros manejadores de archivo, como aquellos para dispositivos dt disco, tienen que ser establecidos por su programa. También puede utilizar estos servicios par; redireccionar la entrada y la salida a otros dispositivos, aunque esta característica por el momentc no nos interesa.

## MANEJADORES DE ARCHIVO PARA DESPLIEGUE EN PANTALLA

La función 40H de la INT 21H del DOS utiliza los manejadores de archivo para solicitar la: operaciones de despliegue. Cargue los registros siguientes:

- AH = Función 40H
- BX = Manejador de archivo 01
- CX = Número de caracteres a desplegar
- DX = Dirección del área de despliegue

Una operación INT exitosa regresa al AX el número de bytes escritos y pone en cero la bandera *á* acarreo (la cual puede usted examinar).

Una operación INT fallida pone en uno la bandera de acarreo y regresa un código de erro en el AX: 05H = acceso denegado (para un dispositivo no válido o desconectado) o 06H = manejador no válido. Ya que el AX puede contener ya sea una longitud o un código de error, 1 única forma de determinar una condición de error es probar la bandera de acarreo, aunque lo errores en el despliegue son raros:

```
JC rutina-de-error ;Prueba por si existe error en el despliegue
```

La operación responde igual que la función 09H del DOS a los caracteres de control 071 (Campana), 08H (Retroceso), OAH (Avance de línea) y ODH (Retorno de carro). Las instruccic nes siguientes ilustran esta operación:

```
DISAREA  DB     'PC Users Society' , ODH, OAH  ;Área de despliegue


         MOV    AH,4 0H                       ;Petición de despliegue
         MOV    BX, 01                        ,-Manejador de archivo de salida
         MOV    CX , 18                       ,-Despliega 18 caracteres
         LEA    DX,DISAREA                    ;Área de despliegue
         INT    21H                           ;Llama al DOS
```

**Ejercicio: Despliegue en la pantalla**

Usemos DEBUG para examinar los efectos internos de utilizar un manejador de archivo para desplegar su propio nombre. Cargue DEBUG, y cuando aparezca su indicación, teclee A 100 para empezar a introducir las instrucciones siguientes (pero no los números de la extrema izquierda) en el desplazamiento 100H (recuerde que DEBUG supone que los números ingresados están en formato hexadecimal):

```
10 0 MOV AH,4 0
102  MOV BX,01
105  MOV CX,xx  (Inserte  la  longitud de  su nombre)
108  MOV DX,10E
10B  INT 21
10D  NOP
10E  DB  'Your ñame'
```

El programa establece el AH para solicitar un despliegue y establece el desplazamiento 10EH en el DX —la localidad del DB que contiene su nombre.

Cuando haya tecleado las instrucciones, presione otra vez Enter. Para desensamblar el programa utilice el comando U (U 100,10D) y rastree la ejecución, presione R y después repetidos comandos T. Al llegar a la instrucción INT, utilice el comando P (Proceder) para ejecutar toda la interrupción hasta la instrucción NOP. Su nombre debe ser mostrado en la pantalla. Utilice el comando Q para salir del DEBUG.

## MANEJADORES DE ARCHIVO PARA ENTRADA DESDE EL TECLADO

La función 3FH de la INT 21H del DOS, utiliza manejadores de archivo para solicitar entrada del teclado, aunque es una operación un poco ineficaz. Cargue los registros siguientes:

- AH = Función 3FH
- BX = Manejador de archivo 00
- CX = Número máximo de caracteres que se aceptan
- DX = Dirección del área de datos para introducir los caracteres

Una operación exitosa INT pone en cero la bandera de acarreo (que puede probar) y establece el AX con el número de caracteres introducidos.

Una operación INT fallida podría deberse a un manejador no válido; la operación pone en uno la bandera de acarreo e inserta un código de error en el AX: 05H = acceso denegado (para un dispositivo no válido o uno desconectado) o 06H = manejador no válido. Ya que el AX podría contener ya sea la longitud o un código de error, la única forma de determinar una condición de error es examinar la bandera de acarreo, aunque los errores de teclado presumiblemente son raros.

Igual que la función OAH del DOS, la función 3FH también actúa sobre el carácter de retroceso, pero ignora teclas de función extendidas tal como F1, Inicio y AvPág.

Las instrucciones siguientes ilustran el uso de la función 3FH del DOS:

```
INAREA   DB    20 DUP(' ')  ;Area de entrada


         MOV   AH,3FH       ,-Petición de entrada
         MOV   BX,00        ;Manejador de archivo para el teclado
         MOV   CX,20        ,•Máximo 20 caracteres
         LEA   DX, INAREA   ,-Área de entrada
         INT   21H          :Llama al DOS
```

La operación INT espera que usted introduzca caracteres, pero desafortunadamente no verifica si el número de éstos excede el máximo en el registro CX (20 en el ejemplo). La presión de la tecla Enter (ODH) señala la terminación de una entrada. Por ejemplo, el tecleo de los caracteres "PC Users Group" introduce lo siguiente en INAREA:

```
|PC Users Group|ODH|OAH|
```

Los caracteres tecleados son seguidos de manera inmediata por un Enter (ODH), que usted tecleó, y un avance de línea (OAH) que no tecleó. A causa de este hecho, el número máximo y la longitud del área de entrada deben dar espacio para dos caracteres adicionales. Si teclea menos caracteres del máximo, las localidades siguientes en memoria a los caracteres ingresados aún contienen los caracteres ingresados con anterioridad.

Una operación INT exitosa pone en cero la bandera de acarreo y establece el AX con el número de caracteres enviados. En el ejemplo anterior, este número es 14 más 2 por los caracteres Enter y avance de línea, es decir 16. De acuerdo con esto, un programa puede determinar el número real de caracteres introducidos. Aunque esta característica es trivial para respuesta SI y NO, es útil para respuestas con longitud variable, como nombres.

Si teclea un nombre que exceda el máximo en el registro CX, la operación en realidad acepta todos los caracteres. Considere una situación en la que el CX contiene 08 y un usuario introduce los caracteres "PC Exchange". La operación coloca los primeros ocho caracteres en el área de entrada "PC Excha" sin Enter ni Avance de línea siguiéndolos y establece el AX con una longitud de 08. Ahora, observe esto: la siguiente operación INT por ejecutar no acepta un nombre directamente del teclado, ya que el resto de la cadena anterior aún se encuentra en su *búfer.* Envía "nge" seguido por los caracteres Enter y Avance de línea al área de entrada y establece el AX en 05. Ambas operaciones son "normales" y ponen en cero la bandera de acarreo:

```
Primer INT:     PC Excha         AX = 08
Segundo INT:    nge, ODH, OAH    AX = 05
```

Un programa puede identificar si el usuario ha tecleado un número "válido" de caracteres si (a) el número que regresa el AX es menor que el que está en el CX o (b) el número regresado en el AX es igual al que está en el CX y los dos últimos caracteres en el área de entrada son ODH y OAH. Si ninguna de estas dos condiciones son verdaderas, tendrá que emitir INT adicionales para aceptar los caracteres restantes. Después de todo esto, ¡quizá se pregunte cuál es el sentido de especificar una longitud máxima en el CX!

Ejercicio: Ingreso de datos

A continuación haremos un ejercicio con DEBUG en el que puede ver el efecto de utilizar la función 3FH del DOS para ingresar datos. El programa permite que usted teclee hasta 12 caracteres, incluyendo un carácter para Enter y uno para el Avance de línea. Cargue DEBUG, y cuando aparezca la indicación, en la localidad 100H, introduzca las instrucciones siguientes (pero no los números):

```
100   MOV AH, 3F
102   MOV BX, 00
105   MOV CX, 0C
108   MOV DX,10F
10B   INT 21
10D   JMP 100
10F   DB 20 20 20 20 20 20 20 20 20 20 20 20
```

El programa establece el AH y el BX para solicitar una entrada desde el teclado e inserta la longitud máxima en el CX. También establece el desplazamiento 10FH en el DX —la localidad del DB, en donde los caracteres ingresados van a comenzar.

Cuando ha tecleado las instrucciones, otra vez presione Enter. Pruebe el comando U (U 100,10E) para desensamblar el programa. Utilice los comandos R y repetidos T para rastrear la ejecución de las cuatro instrucciones MOV. En la localidad 10BH, utilice P (Proceder) para ejecutar a través de la interrupción. La operación espera que usted teclee caracteres seguidos por un Enter. Verifique el contenido del registro AX y de la bandera de acarreo, y utilice D DS:10F para desplegar los caracteres ingresados en memoria. Puede continuar el ciclo de manera indefinida. Teclee Q para salir de DEBUG.

## PUNTOS CLAVE

- El despliegue monocromático permite utilizar 4K bytes de memoria: 2K están disponibles para caracteres y 2K para un atributo de cada carácter.
- El despliegue básico de color permite utilizar 16K bytes y puede operar en color o monocromo. Puede procesar ya sea en modo de texto, para despliegue normal de caracteres, o en modo gráfico.
- Sea consistente en el uso de la notación hexadecimal. Por ejemplo, INT 21 no es lo mismo que INT 21H.

- La instrucción INT 10H transfiere el control al BIOS para operaciones de despliegue. Dos operaciones comunes son la función 02H (ubicar el cursor) y 06H (recorrer la pantalla).
- DOS INT 21H provee funciones especiales para manejar algunos problemas input/output.
- La función 09H de la INT 21H del DOS para despliegue define un delimitador ($) inmediatamente después del área de despliegue. Un delimitador ausente puede provocar efectos espectaculares en la pantalla.
- La función OAH de la INT 21H para entrada del teclado espera que el primer byte contenga un número máximo e inserta de manera automática un valor real en el segundo byte.
- Un manejador de archivo es un número que se refiere a un dispositivo específico. Algunos números para los manejadores están preestablecidos, mientras que otros los puede establecer su programa.
- Para desplegar la función 40H del DOS, utilice el manejador 01 en el BX.
- Para la función 3FH del DOS en la entrada del teclado, utilice 00 en el BX. La operación incluye los caracteres Enter y Avance de Línea después de los caracteres tecleados en el área de entrada. No verifica que las entradas excedan el máximo que usted especificó.

## PREGUNTAS

9-1. ¿Cuáles son los valores hexadecimales para (a) la posición superior izquierda y (b) la posición inferior derecha en una pantalla de 80 columnas?

9-2. Codifique la instrucción para fijar el cursor en el renglón 12, columna 8.

9-3. Codifique las instrucciones para limpiar la pantalla, empezando en el renglón 12, columna 0 hasta el renglón 22, columna 79.

9-4. Codifique los datos y la función 09H de la INT 21H del DOS, para mostrar el mensaje "¿Cuál es la fecha (mm/dd/aa)?" Haga que una señal auditiva siga al mensaje.

9-5. Codifique los datos y la función OAH de la INT 21H del DOS, para aceptar entrada desde el teclado de acuerdo con el formato de la pregunta 9-4.

9-6. La sección titulada "Cómo limpiar el área de entrada" muestra cómo limpiar toda el área de entrada del teclado, definida como NAMEFLD. Cambie el ejemplo de modo que limpie sólo los caracteres que queden a la derecha de nombre más recientemente ingresado.

9-7. Teclee el programa de la figura 9.2 con los cambios siguientes: (a) En lugar del renglón 12, establezca el centro en el renglón 15; (b) en lugar de limpiar toda la pantalla, limpie sólo del renglón 0 al 15. Ensamble, enlace y pruebe el programa nuevo.

9-8. Identifique los manejadores de archivo estándar para (a) entrada del teclado; (b) despliegue normal en pantalla; (c) la impresora.

9-9. Codifique los datos y la función 40H de la INT 21H del DOS, para mostrar el mensaje "¿Cuál es la fecha (mm/dd/aa)?" Después del mensaje, envíe una señal auditiva.

9-10. Codifique los datos y la función 3FH de la INT 21H del DOS, para aceptar entrada desde el teclado de acuerdo con el formato de la pregunta 9-4.

9-11. Corrija el programa que se muestra en la figura 9-2 para utilizar las funciones 3FH y 40H de la INT 21H del DOS para entrada y despliegue. Ensamble, enlace y pruebe el programa nuevo.

CAPÍTULO **10**

# Procesamiento avanzado de la pantalla

OBJETIVO

Estudiar las características avanzadas de manejo de la pantalla, incluyendo recorrido, video inverso, intermitencia y gráficas a color.

## INTRODUCCIÓN

El capítulo 9 introdujo las características básicas concernientes al manejo de la pantalla y la entrada desde el teclado. Este capítulo trata las características avanzadas para los adaptadores de video, modos de configuración (texto o gráfico) y manejo de la pantalla. La primera sección describe los adaptadores comunes de video y sus áreas de despliegue de video asociadas.

Las secciones sobre el modo de texto explican el uso del byte de atributo para color, intermitencia e intensidad, así como las instrucciones para establecer el tamaño y posición del cursor, recorrer hacia arriba o hacia abajo de la pantalla y desplegar caracteres. Las últimas secciones explican el uso de los modos gráficos, junto con las distintas instrucciones usadas para su despliegue.

Este capítulo introduce los siguientes servicios ofrecidos por la INT 10H del BIOS:

| | |
|---|---|
| 00H | Establece el modo de video |
| 01H | Establece el tamaño del cursor |
| 02H | Establece la posición del cursor |
| 03H | Lee la posición del cursor |
| 04H | Lee la posición de la pluma óptica |
| 05H | Selecciona la página activa |

| | |
|---|---|
| 06H | Recorre la pantalla hacia arriba |
| 07H | Recorre la pantalla hacia abajo |
| 08H | Lee el atributo o carácter en la posición del cursor |
| 09H | Despliega el atributo o carácter en la posición del cursor |
| OAH | Despliega el carácter en la posición del cursor |
| OBH | Establece la paleta de colores |
| OCH | Escribe el pixel punto |
| ODH | Lee el pixel punto |
| OEH | Escribe en teletipo |
| OFH | Obtiene el modo actual de video |
| 11H | Genera carácter |
| 12H | Selecciona rutina alterna de pantalla |
| 13H | Despliega cadena de caracteres |
| 1BH | Regresa la información de funcionalidad o de estado |
| 1CH | Guarda o restaura el estado de video |

## ADAPTADORES DE VIDEO

Los más comunes adaptadores de video son:

| | |
|---|---|
| MDA | Adaptador de pantalla monocromática |
| HGC | Tarjeta de gráficos Hércules |
| CGA | Adaptador de gráficos en colores |
| EGA | Adaptador de gráficos mejorado |
| MCGA | Adaptador de gráficos en multicolores (PS/2 modelos 25 y 30) |
| VGA | Matriz de gráficos de video |

El VGA y sus clones super VGA reemplazaron a los adaptadores de video CGA y EGA. Programas escritos para un CGA o un EGA por lo común pueden correr con un sistema VGA, aunque programas escritos específicamente para VGA no corren en un CGA o un EGA.

El adaptador de video consta de tres unidades básicas: el controlador de video, el video de BIOS y el área de despliegue de video.

1. El *controlador de video,* esta unidad "es el caballo de batalla", genera las señales de rastreo del monitor para el modo seleccionado, texto o gráfico. El procesador de la computadora envía instrucciones a los registros del controlador y lee ahí la información de estado.
2. El *video de BIOS,* que actúa como una interfaz con el adaptador de video, contiene rutinas, como para establecer el cursor y desplegar caracteres.
3. El *área de despliegue de video* en memoria contiene la información que el monitor va a mostrar. Las interrupciones que manejan el despliegue en pantalla de forma directa transfieren a esta área sus datos. Las localidades del adaptador de video dependen de los modos de video que se estén usando. Para los adaptadores principales, a continuación están las direcciones del inicio de los segmentos de despliegue de video:
   - A000:[0] Utilizada para descripción de fuentes cuando está en modo de texto y para gráficos de alta resolución para EGA, MCGA y VGA
   - B000:[0] Modo de texto monocromático para MDA, EGA y VGA

- B100:[0] ParaHCG
- B800:[0] Modos de texto para CGA, MCGA, EGA y VGA y modos gráficos para CGA, EGA, MCGA y VGA.

El monitor gráfico de color RGB común permite la entrada de señales que son enviados a tres cañones de electrones (rojo, verde y azul, para cada uno de los colores primarios aditivos).

## ESPECIFICACIONES DEL MODO DE VIDEO

La función 00H, de la INT 10H de BIOS, puede designar el *modo* para el programa que se está ejecutando actualmente o puede conmutar entre texto y gráfico. Configurar el modo también limpia la pantalla. Como ejemplo, el modo 03 representa modo de texto, color y resolución de la pantalla, dependiendo del tipo de monitor.

Para designar un modo nuevo, solicite la INT 10H, con la función 00H en el registro AH y el modo en el AL. El ejemplo siguiente establece el modo de video en texto a color estándar en cualquier tipo de monitor a color (si intenta esta operación, notará que también es una forma rápida de limpiar la pantalla):

```
MOV   AH, 00H     ,• Petición para designar el modo
MOV   AL.03H      ;Texto o estándar a color,  80 x 25
INT   10H         ;Llama al BIOS
```

Si escribe programas para monitores de video desconocidos, puede utilizar la INT 10H, función OFH (tratada más adelante), la cual regresa en el AL el modo de video actual. Otro enfoque es usar la INT 11H de BIOS para determinar el dispositivo conectado al sistema, aunque la información enviada es muy primitiva. La operación regresa un valor al AX, con los bits 5 y 4 que indican el modo de video:

- 01:40 x 25, usando un adaptador de color
- 10:80 **X** 25, usando un adaptador de color
- 11:80 x 25, usando un adaptador monocromático.

Puede examinar el AX para saber el tipo de monitor y en consecuencia establecer el modo.

## MODO DE TEXTO

El modo de texto se utiliza para el despliegue normal en la pantalla de caracteres ASCII. El procesamiento es semejante tanto para monocromático como a color, salvo que a color no permite el atributo de subrayado. El modo de texto proporciona acceso a todo el conjunto de 256 caracteres ASCII extendido. La figura 10-1 muestra los modos de texto comunes, con el número de modo a la izquierda.

Modos de texto **00** (mono) y **01** (color). Estos modos permiten usar un formato de 40 columnas. Aunque fueron diseñados originalmente para el CGA, son compatibles los siguientes y también operan con funciones en sistemas EGA y VGA.

| Modo | Tamaño | Tipo | Adaptador | Resolución | Colores |
|---|---|---|---|---|---|
| 00 | (25 renglones, 40 cois) | Mono | CGA | 320 X 200 | |
| | | | EGA | 320 X 350 | |
| | | | MCGA | 320 X 400 | |
| | | | VGA | 360 X 400 | |
| 01 | (25 renglones, 40 cois) | Color | CGA | 320 X 200 | 16 |
| | | | EGA | 320 X 350 | 16 de 64 |
| | | | MCGA | 320 X 400 | 16 de 262,144 |
| | | | VGA | 360 X 400 | 16 de 262,144 |
| 02 | (25 renglones, 80 cois) | Mono | CGA | 640 X 200 | |
| | | | EGA | 640 X 350 | |
| | | | MCGA | 640 X 400 | |
| | | | VGA | 720 X 400 | |
| 03 | (25 renglones, 80 cois) | Color | CGA | 640 X 200 | 16 |
| | | | EGA | 640 X 350 | 16 de 64 |
| | | | MCGA | 640 X 400 | 16 de 262,144 |
| | | | VGA | 720 X 400 | 16 de 262,144 |
| 07 | (25 renglones, 80 cois) | Mono | MDA | 720 X 350 | |
| | | | EGA | 720 X 350 | |
| | | | VGA | 720 X 400 | |

Nota: MDA: Adaptador de pantalla monocromática
CGA: Adaptador de gráficos en color
MCGA: Arreglo de gráficos multicolores
VGA: Arreglo de gráficos de video

Figura 10-1 Modos de texto para despliegues en video

**Modos de texto 02 (mono) y 03 (color).** Estos modos proporcionan el formato convencional de 80 columnas. Aunque diseñados originalmente para el CGA, son compatibles con los posteriores y también funcionan con los sistemas EGA y VGA.

**Modo de texto 07 (mono).** Éste es el modo estándar monocromático para MDA, EGA y VGA y ofrece respetables resoluciones en pantalla.

### Byte de atributo

Un *byte de atributo* en modo de texto (no en modo gráfico) determina las características de cada carácter mostrado. Cuando un programa establece un atributo, permanece activado; esto es, todos los caracteres subsecuentes desplegados tienen el mismo atributo hasta que otra operación lo cambie. Puede utilizar las funciones de la INT 10H para generar un atributo de la pantalla y realizar acciones como recorrer hacia arriba, recorrer hacia abajo, leer un atributo o un carácter o desplegar un atributo o un carácter. Si utiliza DEBUG para ver el área de despliegue de video de su sistema, verá cada carácter de un byte, seguido de manera inmediata por su atributo de un byte.

El byte de atributo tiene el formato siguiente, de acuerdo con la posición del bit:

| | | Fondo | | | | Frente | | |
|---|---|---|---|---|---|---|---|---|
| Atributo: | BL | R | G | B | I | R | G | B |
| Número de bit: | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Las letras R,G y B indican las posiciones de bits para rojo, verde y azul, respectivamente.

- Bit 7 (BL) establece intermitencia
- Bits 6-4 determinan el *fondo* de la pantalla
- Bit 3 (I) establece la intensidad alta
- Bits 2-0 determinan *cífrente* o *primer plano* (para el carácter que será desplegado).

Los bits RGB definen un color (en color y en monocromático, 000 es negro y 111 es blanco). Por ejemplo un atributo con el valor 0000 0111 significa fondo negro con primer plano blanco.

### Despliegue monocromático

Para un monitor monocromático, el bit 0 establece el atributo de subrayado. Para especificar atributos, puede establecer combinaciones de bits como se muestra a continuación:

| Fondo | Frente | Característica | Fondo BLRGB | Frente IRGB | Hex |
|---|---|---|---|---|---|
| Negro | Negro | No despliega | 0 0 0 0 | 0 0 0 0 | 00H |
| Negro | Blanco | Normal | 0 0 0 0 | 0 1 1 1 | 07H |
| Negro | Blanco | Intermitencia | 1 0 0 0 | 0 1 1 1 | 87H |
| Negro | Blanco | Intenso | 0 0 0 0 | 1 1 1 1 | 0FH |
| Blanco | Negro | Video inverso | 0 1 1 1 | 0 0 0 0 | 70H |
| Blanco | Negro | Inverso, intermitente | 1 1 1 1 | 0 0 0 0 | F0H |
| | | Subrayado | 0 0 0 0 | 0 0 0 1 | 01H |

### Despliegue a color

En muchos monitores a color, el fondo puede mostrar uno de ocho colores y los caracteres pueden mostrar uno de 16 colores. La intermitencia e intensidad sólo se aplican al primer plano. También puede seleccionar uno de 16 colores para el borde (marco). Los monitores de color no permiten subrayado; en lugar de eso, al establecer un bit en 0 selecciona el color azul como primer plano.

El byte de atributo es utilizado de la misma manera como se mostró con un monitor monocromático. Los tres colores básicos son rojo, verde y azul. Puede combinarlos en el byte de atributo para formar un total de ocho colores (incluyendo blanco y negro) y puede establecer alta intensidad, para un total de 16 colores:

| Color | IRGB | Color | IRGB |
|---|---|---|---|
| Negro | 0 0 0 0 | Gris | 1 0 0 0 |
| Azul | 0 0 0 1 | Azul claro | l 0 0 1 |
| Verde | 0 0 1 0 | Verde claro | l 0 1 0 |
| Cian | 0 0 1 1 | Cian claro | 1 0 1 1 |
| Rojo | 0 1 0 0 | Rojo claro | 1 1 0 0 |
| Magenta | 0 1 0 1 | Magenta claro | 1 1 0 1 |
| Café | 0 1 1 0 | Amarillo | 1 1 1 0 |
| Blanco | 0 1 1 1 | Blanco brillante | 1 1 1 1 |

Si los colores del fondo y del primer plano son iguales, el carácter mostrado es invisible. También puede utilizar el byte de atributo para generar un carácter intermitente en el primer plano. Aquí están algunos atributos comunes:

| Fondo | Primer plano | Fondo BL R G B | Primer plano I R G B | Hex |
|---|---|---|---|---|
| Negro | Negro | 0 0 0 0 | 0 0 0 0 | 00 |
| Negro | Azul | 0 0 0 0 | 0 0 0 1 | 01 |
| Azul | Rojo | 0 0 0 1 | 0 1 0 0 | 14 |
| Verde | Cian | 0 0 1 0 | 0 0 1 1 | 23 |
| Blanco | Magenta claro | 0 1 1 1 | 1 1 0 1 | 7D |
| Verde | Gris (intermitente) | 1 0 1 0 | 1 0 0 0 | A8 |

Puede utilizar la INT 11H para determinar el tipo de monitor instalado. Después, para monocromático, use 07H para establecer el atributo normal (fondo negro, frente blanco); para color, utilice cualquiera de las combinaciones de colores descritas. El color queda activo hasta que otra operación lo cambia. El modo de texto permite usar las páginas de pantalla 0-3, en donde la página 0 es la pantalla normal.

Como ejemplo, la siguiente operación INT 10H (explicada más adelante) utiliza la función 09H para mostrar cinco asteriscos verde claro e intermitentes sobre fondo magenta:

```
MOV     AH,09H      /Solicita desplegar
MOV     AL,'*'      /Asterisco
MOV     BH,00H      /Página número 0
MOV     BL,0DAH     /Atributo de color
MOV     CX, 05      /Cinco veces
INT     10H         /Llama al BIOS
```

Puede utilizar DEBUG para revisar este ejemplo, así como para experimentar con otras combinaciones de colores.

## PÁGINAS DE PANTALLA

Los modos de texto le permiten almacenar datos en memoria de video en *páginas.* Los números de página son desde 0 hasta 3 para el modo normal de 80 columnas (y 0 hasta 7 para la raramente utilizada pantalla de 40 columnas). En modo de 80 columnas, la página número 0 es por omisión e inicia en el área de despliegue de video en B800[0], la página 1 inicia en B900[0], la página 2 en BA00[0] y la página 3 en BB00[0].

Puede formatear cualquiera de las páginas en memoria, aunque sólo puede desplegar una página a la vez. Cada carácter que se muestra en la pantalla necesita dos bytes de memoria: un byte para el carácter y un segundo byte para su atributo. De esta forma una página completa de caracteres, para 80 columnas y 25 renglones, necesita 80 x 25 x 2 = 4,000 bytes. La cantidad de memoria realmente asignada a cada página es 4K, o 4,096 bytes, así que después de cada página la siguen 96 bytes no utilizados.

## INTERRUPCIÓN 10H DEL BIOS PARA EL MODO DE TEXTO

Con anterioridad, usamos la función 00H de la INT 10H, para establecer el modo de despliegue. La INT 10H también tiene otros servicios (disponibles por medio de la función en el AH) para facilitar el manejo de toda la pantalla. La interrupción conserva el contenido de los registros BX, CX, DX, DI, SI y BP, pero no el AX, algo que debe recordar si utiliza la INT 10H en un ciclo. Las secciones siguientes describen cada función.

### INT 10H, función 00H: Establece modo de video

Como se describió antes, esto establece al AL con el modo, por lo común 03 para color o 07 para monocromático. (Véase la figura 10-1.)

### INT 10H, función 01H: Establece el tamaño del cursor

El cursor no es parte del conjunto de caracteres ASCII y sólo existe en modo de texto. La computadora mantiene su característico hardware para control del cursor, con operaciones especiales INT para su uso. El símbolo del cursor normal es similar a un carácter de subrayado, pero puede utilizar la función 01H de la INT 10H para ajustar el tamaño vertical del cursor. Establezca estos registros:

- CH (bits 4-0) = parte superior del cursor ("línea inicial de rastreo").
- CL (bits 4-0) = parte inferior del cursor ("línea final de rastreo").

Puede ajustar el tamaño del cursor entre la parte superior y la inferior: 0:14 para VGA, 0:13 para monocromático y EGA y 0:7 para CGA. Para un VGA, el código siguiente agranda el cursor desde la parte superior hasta la inferior:

```
MOV     AH, 01H     ,-Petición para designar el tamaño del cursor
MOV     CH.00       ,-Línea inicial de rastreo
MOV     CL, 14      ,-Línea final de rastreo
INT     10H         ;Llama al BIOS
```

Ahora el cursor parpadea como un rectángulo relleno. Puede ajustar su tamaño a cualquiera entre los límites establecidos, por ejemplo, 04:08, 03:10, etc. El cursor conserva sus atributos hasta que otra operación los cambie. Usando 0:14 (VGA), 12:13 (monocromático o EGA) o 6:7 (CGA) se restablece el cursor normal. Si no está seguro de los límites en su monitor, primero intente ejecutar una función 03H con DEBUG.

### INT 10H, función 02H: Establece la posición del cursor

Esta útil operación coloca el cursor en cualquier parte de la pantalla, de acuerdo con las coordenadas renglón:columna. Establezca estos registros:

- BH = Número de página, para modo de texto con 80 columnas, puede ser 0 (por omisión), 1,2 o 3.
- DH = Renglón
- DL = Columna

La posición del cursor en cada página es independiente de su posición en las otras páginas. Ese código coloca al cursor en el renglón 5, columna 20, para la página 0:

```
MOV  AH,02H      ;Petición para designar el cursor
MOV  BH, 00      ;Página número 0
MOV  DH, 05      ,-Renglón
MOV  DL, 20      ,-Columna
INT  10H         ;Llama al BIOS
```

### INT 10H, función 03H: Lee la posición del cursor

Un programa puede utilizar la función 03H para determinar el renglón, columna y tamaño actuales del cursor, en particular en situaciones en donde un programa tiene que utilizar la pantalla por un momento y tiene que guardar y restaurar la pantalla original. Coloque el número de página en el BH, sólo para la función 02H:

```
MOV  AH,03H     /Petición de colocar el cursor
MOV  BH,00      /Número de página 0 (normal)
INT  10H        /Llama al BIOS
```

La operación regresa estos valores:

- AX y BX = Sin cambio
- CH = Línea de rastreo inicial del cursor
- CL = Línea de rastreo final del cursor
- DH = Renglón
- DL = Columna

El ejemplo siguiente utiliza la función 03H para leer el cursor y determinar su posición y tamaño y después usa la función 02H para avanzar el cursor a la columna siguiente en la pantalla:

```
MOV  AH.03H     Petición de posición del cursor
MOV  BH,00      Página 0
INT  10H        Llama al BIOS
MOV  AH,02H     Coloca el cursor
INC  DL          en la columna siguiente
INT  10H        Llama al BIOS
```

### INT 10H, función 05H: Selección de la página activa

La función 05H permite establecer la página que será desplegada para los modos de texto 0-3 > 13-16. Puede crear páginas diferentes y pedir pasar de una página a otra. Las páginas en modo de 80 columnas son 0-3. Aquí está el código para esta función:

```
MOV  AH.05H      ;Petición de página activa
MOV  AL,#pág     ;Número de página
INT  10H         /Llama al BIOS
```

INT 10H, función 06H: Recorrer hacia arriba la pantalla

Cuando un programa de manera inadvertida despliega texto hacia abajo de la pantalla después de la parte inferior, la línea siguiente "sale" del inicio de la parte superior. Pero aun si la operación de interrupción especifica la columna cero, las líneas nuevas llevan sangría y las líneas subsecuentes pueden estar mal alineadas. La solución es recorrer la pantalla, de manera que las líneas desplegadas "salgan" por la parte superior y líneas en blanco aparezcan en la parte inferior.

Usted ya utilizó la función 06H, en el capítulo 9, para limpiar la pantalla. Colocar un número cero en el AL provoca que toda la pantalla se recorra hacia arriba, y en realidad se limpie. Establecer un valor diferente de cero en el AL provoca que ese número de línea se recorra hacia arriba. Cargue los registros siguientes:

- AL = Número de líneas o cero para toda la pantalla
- BH = Atributo
- CX = Renglón:columna iniciales
- DX = Renglón:columna finales

El código siguiente recorre toda la pantalla una línea y establece un atributo de color:

```
MOV  AX,0601H    /Recorre hacia arriba una línea
MOV  BH,30H      /Fondo en cian, con primer plano en negro
MOV  CX.OOOO     /Desde 00,00
MOV  DX,184FH    / hasta 24,79 (pantalla completa)
INT  10H         /Llama al BIOS
```

A continuación está el enfoque estándar para recorrer una sola línea:

1. Definir un elemento con nombre, por ejemplo ROW, inicializado en cero, para establecer la posición del renglón del cursor.
2. Desplegar una línea y avanzar el cursor a la línea siguiente.
3. Examinar para ver si ROW está cercano a la parte inferior de la pantalla (CMP ROW,22).
4. Si no es así, incrementar ROW (INC ROW) y salir.
5. Si es cierto, recorrer una línea, utilice ROW para colocar el cursor y hacer ROW igual a 00.

Los registros CX y DX permiten recorrer cualquier parte de la pantalla. Pero sea muy cuidadoso al hacer corresponder el valor de AL con la distancia en el CX:DX, en especial cuando haga referencia a una parte de la pantalla. Las instrucciones siguientes recorren cinco líneas, y en realidad crean una ventana en el centro de la pantalla con sus propios atributos:

```
MOV  AX,0605H    /Recorre cinco líneas
MOV  BH,S1H      /Fondo café, con primer plano azul
MOV  CX,0A1CH    /Desde el renglón 10, columna 28
```

```
MOV DX,0E34H     ; hasta el renglón 14, columna 52 (parte de pantalla)
INT 10H          ;Llama al BIOS
```

El ejemplo especifica un recorrido de cinco líneas, que es el mismo número que la distancia entre los renglones 10 y 14. Ya que el atributo para una ventana permanece hasta que otra operación lo cambie, al mismo tiempo puede establecer varias ventanas con diferentes atributos.

### INT **10H, función 07H: Recorrer hacia abajo la pantalla**

Para modo de texto, el recorrido hacia abajo de la pantalla provoca que las líneas inferiores desaparezcan por la parte inferior y aparezcan líneas en blanco en la parte superior. Cargue los registros siguientes igual que para la función 06H (recorrido hacia arriba):

- AL = Número de líneas, o cero para la pantalla completa
- BH = Atributo
- CX = Renglón: columna iniciales
- DX = Renglón:columna finales

### INT 10H, función 08H: Leer atributo o carácter en la posición del cursor

La función 08H puede leer tanto un carácter como su atributo del área de despliegue de video en los modos de texto o gráfico. Cargue el número de página normalmente, en **el** BH, como lo muestra el ejemplo siguiente:

```
MOV AH,08H     ;Petición de leer atributo o carácter
MOV BH,00      /Número de página 0 (normal)
INT 10H        ,-Llama al BIOS
```

La operación regresa el carácter en el AL y su atributo en el AH. En modo gráfico, para un carácter no ASCII la operación regresa OOH. Puesto que sólo se **lee** un carácter a la vez, tiene que codificar un ciclo para leer una sucesión de caracteres.

### INT 10H, función 09H: Desplegar atributo o carácter en la posición del cursor

Aquí está una operación divertida que despliega caracteres en modo de texto o gráfico con intermitencia, en video inverso y todo eso. Establezca los registros:

- AL = Un solo carácter ASCII que será desplegado cualquier número de veces
- BH = Número de página
- BL = Atributo
- CX = Número de veces que la operación despliega de manera repetida el carácter que esté en el AL.

A continuación veremos un ejemplo que despliega 80 guiones y establece un atributo d( color:

```
MOV  AH,09H          ,-Petición de despliegue
MOV  AL,'-'          ;Carácter que se despliega
MOV  BH, 0           ;Página número 0
MOV  BL,61H          ;Fondo café, primer plano azul
MOV  CX, 80          ;80 caracteres repetidos
INT  10H             ;Llama al BIOS
```

La operación no avanza el cursor ni responde al carácter de la campana, retorno de carro, avance de línea o tabulador; en lugar de eso, intenta desplegarlos como caracteres ASCII. El código siguiente despliega cinco corazones intermitentes con video inverso:

```
MOV  AH,09H         /Petición de despliegue
MOV  AL,03H         ,•Corazón (que será desplegado)
MOV  BH, 00         ,•Página número 0 (normal)
MOV  BL,0F0H        /Intermitencia y video inverso
MOV  CX, 05         ;Cinco veces
INT  10H            ;Llama al BIOS
```

El despliegue de caracteres diferentes requiere un ciclo. En modo de texto, pero no en el gráfico, los caracteres desplegados de manera automática van de una línea a la siguiente. Para desplegar una indicación o un mensaje, codifique una rutina que establezca el CX en 01 y cree un ciclo para mover un carácter a la vez desde la memoria al AL. (Como el CX está ocupado, no se puede usar con facilidad la instrucción LOOP.) También, después de desplegar cada carácter, utilice la función 02H de la INT 10H, para avanzar el cursor a la columna siguiente.

Puede utilizar esta operación para cambiar cualquier página de video válida y después utilizar la función 05H para desplegar la página.

### INT 10H, función OAH: Despliega un carácter en la posición del cursor

Esta operación despliega un carácter en modo de texto o gráfico. La única diferencia entre las funciones OAH y 09H en modo de texto es que la función OAH utiliza el atributo actual, mientras que la función 09H establece el atributo. Aquí está el código para esta función:

```
MOV   AH, OAH          /Petición de despliegue
MOV   AL,carácter      /Carácter que se despliega
MOV   BH,#página       /Número de página
MOV   CX,repetición    /Número de caracteres repetidos
INT   10H              /Llama al BIOS
```

Con frecuencia, las funciones de la INT 21H del DOS que pueden imprimir cadenas de caracteres y responder a los caracteres de control de la pantalla son más adecuadas que las operaciones del BIOS.

### INT 10H, función OEH: Escribir en teletipo

Esta operación le permite utilizar un monitor como terminal para despliegue simple. Establezca la función OEH en el AH, el carácter para desplegar en el AL, el número de página en el BH y el color del primer plano (modo gráfico) en el BL:

```
MOV  AH, OEH          ;Petición para desplegar
MOV  AL,carácter      ;Carácter que se despliega
MOV  BH,#página       .•Número de página activa (algunos sistemas)
MOV  BL,color         ;Color del primer plano (modo gráfico)
INT  10H              ;Llama al BIOS
```

Los caracteres de control de retroceso (08H), campana (07H), retorno de carro (ODH) y avance de línea (OAH) actúan como comandos para formatear la pantalla. De forma automática, la operación avanza el cursor y cuando llega al final de la línea, envía los caracteres a la línea siguiente, recorre la pantalla y mantiene los atributos presentes de la pantalla.

### INT 10H, función OFH: Obtiene el modo actual de video

Utilice esta función para determinar el modo actual de video. (Véase también la función OOH.) Aquí está un ejemplo:

```
MOV  AH, OFH          ; Petición de modo de video
INT  10H              ;Llama al BIOS
CMP  AL, 03           ,-Si el modo es 3,
JE   . . .            ,- entonces saltar
```

La operación regresa estos valores:

- AL = Modo actual de video
- AH = Caracteres por línea (20, 40 u 80, en donde 50H — 80)
- BH = Número de página actual

### INT 10H, función 11H: Generador de carácter

Esta complicada función para los sistemas EGA, MCGA y VGA inicia un modo establecido y restaura el ambiente de video. Una discusión está fuera del alcance de esta obra.

### INT 10H, función 12H: Selecciona la rutina alterna de pantalla

Esta función permite usar monitores EGA y VGA. Para obtener información sobre cualquiera de estos monitores, cargue 10H en el BL; la operación regresa:

- BH = 00H para color y 01H para monocromático
- BL = 00H para 64K, 01H para 128K, 02H para 192K y 03H para 256K
- CH = Bits del adaptador
- CL = Configuración de conmutación.

La operación permite usar varias funciones elaboradas para las computadoras del tipo PS/2, tal como 30H (selecciona líneas de rastreo), 31H (carga la paleta por omisión) y 34H (emulación de un cursor).

INT 10H, función 13H: Despliega una cadena de caracteres

Para monitores EGA y VGA, esta operación despliega cadenas con opciones de establecer el atributo y mover el cursor y actúa sobre los caracteres de control de retroceso, campana, retorno de carro y avance de línea. Los registros ES:BP deben contener la dirección segmento: desplazamiento de la cadena que se despliega. El código es como sigue:

```
MOV   AH,13H            ;Petición para desplegar
MOV   AL,subfunción     ;0, 1, 2 o 3
MOV   BH,#página         Número de página
MOV   BL,atributo       /Atributos de la pantalla
LEA   BP,dirección      /Dirección de la cadena en ES:BP
MOV   CX,longitud       /Longitud de la cadena de caracteres
MOV   DX,pantalla       /Posición relativa de inicio en la pantalla
INT   10H               /Llama al BIOS
```

Las cuatro subfuncionés en el AL son:
00 Despliega el atributo y la cadena; no avanza el cursor.
01 Despliega el atributo y la cadena; avanza el cursor.
02 Despliega el carácter y después el atributo, no avanza el cursor.
03 Despliega el carácter y después el atributo; avanza el cursor.

## USO DEL BIOS PARA DESPLEGAR EL CONJUNTO DE CARACTERES ASCII

El programa de la figura 9-1 utilizó la DOS INT 21H para desplegar el conjunto de caracteres ASCII, pero la operación *actuó sobre* los caracteres de control de retroceso, campana, retorno de carro y avance de línea, en lugar de *desplegarlos.* El programa corregido de la figura 10-2 ilustra el uso de la INT 10H del BIOS con las funciones siguientes:

0FH Obtiene el modo actual de video y lo guarda.
00H Para este programa, establece el modo de video 03 y al salir restaura el modo original.
08H Lee el atributo en la posición actual del cursor, para usarlo con la función 06H.
06H Recorre hacia arriba la pantalla para limpiarla usando el atributo para sólo leer. También crea una ventana de 16 líneas para los caracteres desplegados, con primer plano café y fondo azul.
02H Establece inicialmente el cursor, y lo avanza para cada carácter desplegado.

OAH En la posición actual del cursor, despliega cada carácter, incluyendo los caracteres de control.

Los caracteres son desplegados en 16 columnas y 16 renglones. Este programa, al igual que los otros en este libro, están escritos prefiriendo la claridad en lugar de la eficiencia en el procesamiento. Puede corregir el programa para hacerlo más eficiente, por ejemplo, usando los registros para el renglón, la columna y el generador de carácter ASCII. También, como la INT 10H sólo destruye el contenido del registro AX, los valores en los otros registros no tienen que volver a cargarse. Sin embargo, el programa no correría mucho más rápido y perdería algo de claridad.

## CARACTERES ASCII EXTENDIDOS

Entre los caracteres ASCII extendidos, 128-255 (80H-FFH) están varios caracteres especiales para despliegue de indicaciones, menús y logotipos. Por ejemplo, estos caracteres son usados para dibujar un rectángulo con líneas continuas sencillas y dobles:

```
TITLE     P10BIOAS (COM)   INT 10H para desplegar el conjunto de caracteres ASCII
          .MODEL  SMALL
          .CODE
          ORG     100H
BEGIN:    JMP     SHORT  MAIN
CTR       DB      00              Contador de caracteres ASCII
COL       DB      24              Columna de la pantalla
ROW       DB      04              Renglón de la pantalla
MODE      DB                      Modo de video
                  Procedimiento  principal:

MAIN      PROC    NEAR
          CALL    B10MODE        ;Obtiene/designa el modo de video
          CALL    C10CLR         ,-Limpia  la pantalla
A20:
          CALL    D10SET          Coloca el cursor
          CALL    E10DISP         Despliega  caracteres
          CMP     CTR,OFFH        ¿Es el último carácter desplegado?
          JE      A30               sí,  entonces salir
          INC     CTR             Incrementar el contador ASCII
          ADD     COL,02          Incrementar  la columna
          CMP     COL,56          ¿Se llegó a la última columna?
          JNE     A20               no,  entonces saltar
          INC     ROW               sí,  incrementar el renglón
          MOV     COL,24            y reiniciar  la columna
          JMP     A20
A30:
          CALL    Fl OREAD        Obtener carácter del teclado
          CALL    G10MODE         Restaurar el modo de video
          MOV     AX,4C00H        Salir al DOS
          INT     21H
MAIN      ENDP
                  Obtener y designar el modo de video

B10MODE   PROC    NEAR
          MOV     AH, OFH         ;Petición para obtener el modo
          INT     10H
          MOV     MODE, AL        ;Guardar el modo
          MOV     AH,OOH          /Petición para establecer un nuevo modo
          MOV     AL, 03          /Color estándar
          INT     10H
          RET
B10MODE   ENDP
```

Figura 10-2 INT 10H para desplegar el conjunto de caracteres ASCII

```
                Limpia la pantalla y crea una ventana:

C10CLR    PROC  NEAR
          MOV   AH,08H        Petición para obtener el atributo
          INT   10H             actual en AH
          MOV   BH, AH        Lo mueve al BH
          MOV   AX,0600H      Recorre toda la pantalla
          MOV   CX,0000       Posición superior izquierda
          MOV   DX,184FH      Posición inferior derecha
          INT   10H
          MOV   AX,0610H      Crea una ventana de 16 líneas
          MOV   BH,16H        Café sobre azul
          MOV   CX,0418H      Esquina superior izquierda en 04:24
          MOV   DX,1336H      Esquina inferior derecha en 19:54
          INT   10H
          RET
C10CLR    ENDP
                Coloca el cursor en el renglón y columna:

D10SET    PROC  NEAR
          MOV   AH,02H        Petición para colocar el cursor
          MOV   BH, 00        Página 0 (normal)
          MOV   DH,ROW        Nuevo renglón
          MOV   DL,COL        Nueva columna
          INT   10H
          RET
D10SET    ENDP
                Despliega caracteres ASCII:

E10DISP   PROC  NEAR
          MOV   AH, OAH       Despliega
          MOV   AL,CTR        carácter ASCII
          MOV   BH, 00        Página 0
          MOV   CX, 01        Un carácter
          INT   10H
          RET
E10DISP   ENDP
                Obliga a detenerse, obtiene un carácter del teclado

FlOREAD   PROC  NEAR
          MOV   AH,10H        ,• Petición para obtener un carácter
          INT   16H
          RET
FlOREAD   ENDP
                Restaura el modo de video original

310MODE   PROC  NEAR
          MOV   AH,00H        ;Petición para establecer el modo
          MOV   AL,MODE       ;Valor original
          INT   10H
          RET
310MODE   ENDP
          END   BEGIN
```

Figura 10-2 (continuación)

| Carácter | Línea sencilla | Línea doble |
|---|---|---|
| Ángulo de la esquina superior izquierda | DAH | C9H |
| Ángulo de la esquina superior derecha | BFH | BBH |
| Ángulo de la esquina inferior izquierda | C0H | C8H |
| Ángulo de la esquina inferior derecha | D9H | BCH |
| Línea continua horizontal | C4H | CDH |
| Línea continua vertical | B3H | BAH |

El código siguiente utiliza la función 09H de la INT 10H para dibujar una línea continua de 25 posiciones de longitud:

```
MOV  AH,09H          /Petición para desplegar
MOV  AL,0C4H         ,-Línea continua sencilla
MOV  BH,00           ;Página número 0
MOV  BL,OFH          ,-Frente negro, fondo blanco, intenso
MOV  CX,25           ;25 repeticiones
INT  10H             ;Llama al BIOS
```

Recuerde que la función 09H no avanza el cursor.

La manera más simple de desplegar una caja es definirla en el segmento de datos y desplegar toda el área. Este ejemplo define y despliega un menú en una caja con línea sencilla:

```
MENÚ     DB    0DAH  17 DUP(0C4H),     OBFH
         DB    0B3H  ' ADD records          0B3H
         DB    0B3H  ' Delete records       0B3H
         DB    0B3H  ' Enter orders         0B3H
         DB    0B3H  ' Print report         0B3H
         DB    0B3H  ' Update accounts      0B3H
         DB    0B3H  ' View records         0B3H
         DB    0C0H  17 DUP(0C4H),     0D9H

         MOV   AH,40H    [Petición para desplegar
         MOV   BX,01           ,-Manejador de archivo para la pantalla
         MOV   CX,152          ;Número de caracteres
         LEA   DX,MENÚ         .-Solicitud
         INT   21H
```

En el capítulo siguiente, la figura 11-1 despliega un menú semejante en una caja con líneas dobles. Los caracteres "con puntos" para crear sombras con frecuencia son utilizados a la derecha o abajo de una caja:

| Número | Carácter |
|---|---|
| BO | La cuarta parte de puntos activados (ligera) |
| B1 | La mitad de los puntos activados (media) |
| B2 | Tres cuartos de los puntos activados (oscura) |
| DBH | Sombra completa (negro) |

## INTERMITENCIA, VIDEO INVERSO Y RECORRIDO DE LA PANTALLA

El programa de la figura 10-3 acepta nombres desde el teclado y los despliega en la pantalla. Para hacer cosas más interesantes, despliega la petición en video inverso (azul sobre blanco), acepta el nombre en forma normal (blanco sobre azul) y despliega el nombre, con intermitencia y en video inverso, en la columna 40 en el mismo renglón. Aquí está el formato:

```
¿Nombre? Benjamín Franklin     Benjamín Franklín  [intermitente]
I                                I
Columna 0                      Columna 40
```

Para controlar la ubicación del cursor, el programa define ROW para incrementar el renglón en la pantalla y COL para avanzar el cursor cuando se despliega la petición y el nombre. (La función 09H de la INT 10H no avanza de manera automática el cursor.) El programa despliega hacia abajo de la pantalla hasta que alcanza el renglón 20 y después empieza a recorrerla una línea hacia arriba por cada petición adicional.

Para entrada desde el teclado, el procedimiento D10INPT utiliza la función OAH de la INT 10H.

```
          page    60,132
TITLE     P10NMSCR (EXE) Video inverso, intermitencia y recorrido de la
          .MODEL  SMALL
          .STACK  64
-------------------------------------------------------------------------
          .DATA
ÑAMEPAR   LABEL   BYTE              ;Lista de parámetros:
MAXNLEN   DB      20                  longitud máxima del nombre
ACTNLEN   DB      7                   número de caracteres ingresado
NAMEFLD   DB      20 DUP(' ')       ;  para el nombre

COL       DB      00
COUNT     DB
PROMPT    DB      'Ñame? '
ROW       DB      00
-------------------------------------------------------------------------
          .CODE
BEGIN     PROC    FAR
          MOV     AX,©data          ;Inicializa el registro
          MOV     DS,AX             ; de segmentos
          MOV     ES,AX
          MOV     AX,0600H
          CALL    Q10SCR            /Limpia la pantalla
A20LOOP:
          MOV     COL,00            ;Establece la columna a cero
          CALL    Q20CURS
          CALL    B10PRMP           ,-Muestra una indicación
          CALL    D10INPT           /Proporciona entrada de nombre
          CMP     ACTNLEN,00        ¿No hay nombre? (indica el final)
          JNE     A30
          MOV     AX,0600H
          CALL    Q10SCR            ,-Si es así, limpiar la pantalla,
          MOV     AX,4C00H          /Salir al DOS
          INT     21H
A30:
          CALL    E10NAME           /Desplegar nombre
          JMP     A20LOOP
BEGIN     ENDP
```

Figura 10-3 Intermitencia, video inverso y recorrido en la pantalla

```
                    Despliega la indicación
B10PRMP   PROC   NEAR
          LEA     SI,PROMPT          ;Designa la dirección de la indicación
          MOV     COUNT,05
B20:
          MOV     BL,71H             Video inverso
          CALL    F10DISP            Rutina de despliegue
          INC     SI                 Carácter siguiente de nombre
          INC     COL                Columna siguiente
          CALL    Q20CURS            Coloca el cursor
          DEC     COUNT              Cuenta descendente
          JNZ     B20                Repite el ciclo n veces
          RET
B10PRMP   ENDP
                    Acepta la entrada de un nombre
D10INPT   PROC    NEAR
          MOV     AH,OAH             Petición de entrada
          LEA     DX, ÑAMEPAR          desde el teclado
          INT     21H
          RET
D10INPT   ENDP
;                 Despliega el nombre en video inverso y con intermitencia:
E10NAME   PROC    NEAR
           LEA     SI,NAMEFLD         Inicializa el nombre
           MOV     COL,40            ,•Designa la columna de pantalla
 E20:
           CALL    Q20CURS            Coloca el cursor
           MOV     BL,0F1H            Video inverso e intermitencia
           CALL    F10DISP            Rutina de despliegue
           INC     SI                 Carácter siguiente en el nombre
           INC     COL                Siguiente columna de la pantalla
           DEC     ACTNLEN            Disminuye la cuenta de la longitud del nomb
           JNZ     E20                Repite el ciclo n veces

           CMP     ROW,20             ¿Cerca del borde inferior de la pantalla?
           JAE     E30
           INC     ROW                  no, incrementa el renglón
           RET
 E30:
           MOV     AX,0S01H
           CALL    Q10SCR               recorre la pantalla
           RET
 E10NAME   ENDP
                     Despliegue
 F10DISP    PROC     NEAR             ,-BL (atributo) se designa antes
            MOV      AH,09H           ;Petición de despliegue
            MOV      AL,[SI]          ,-Obtiene el carácter de nombre
            MOV      BH, 00           ;Número de página
            MOV      CX, 01           ;Un carácter
            INT      10H
            RET
 F10DISP    ENDP
 .                   Recorre la
 Q10SCR     PROC     NEAR             ;AX se designa antes
            MOV      BH,17H           ,-Blanco sobre azul
            MOV      CX,0000
            MOV      DX,184FH          Pantalla completa
            INT      10H
            RET
 Q10SCR     ENDP
                     Coloca el
```

Figura 10-3 (continuación)

```
Q20CURS   PROC   NEAR
          MOV     AH,02H
          MOV     BH, 00          Página
          MOV     DH,R0W          Renglón
          MOV     DL,COL          Columna
          INT     10H
          RET
Q20CURS   ENDP
          END     BEGIN
```

Figura 10-3 (continuación)

## DESPLIEGUE DIRECTO EN VIDEO

Para algunas aplicaciones puede ser muy lento el despliegue en video cuando es enviado a través del DOS y del BIOS. La manera más rápida de desplegar caracteres en pantalla (texto o gráficos) es transferirlos directamente al área de despliegue de video apropiada. Por ejemplo, la dirección de la página 0 en el área de video para el modo 03 (texto en color) es B800[0]H. Cada carácter en pantalla necesita dos bytes de memoria, uno para el carácter y el que le sigue de manera inmediata para su atributo. Con una pantalla de tamaño de 80 columnas y 25 renglones, una página en el área de video necesita 80 x 25 x 2 = 4,000 bytes.

Los primeros dos bytes en el área de despliegue de video representan una posición de la pantalla, para el renglón 00, columna 00, y los últimos bytes en F9EH y F9FH representan la posición en pantalla para el renglón 24, columna 79. Con sólo mover un carácter:atributo al área de video de la página activa, se provoca que el carácter aparezca de manera inmediata en la pantalla. Puede verificar esto con los comandos de DEBUG. Primero, despliegue el área de video en B800[0]H:

```
D B800:00
```

El despliegue muestra que estaba en la pantalla en el momento que tecleó el comando, lo cual por lo regular es un conjunto de bytes que contienen 20 07H (por carácter en blanco, fondo negro y primer plano blanco). Observe que DEBUG y usted están compitiendo por la misma área de despliegue y la pantalla. Trate de cambiar la pantalla con estos comandos para desplegar caritas felices en los renglones superiores e inferiores:

```
E B800:000 01 25 02 36 03 47

E B800:F90 01 25 02 36 03 47
```

El programa de la figura 10-4 da un ejemplo de transferencia directa de datos al área de despliegue de video en B900[0]H; esto es, la página 1, en lugar de la página cero por omisión. El programa utiliza la característica SEGMENT AT para definir el área de despliegue de video del BIOS, en realidad como un segmento ficticio. (Esto no es una violación de la regla de que un programa .COM sólo puede tener un segmento.) VID ÁREA identifica la posición en la página 01 al inicio del segmento.

El programa despliega caracteres en los renglones 5 hasta el 20 y en las columnas 10 hasta la 70. El primer renglón despliega una cadena del carácter A (41H) con un atributo de 01H, el segundo renglón despliega una cadena del carácter B (42H) con un atributo de 02H, y así sucesivamente, con el carácter:atributo incrementados para cada renglón.

```
                        TITLE     P10DRVID (EXE)  Despliegue directo en video
                                  .MODEL   SMALL
0000                    VIDSEG    SEGMENT AT 0B900H     Página 1 del área de video
0000  1000[?]           VIDARÉA   DB      1000H DUP(?)
1000                    VIDSEG    ENDS

                        ----------------------------------------------------------
                                  .STACK 64
                        ----------------------------------------------------------

                                  .CODE
0000                    BEGIN     PROC    FAR
0000  B8      R                  MOV     AX,VIDSEG      •Direccionabilidad para
0003  8E CO                     MOV     ES, AX           el área de video
                                  ASSUME  ES:VIDSEG
0005  B4 OF                     MOV     AH,OFH         | Petición para obtener
0007  CD 10                     INT     10H            |   y guardar
0009  50                       PUSH    AX             |   el modo actual
000A  53                       PUSH    BX             |   y la página
000B  B4 00                    MOV     AH,OOH         | Petición para designar
000D  BO 03                    MOV     AL,03          |   el modo 03, y limpiar la pantalla
OOOF  CD 10                    INT     10H;           |
0011  B4 05                    MOV     AH,05H         | Petición para designar
0013  BO 01                    MOV     AL,01H         |   la página #01
0015  CD 10                    INT     10H            |
0017  E8 002E R                CALL    CÍOPROC        | Procesa el área de video
001A  E8 004D R                CALL    E10INPT        | Proporciona entrada
001D  B4 05                    MOV     AH,05H         | Restaura
001F  5B                       POP     BX             |   el número de
0020  8A C7                    MOV     AL,BH          |   página original
0022  CD 10                    INT     10H            |
0024  58                       POP     AX             | ,-Restaura el modo
0025  B4 00                    MOV     AH,OOH         |   de video (en AL)
0027  CD 10                    INT     10H            |
0029  B8 4C00                  MOV     AX,4C00H       | Sale al DOS
002C  CD 21                    INT     21H            |
002E                    BEGIN     ENDP

002E                    C10PROC   PROC    NEAR
002E  BO 41                    MOV     AL,41H         | Carácter que se despliega
0030  B4 01                    MOV     AH,01H         | Atributo
0032  BF 0294                  MOV     DI,660         | Inicio del área de despliegue
0035  B9 003C           C30:    MOV     CX,60          | Caracteres por renglón
                        C40:    MOV     WORD PTR[VIDAREA+DI],AX
0038  25   89  35 0000  R                             | AX en el área de despliegue
003D  47                       INC     DI             | Siguientes posiciones
003E  47                       INC     DI             |   de video
003F  E2 F7                    LOOP    C40            | Repite 60 veces
0041  FE C4                    INC     AH             | Atributo siguiente
0043  FE CO                    INC     AL             | Carácter siguiente
0045  83 C7 23                ADD     DI, 40         | Sangría para el renglón siguiente
0048  3C 51                    CMP     AL,51H         •¿Último carácter a desplegar?
004A  75 E9                    JNE     C30            |   no, repetir
004C  C3                       RET                    |   sí, regresar
004D                    CIOPROC   ENDP

004D                    E10INPT   PROC    NEAR
004D  B4 10                    MOV     AH,10H         /Petición para entrada
004F  CD 16                    INT     16H
0051  C3                       RET
0052                    E10INPT   ENDP
                                  END     BEGIN
```

Figura 10-4 Despliegue directo en video

El programa establece la posición inicial de una página en el área de despliegue de video con base en el hecho de que hay 80 x 2 = 160 columnas en un renglón. Entonces la posición inicial para el renglón 10, columna 10, es (160 x 10 renglones) + (10 columnas x 2) = 660. Después de desplegar un renglón, el programa avanza 40 posiciones en el área de despliegue para el inicio de la línea siguiente y termina cuando llega a la letra Q (51H).

El segmento de despliegue de video para la página 1 está definido como VIDSEG y la página como VID ARE A. El programa establece el registro ES como el registro del segmento para VIDSEG. Al inicio, el programa guarda el modo y la página actuales y después establece el modo 03 y la página 01.

En el procedimiento C10PROC, el carácter y atributo iniciales son inicializados en el AX y el desplazamiento inicial del área de video en el DI. La instrucción MOV WORD PTR [VIDÁREA + DI],AX mueve el contenido del AL (el carácter) al primer byte del área de despliegue y el AH (el atributo) al segundo byte. La rutina LOOP ejecuta esta instrucción 60 veces y despliega el carácter:atributo en toda la pantalla. Después incrementa el carácter:atributo y añade 40 al DI: 20 para el final del renglón actual y 20 para sangrar el inicio del renglón siguiente (en la pantalla, 10 columnas cada vez). Después la rutina repite el despliegue del siguiente renglón de caracteres.

Al terminar el despliegue, el procedimiento E10INPT espera a que el usuario presione una tecla y después el programa restaura el modo y página originales.

## MODO GRÁFICO

Los adaptadores gráficos tienen dos modos básicos de operación: texto (por omisión) y gráfico. Utilice la función 00H de la INT 10H del BIOS para establecer el modo gráfico o de texto, como lo muestran los dos ejemplos siguientes:

**1.** Establece el modo gráfico para VGA:

```
MOV AH, O0H       ,-Petición para designar el modo
MOV AL,0CH        /Gráficos en color
INT 10H           /Llama al BIOS
```

**2.** Establece el modo de texto:

```
MOV AH,00H        /Petición para designar el modo
MOV AL,03H        /Texto en color
INT 10H           /Llama al BIOS
```

El EGA y el VGA proporcionan una resolución mucho mayor que el CGA original y son compatibles con él en muchas formas. Las resoluciones y modos para adaptadores gráficos están mostrados en la figura 10-5 y son como sigue:

- *Modos gráficos 04H, 05H y 06H.* La dirección del área de despliegue de video para estos modos es B800[0]. Éstos son los modos originales del CGA, que también son utilizados por los EGA y VGA por su compatibilidad con posteriores, de manera que programas escritos para el CGA pueden correr en un EGA o VGA.

| Modo | Tipo | Adaptador | Resolución | Colores |
|---|---|---|---|---|
| 04H | Color | CGA, EGA, MCGA, VGA | 320 X 200 | 4 |
| 05H | Mono | CGA, EGA, MCGA, VGA | 320 X 200 | |
| 06H | Mono | CGA, EGA, MCGA, VGA | 640 X 200 | |
| ODH | Color | EGA, VGA | 320 X 200 | 16 |
| OEH | Color | EGA, VGA | 640 X 200 | 16 |
| OFH | Mono | EGA, VGA | 640 X 350 | |
| 10H | Color | EGA, VGA | 640 X 350 | 16 |
| 11H | Color | MCGA,VGA | 640 X 480 | 2 de 262,144 |
| 12H | Color | VGA | 640 X 480 | 16 de 262,144 |
| 13H | Color | MCGA,VGA | 320 X 200 | 256 de 262,144 |

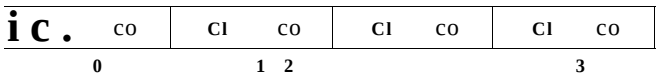
Figura 10-**5** Modos gráficos para despliegue en video

- *Modos gráficos ODH, OEH, OFH y 10H.* La dirección del área de despliegue de video para estos modos es A000[0]. Éstos son los modos originales del EGA, que también son usados por el VGA por su compatibilidad con posteriores, de manera que programas escritos para el EGA por lo común pueden correr en un VGA. También estos modos permiten usar 8,4, 2 y 2 páginas, respectivamente, del área de despliegue de video, por omisión con la página 0.
- *Modos gráficos 11H, 12H y 13H.* La dirección del área de despliegue de video para estos modos es A000[0]. Estos modos están diseñados específicamente para el VGA (y el ahora raro MCGA) y no se pueden usar con otros adaptadores de video.

En modo gráfico, la ROM contiene patrones de puntos sólo para los 128 caracteres (inferiores). La INT 1FH proporciona acceso a un área de memoria de 1K que define los 128 caracteres superiores, ocho bytes por carácter.

Pixeles

El modo gráfico utiliza *pixeles* (también llamados elementos gráficos o pels) para generar patrones en color. Por ejemplo, el modo 04H para gráficos en color estándar proporciona 200 renglones de 320 pixeles. Cada byte representa cuatro pixeles (esto es, dos bits por pixel), numerados de 0 a 3, como sigue:




En cualquier momento dado, existen cuatro colores disponibles, con números de 0 a 3. La limitación de cuatro colores es porque un pixel en dos bits provee de cuatro combinaciones: 00, 01, 10 y 11. Puede seleccionar el pixel 00 para cualquiera de los 16 colores disponibles para el fondo:

| Color | | Color | |
|---|---|---|---|
| Negro | 0000 | Gris | 1000 |
| Azul | 0001 | Azul claro | 1001 |
| Verde | 0010 | Verde claro | 1010 |
| Cian | 0011 | Cian claro | 1011 |
| Rojo | 0100 | Rojo claro | 1100 |
| Magenta | 0101 | Magenta claro | 1101 |
| Café | 0110 | Amarillo | 1110 |
| Gris claro | 0111 | Blanco | 1111 |

Y puede seleccionar los pixeles 01, 10 y 11 para cualquiera de las tres paletas de colores:

| C1 | CO | Paleta 0 | Paleta 1 |
|---|---|---|---|
| 0 | 0 | fondo | fondo |
| 0 | 1 | verde | cian |
| 1 | 0 | rojo | magenta |
| 1 | 1 | café | blanco |

Utilice la función OBH de la INT 10H para seleccionar una paleta de colores y el fondo. Así, si tiene que elegir fondo en color amarillo y la paleta 0, los colores disponibles son amarillo, verde, rojo y café. Un byte con el valor para pixeles 10101010 desplegaría todo como rojo. Si elige el fondo azul y la paleta 1, los colores disponibles son azul, cian, magenta y blanco. Un byte con el valor para pixeles 00011011 desplegaría azul, cian, magenta y blanco.

## INTERRUPCIÓN 10H DEL BIOS PARA GRÁFICOS

La INT 10H facilita el manejo completo de la pantalla para modo gráfico y modo de texto, como vimos. La operación preserva el contenido de los registros BX, CX, DX, DI, SI y BP, pero no el de AX. Las secciones siguientes describen cada una de las funciones de la INT 10H.

### INT 10H, función 00H: Establece el modo de video

La función 00H en el AH y el modo 12H en el AL establecen el modo estándar gráfico en color para el VGA:

```
MOV   AH,00H      ;Petición para designar el modo
MOV   AL,12H      ; con resolución 640 x 480 VGA
INT   10H         ,-Llama al BIOS
```

Establecer el modo gráfico hace que el cursor desaparezca.

### INT 10H, función 04H: Lee la posición de la pluma óptica

Utilice esta función con gráficos para determinar el estado de una pluma óptica. La operación regresa la información siguiente:

| | |
|---|---|
| AH | 0 si el estado es no funcionando, y 1 si es funcionando. |
| DX | Renglón en el DH y columna en el DL. |
| CH/BX | Posición de pixel, con línea (horizontal) de la malla en el BH y columna o punto en el BX. |

### INT 10H, función 08H: Lee el atributo o carácter en la posición del cursor

Esta función puede leer los caracteres y los atributos desde el área de despliegue tanto en modo de texto como en modo gráfico. Véase la sección anterior, "Interrupción 10H del BIOS para el modo de texto".

### INT 10H, función 09H: Despliega atributo o carácter en la posición actual del cursor

Para modo gráfico, utilice el BL para definir el color del primer plano. Si el bit 7 es cero, el color definido reemplaza los colores actuales presentes de pixeles; si el bit 7 es uno, el color definido es combinado (se le aplica un XOR) con ellos. Para detalles, vea la sección anterior, "Interrupción 10H del BIOS para el modo de texto".

### INT 10H, función OAH: Despliega un carácter en la posición del cursor

Véase la sección anterior, "Interrupción 10H del BIOS para el modo de texto".

### INT 10H, función OBH: Establece una paleta de colores

Utilice esta función para establecer la paleta de colores y desplegar un carácter gráfico. El número en el BH (00 o 01) determina el propósito del registro BL:

1. BH = 00. Selecciona el color del fondo, en donde el BL contiene el número del color en los bits 0-3 (cualquiera de 16 colores):

```
MOV AH, OBH      ;Petición

MOV BH,00        ; fondo

MOV BL,04        ,- color rojo

INT 10H          ;Llama al BIOS
```

2. BH = 01. Selecciona la paleta para gráficos, en donde BL contiene la paleta (0 o 1):

```
MOV   AH,OBH     Petición de color

MOV   BH,01      Selecciona la paleta

MOV   BL,00       número 0 (verde, rojo, café)

INT   10H        Llama al BIOS
```

Una vez que se selecciona una paleta, permanece activa. Pero cuando cambia la paleta, toda la pantalla cambia a esa combinación de colores. Si utiliza la función OBH mientras está en modo de texto, el número establecido para el color 0 de la paleta determina el color del borde.

### INT 10H, función OCH: Escribe un pixel punto

Utilice la función OCH para desplegar un color seleccionado (fondo y paleta). Establezca estos registros:

- AL = Color del pixel
- BH = Número de página (EGA o VGA)
- CX = Columna
- DX = Renglón.

El número mínimo para la columna o el renglón es 0 y el número máximo depende del modo de video. El ejemplo siguiente establece un pixel en la columna 50, renglón 70 en la pantalla:

```
MOV  AH.OCH        /Petición para escribir un punto
MOV  AL, 03        /Color del pixel
MOV  BH,0          /Página número 0
MOV  CX,50         /Posición horizontal  (columna)
MOV  DX.70         /Posición vertical  (renglón)
INT  10H           /Llama al BIOS
```

EGA/VGA modos ODH, OEH, OFH y 10H proporcionan 8, 4, 2 y 2 páginas de área de despliegue de video, respectivamente. La página por omisión es la número 0.

**INT 10H, función ODH: Lee un pixel punto**

Esta operación, la opuesta de la función OCH, lee un punto para determinar el número de su color. Establezca el BH con el número de página (EGA o VGA), el CX con la columna y el DX con el renglón. El número mínimo para la columna o el renglón es cero y el máximo depende del modo de video. La operación regresa el color del pixel en el AL.

**INT 10H, función OEH: Escribe en teletipo**

Véase la sección anterior, "Interrupción 10H del BIOS para el modo de texto".

**INT 10H, función 10H: Establece los registros de la paleta**

Esta función maneja los sistemas EGA y VGA. Un código de subfunción en el AL determina la operación:

- 00 Establece un registro de paleta, donde BH contiene el número a establecer y el BL el registro a establecer.
- 01 Establece el registro de rastreo, donde el BH contiene el número que se establece.
- 02 Establece todos los registros de paletas y de rastreo, ES:DX apunta a una tabla de 17 bytes, en donde los bytes 0-15 son números de paleta y el byte 16 es el número de rastreo.
- 03 Conmuta el bit para intensificar/intermitencia, donde 00 en el BL permite intensificar y 01 permite intermitencia.

Otras códigos de subfunciones AL para el VGA bajo la función 10H son 07H (lee registro individual de la paleta), 08H (lee el registro de rastreo), 09H (lee todos los registros de la paleta y de rastreo), 10H (establece un registro individual de color), 12H (establece un bloque de registros de color), 13H (selecciona una página de color), 15H (lee un registro individual de color), 17H (lee un bloque de registros de color) y 1AH (lee el estado de la página de color).

**INT 10H, función 1AH: Código de combinación de despliegue de lectura/escritura**

Esta operación regresa los códigos que identifican el tipo de despliegue que está en uso.

**INT 10H, función 1BH: Regresa la información de funcionalidad/estado**

Esta complicada operación regresa la información a un búfer de 64 bytes identificando el modo de video, tamaño del cursor, página a la que se le da soporte y así sucesivamente.

**INT 10H, función 1CH: Guarda o restaura el estado de video**

Esta función guarda o restaura el estado de video, incluyendo el estado de los registros de color, el área de datos del BIOS y el hardware del video.

## CÓMO ESPECIFICAR Y DESPLEGAR EL MODO GRÁFICO

El programa de la figura 10-6 utiliza varias funciones INT 10H, incluyendo las siguientes, para **el** despliegue de gráficos:

- OFH: Conserva el modo original
- OOH: Establece el modo gráfico
- OBH: Selecciona el fondo en color verde
- OCH: Escribe pixeles punto para 640 columnas y 350 renglones.

La pantalla actual desplegada es de 210 renglones y 512 columnas. Observe que los renglones y columnas están en términos de puntos, no de caracteres.

El programa incrementa el color para cada renglón (así que los bits 0000 se convierten en 0001, etc.) y como sólo los cuatro bits de la extrema derecha son utilizados, el color se repite después de 16 renglones. El despliegue inicia 64 columnas a partir de la izquierda de la pantalla y termina 64 columnas a partir de la derecha.

Al final, el programa espera a que el usuario presione una tecla, y después restaura el despliegue al modo original. Para un sistema VGA, podría experimentar con varios modos gráficos.

## DETERMINACIÓN DEL TIPO DE ADAPTADOR DE VIDEO

Ya que los adaptadores gráficos de video permiten el uso de varios servicios, hay ocasiones en que se necesita saber qué tipo de adaptador está instalado en un sistema. La manera recomendada es primero verificar si es VGA, después por EGA y por último CGA o MDA. Aquí están los pasos:

**1.** Para determinar si está instalado un VGA:

```
MOV  AH, 1AH        ,-Petición de la función VGA
MOV  AL, 0          ; y subfunción 0
INT  10H            ;Llama al BIOS
CMP  AL, 1AH        ;Si el AL contiene 1AH regresar
JE   VGAFOUND       ; el sistema contiene un VGA
```

**2.** Para determinar si está instalado un EGA:

```
MOV AH,12H          ;Petición de la función EGA
MOV BL, 10H         ,-Cantidad de memoria EGA
INT 10H             ;Llama al BIOS
```

```
TITLE     P10GRAFX (COM)   Despliegue gráfico
          .MODEL SMALL
          .CODE
          ORG     100H
BEGIN     PROC    NEAR
          MOV     AH, OFH          Conserva
          INT     10H                modo de video
          PUSH    AX                 original
          CALL    B10MODE          Designa el modo gráfico
          CALL    C10DISP          Despliegue gráfico en color
          CALL    D10KEY           Obtiene respuesta del teclado
          POP     AX               Restaura
          MOV     AH,00H             el modo original
          INT     10H                (en AL)
          MOV     AX,4C00H         Sale al DOS
          INT     21H
BEGIN     ENDP

B10MODE   PROC    NEAR
          MOV     AH.OOH         ,-Establece el modo gráfico EGA/VGA
          MOV     AL,10H         ;640 cois x 350 renglones
          INT     10H
          MOV     AH,OBH           Designa la paleta para el fondo
          MOV     BH, 00           Fondo
          MOV     BL,07H           Gris
          INT     10H
          RET
B10MODE   ENDP

C10DISP   PROC    NEAR
          MOV     BX, 00           ;Designa la página inicial,
          MOV     CX, 64           ,•  color, columna
          MOV     DX, 70           ;   y renglón
C20:
          MOV     AH,OCH           Escribe el pixel punto
          MOV     AL, BL           •Designa el color
          INT     10H              •Se conservan BX,  CX y DX
          INC     CX               •Incrementa la columna
          CMP     CX,576           •¿Es la columna 576?
          JNE     C20                no, repetir
          MOV     CX, 64             sí, restaurar la columna
          INC     BL               •Cambiar el color
          INC     DX               •Incrementa el renglón
          CMP     DX,280           •¿Es el renglón 280?
          JNE     C20                no, repetir
          RET                        sí, terminar
C10DISP   ENDP

D10KEY    PROC    NEAR
          MOV     AH,10H           ;Petición para entrada
          INT     16H              ;  desde el teclado
          RET
D10KEY    ENDP
          END     BEGIN
```

Figura 10-6 Despliegue gráfico en color

```
CMP  BL,10H        ;Si el BL ya no contiene 10H,

JNE  EGAFOUND      ; el sistema tiene un EGA
```

Ya que un EGA puede estar instalado junto con un MDA o un CGA, puede necesitar determinar si el EGA está activo. El área de datos del BIOS en 40:0087 contiene un byte de instrucción EGA. Verifique el bit 3, donde 0 significa que el EGA está activo y 1 significa que está inactivo.

3. Para determinar si está instalado un CGA o un MDA, examine la palabra en la localidad 40:0063, que contiene la dirección base del controlador de memoria. Observe que 3BxH significa MDA y 3DxH significa CGA.

## PUNTOS CLAVE

- El byte de atributo para modo de texto proporciona intermitencia, video inverso e intensidad. Para texto en color, los bits RGB permiten seleccionar colores pero no subrayado.
- La INT 10H de BIOS proporciona funciones para el procesamiento completo de la pantalla, como configurar el modo de video, establecer la posición del cursor, recorrido de la pantalla, lectura desde el teclado y escritura de caracteres.
- Si su programa despliega líneas en la parte inferior de la pantalla, utilice la función 06H de la INT 10H del BIOS para recorrer hacia arriba la pantalla antes de que el despliegue alcance la parte inferior.
- Para los servicios de la INT 10H que despliegan un carácter, tiene que avanzar el cursor y tal vez repetir el carácter en la pantalla.
- La memoria de 16K para despliegue en color permite almacenar "páginas" o "pantallas" adicionales. Existen cuatro páginas para pantallas de 80 columnas.
- La manera más rápida de desplegar caracteres en pantalla (texto o gráficos) es transferirlos de forma directa al área de video apropiada.
- Un pixel (elemento gráfico) consiste en un número especificado de bits, dependiendo del adaptador gráfico y de la resolución (baja, media o alta).
- Para los modos gráficos 04 y 05 puede seleccionar cuatro colores, de los cuales uno es cualquiera de los 16 colores disponibles y los otros tres son de una paleta de colores.

## PREGUNTAS

**10-1.** Proporcione los bytes de atributo, en binario, y para pantallas monocromáticas, para lo siguiente: (a) sólo subrayado: (b) blanco y negro, con intensidad normal; (c) video inverso, con intensidad alta.

**10**-2. Proporcione los bytes de atributo, en binario, para lo siguiente: (a) magenta sobre cian claro; (b) café sobre amarillo; (c) rojo sobre gris, intermitente.

**10-3.** Codifique las rutinas siguientes: (a) Establezca el modo monocromático de 80 columnas; (b) establezca el tamaño del cursor con inicio en la línea cinco y línea final 12; (c) recorra la pantalla hacia arriba 10 líneas; (d) despliegue 10 "puntos" intermitentes con medios puntos (hexadecimal Bl) encima.

**10-4.** En el modo de texto 03, ¿cuántos colores están disponibles para el fondo y el primer plano?

**10**-5. Codifique las instrucciones para desplegar cinco caracteres de diamante en modo de texto con verde claro sobre magenta.

**10-6.** ¿Qué modo le permite el uso de páginas de pantalla?

**10**-7. Escriba un programa que utilice la función OAH de la INT 21H, para aceptar datos desde el teclado y la función 09H para desplegar los caracteres. El programa limpiará la pantalla, establecerá los colores (selecciónelos) y aceptará un conjunto de datos desde el teclado empezando en la posición actual del cursor. El conjunto de datos podría ser de cuatro o cinco líneas (digamos, de una longitud de hasta 25 caracteres) ingresados desde el teclado, cada conjunto seguido de un Enter. Puede usar diferentes colores, video inverso o sonido, para experimentar. Después coloque el cursor en un

renglón y columna diferentes (usted decida) y despliegue los datos ingresados en esa posición. El programa sirve para aceptar cualquier número de conjuntos de datos. Puede terminar cuando el usuario presione Enter sin datos. Escriba el programa con una pequeña rutina con la lógica principal y una serie de subrutinas llamadas. Incluya algunos comentarios concisos.

**10-8.** Corrija el programa de la pregunta 10-7, de manera que utilice la INT 16H para entrada desde el teclado y la función 09H de la INT 10H para el despliegue.

**10-9.** Explique cómo el byte de atributo limita el número de colores disponibles.

**10-10.** Codifique las instrucciones para establecer el modo gráfico para estas resoluciones: (a) 320 x 200; (b) 640 x 200; (c) 640 x 480.

**10-11.** Codifique las instrucciones para seleccionar el fondo en azul en modo gráfico.

**10-12.** Codifique las instrucciones para leer un punto del renglón 12, columna 13 en modo gráfico.

**10-13.** Corrija el programa de la figura 10-6 de manera que proporcione lo siguiente: (a) un modo gráfico adecuado para su monitor; (b) fondo en color rojo; (c) renglón de inicio 10 y final en 30; (d) columna inicial en 20 y final en 300.

**10-14.** Con base en los cambios hechos en la pregunta 10-13, corrija el programa para desplegar una columna de puntos (en lugar de un renglón) a un tiempo. Esto es, despliegue puntos hacia abajo en la pantalla, después avance a la columna siguiente, y así sucesivamente.

## CAPÍTULO 11

# Procesamiento avanzado del teclado

OBJETIVO

**Estudiar todas las operaciones del teclado y las características avanzadas de entrada desde el teclado, incluyendo el estado del shift, el búfer del teclado y los códigos de rastreo.**

### INTRODUCCIÓN

Este capítulo describe las diferentes operaciones para manejo del teclado, algunas de las cuales tienen usos especializados. De estas operaciones la función OAH de la INT 21H (estudiada en el capítulo 9), y la INT 16H (estudiada en este capítulo) deben proporcionarle casi todas las operaciones con el teclado que usted necesitará.

Otros temas en el capítulo incluyen los bytes de estado del shift del teclado, códigos de rastreo y el área del búfer del teclado. Los bytes de estado del shift en el área de datos del BIOS permiten a un programa determinar, por ejemplo, si las teclas Ctrl, Shift o Alt han sido presionadas. El código de rastreo es un número único asignado a cada tecla en el teclado que permite al sistema identificar el origen de una tecla presionada y permite a un programa verificar las teclas de función extendidas, como Inicio, AvPág y Flechas. Y el área del búfer del teclado ofrece espacie en memoria para que usted teclee por adelantado antes de que un programa solicite en realidad *um* entrada.

Las operaciones introducidas en este capítulo son las siguientes:

**FUNCIONES DE LA INT 21H DEL DOS**

01H Entrada desde el teclado con repetición en la pantalla

| | |
|---|---|
| 06H | E/S directa a la consola |
| 07H | Entrada directa desde el teclado sin repetición |
| 08H | Entrada desde el teclado sin repetición en pantalla |
| OAH | Entrada al búfer del teclado |
| OBH | Verificación del estado del teclado |
| OCH | Limpiar el búfer del teclado y llamar una función |

**FUNCIONES DE LA INT 16H DEL BIOS**

| | |
|---|---|
| 00H | Lee un carácter |
| 01H | Determina si un carácter está presente |
| 02H | Regresa el estado actual del shift |
| 05H | Escribe en el teclado |
| 10H | Lee un carácter desde el teclado |
| 11H | Determina si un carácter está presente |
| 12H | Regresa el estado actual del shift del teclado |

## EL TECLADO

El teclado proporciona tres tipos básicos de teclas:

1. Las letras desde la A hasta la Z, los números desde el 0 hasta el 9 y caracteres como %, $ y #.
2. Las teclas extendidas de función, que consisten en:
   - Teclas de función de programa (Fl, etc., Shift+F1, etc.).
   - Teclas del panel numérico con BloqNum apagado (Inicio, Fin, Flechas, Supr, Ins, RePág y AvPág) y las teclas repetidas en el teclado de 101 teclas.
   - Alt + letras y Alt+teclas de función de programa.
3. Teclas de control para Alt, Ctrl y Shift, que funcionan en conjunción con otras teclas. El BIOS las trata de manera diferente de las otras teclas actualizando su estado actual en los bytes de estado del shift en el área de datos de BIOS. El BIOS no las envía como caracteres ASCII a su programa.

La PC original con sus 83 teclas sufrió la consecuencia de una decisión miope que provocó que las teclas en el llamado panel (o teclado) numérico realizaran dos acciones. Así, los números compartían teclas con Inicio, Fin, Flechas, Supr, Ins, RePág y AvPág, con la tecla BloqNum para conmutar entre ellas. Para resolver este problema, los diseñadores produjeron el teclado extendido con 101 teclas. De las 18 teclas nuevas, sólo dos, FU y F12, proporcionan una función nueva; el resto duplican la función de teclas en el teclado original. Si sus programas permiten presionar FU, F12 o alguna de las nuevas combinaciones de teclas, los usuarios deben tener un teclado ampliado y una computadora con un BIOS que pueda procesarlas. Para la mayoría de las otras operaciones con el teclado, sus programas no necesitan interesarse en el tipo de teclado instalado.

## ESTADO DEL SHIFT DEL TECLADO

El área de datos del BIOS en el segmento 40[0]H contiene varios elementos útiles. Éstos incluyen el primer byte del estado actual del shift del teclado en 40:17H en donde, cuando está en uno, los bits indican lo siguiente:

| Bit | Acción | Bit | Acción |
|---|---|---|---|
| 7 | Inserción activa | 3 | Alt presionada |
| 6 | Estado de BloqMayús activa | 2 | Ctrl presionada |
| 5 | Estado de BloqNum activa | 1 | Shift izquierdo presionado |
| 4 | Estado de Scroll Lock activa | 0 | Shift derecho presionado |

Puede utilizar la función 02H (estudiada más adelante) de la INT 16H para examinar estos valores. Note que "activa" significa que el usuario en ese momento está manteniendo oprimida la tecla; al soltar la tecla pone en cero el valor del bit. El teclado de 83 teclas sólo necesita este byte de estado del shift.

El teclado ampliado de 101 teclas tiene teclas Ctrl y Alt duplicadas (izquierdas y derechas), de modo que se necesita información adicional para examinarlas. El segundo byte de estado del teclado necesario para el teclado de 101 teclas está en 40:18H, en donde un bit en uno indica lo siguiente:

| Bit | Acción | Bit | Acción |
|---|---|---|---|
| 7 | Ins presionada | 3 | Ctrl/BloqNum (pausa) activa |
| 6 | BloqMayús presionada | 2 | SysReq presionada |
| 5 | BloqNum presionada | 1 | Alt izquierda presionada |
| 4 | Scroll Lock presionada | 0 | Ctrl izquierda presionada |

Los bits 0, 1 y 2 están asociados con el teclado ampliado (de 101 teclas). Ahora puede, por ejemplo, examinar si está presionada Ctrl o Alt o ambas.

Otro byte de estado del teclado se encuentra en 40:96H. Aquí el elemento de interés para nosotros es el bit 4; cuando está en uno, indica que está instalado un teclado de 101 teclas.

### Ejercicio con el estado del shift

Para ver el efecto de las teclas Ctrl, Alt y Shift sobre los bytes de estado del shift, cargue DEBUG para ejecución. Introduzca D 40:17 para ver el contenido de los bytes de estado. Presione las teclas BloqMayús, BloqNum y ScrollLock y **otra:** vez introduzca D 40:17 para ver el resultado en ambos bytes de estado. El byte 40:17H debe mostrar 70H (0111 0000B) y el byte en 40:18H es quizá OOH. El byte en 40:96H debe mostrar la presencia (o ausencia) de un teclado de 101 teclas.

Intente cambiar el contenido del byte de estado en 40:17H —introduciendo E 40:17 00. Si su teclado tiene indicadores luminosos para las teclas de bloque, deben apagarse. Ahora intente introduciendo E 40:17 70 para volverlas a encender.

Debe intentar con diferentes combinaciones, aunque es difícil teclear un comando válido DEBUG mientras mantiene oprimidas las teclas Ctrl y Alt. Introduzca Q para salir de DEBUG.

## BÚFER DEL TECLADO

Un elemento de interés en el área de datos del BIOS en 40:1EH es el *búfer del teclado.* Esta característica nos permite teclear hasta 15 caracteres antes que el programa solicite alguna entrada. Cuando presiona una tecla, el procesador del teclado genera el código de rastreo de la tecla (su único número asignado) y de manera automática solicita la INT 09H.

En términos sencillos, la rutina INT 09H del BIOS obtiene el código de rastreo del teclado, lo convierte en un carácter ASCII y lo envía al área del búfer del teclado. A continuación, la INT 16H del BIOS (la operación de más bajo nivel del teclado) lee el carácter del búfer y lo envía a su programa. Su programa nunca necesita solicitar la INT 09H, ya que el BIOS lo hace de forma automática cuando usted presiona una tecla. Una sección posterior cubre la INT 09H y el búfer del teclado con mayor detalle.

## INTERRUPCIÓN 21H DEL DOS PARA ENTRADA DESDE EL TECLADO

Esta sección trata los servicios del DOS que manejan entrada del teclado. Todas estas operaciones, excepto la función OAH, sólo aceptan un carácter. (Para manejar una cadena de caracteres, debe codificar un ciclo que acepte un carácter, verificar las teclas de Retroceso y Enter, si es necesario, repita el carácter en la pantalla y avance el cursor.) Para entrada desde el teclado con el DOS, inserte una función en el AH y solicite la INT 21H. En el estudio de las operaciones que siguen, el término "responder a una petición Ctrl + Break" significa que el DOS terminará el programa si el usuario presiona juntas Ctrl+Break o Ctrl+C. Estas operaciones han sido sustituidas por la función 3FH (estudiada en el capítulo 10), pero para que el estudio esté completo se incluyen aquí.

### Función 01H, de la INT 21H: Entrada del teclado con eco (repetición en pantalla)

Esta operación acepta un carácter desde el búfer del teclado o, si no está presente ninguno, espera una entrada del teclado. La operación regresa uno de dos códigos de estado:

- AL = un número distinto de cero significa que un carácter ASCII estándar está presente, como una letra o un número, que la operación repite en la pantalla.
- AL = cero significa que el usuario ha presionado una tecla de función extendida, como Inicio, Fl o RePág, y el AH aún tiene la función original. La operación maneja las funciones ampliadas de manera ineficiente, intentando enviarlas a la pantalla. Y para obtener el código de rastreo para la tecla de función en el AL, tiene que repetir de manera inmediata la operación INT 21H. La operación también responde a una petición Ctrl+Break.

El código siguiente ilustra esta función:

```
MOV  AH,01H     ;Petición de entrada del teclado
INT  21H        ;Llama al DOS
CMP AL, 0 0     ;¿Se presionó una tecla de función?
JNZ               no,  entonces es un carácter ASCII
```

```
INT 21H        ; sí, entonces repite la operación
...            ; para el código de rastreo
```

**Función** 06H, **de la INT** 21H:E/S **directa de la consola**

Esta operación desconocida, si no rara, puede transferir cualquier carácter o código de control sin interferencia del DOS. Existen dos versiones, para entrada y para salida. Para entrada, carga OFFH en el DL. Si ningún carácter está en el búfer del teclado, la operación pone en uno la bandera de cero y no espera entrada. Si un carácter está esperando en el búfer del teclado, la operación almacena el carácter en el AL y pone en cero la bandera del cero. La operación no repite en la pantalla el carácter y no verifica por Ctrl + Break o Ctrl + PtSc. Un número diferente de cero en el AL representa un carácter ASCII estándar, como una letra o un número. Cero en el AL significa que el usuario ha presionado una tecla de función tal como Inicio, Fl o RePág. Para obtener el código de rastreo en el AL, repita de manera inmediata la operación INT 21H:

```
MOV  AH,06H      /Petición directa a la consola
MOV  DL,OFFH     ;Entrada del teclado
INT  21H         ,•Llama al DOS
JZ   KIO         ,-Repetir si el búfer está vacío
CMP  AL, 00      ;¿Se presionó una tecla de función?
JNZ  K30         ; no, entonces es un carácter ASCII
INT  21H         ; sí, entonces repite la operación
                   para el código de rastreo
```

Para salida en la pantalla, cargue el carácter ASCII (no OFFH) en el DL.

**Función** 07H **de la INT** 21H: **Entrada directa desde el teclado sin repetición en la pantalla**

Esta operación funciona igual que la función 01H, excepto que el carácter ingresado no se repite en la pantalla y la operación no responde a una petición Ctrl+Break. Podría utilizar la operación para introducir una contraseña (o password) que sea invisible o en donde no quiere que la pantalla sea perturbada.

**Función** 08H **de la INT** 21H: **Entrada desde el teclado sin repetición en la pantalla**

Esta operación funciona igual que la función 01H, salvo que el carácter ingresado no se repite en la pantalla.

**Función** OAH **de la INT** 21H: **Entrada del teclado mediante el** búfer

Está operación útil del teclado es estudiada con detalle en el capítulo 9. Sin embargo, su capacidad está limitada por no poder aceptar teclas de función extendida.

**Función** OBH **de la INT** 21H: **Verificación del estado del teclado**

Esta operación regresa FFH en el AL si un carácter está disponible y OOH si ningún carácter está disponible. La función está relacionada a aquellas otras que no esperan por entrada del teclado.

### Función OCH de la INT 21H: Limpia el búfer del teclado y llama a una función

Puede utilizar esta operación en asociación con la función 01H, 06H, 07H, 08H o OAH. Cargue la función que necesite en el AL:

```
MOV   AH,OCH          Petición de entrada del teclado
MOV   AL,función      Función que se necesita
MOV   DX,KBAREA       Área de entrada del teclado
INT   21H             Llama al DOS
```

La operación limpia el búfer del teclado, ejecuta la función que está en AL, y acepta (o espera) un carácter, de acuerdo a la petición en AL. Podría utilizar esta operación para un programa que no permite que el usuario teclee por adelantado.

## INTERRUPCIÓN 16H DEL BIOS PARA ENTRADA DESDE EL TECLADO

La INT 16H del BIOS, la operación básica de teclado del BIOS utilizada de manera extensiva por desarrolladores de software, proporciona los servicios siguientes de acuerdo con la función que esté en el AH.

### Función 00H de la INT 16H: Lee un carácter

Esta operación maneja las teclas del teclado de 83 teclas, pero no acepta entrada de las teclas adicionales en el teclado ampliado de 101 teclas. (Para una entrada que pueda utilizar todo el teclado, vea la función 10H.)

La operación verifica el búfer del teclado por la entrada de un carácter. Si ninguno está presente, la operación espera a que el usuario presione una tecla. Si un carácter está presente, la operación lo regresa en el AL y su código de rastreo en el AH. (Una sección posterior cubre los códigos de rastreo.) Si la tecla presionada es una función extendida, como Inicio o F1, el carácter en el AL es 00H. Aquí están las dos posibilidades:

| Tecla presionada | AH | AL |
|---|---|---|
| Carácter ASCII normal: | Código de rastreo | Carácter ASCII |
| Tecla de función extendida: | Código de rastreo | 00H |

El siguiente código examina el AL contra 00H para determinar si el usuario ha presionado una tecla de función extendida:

```
MOV    AH.00H      ;Petición al BIOS de entrada desde el teclado
INT    16H         ,• Llama al BIOS
CMP    AL,00H      ;¿Es una tecla de función extendida?
JE     G40         ; si
```

Como la operación no repite el carácter en la pantalla, tiene que emitir una interrupción de despliegue en pantalla para ese propósito.

### Función 01H de la INT 16H: Determina si un carácter está presente

Esta operación es semejante a la función OOH, pero con una diferencia importante. Si un carácter ingresado está presente en el búfer del teclado, la operación pone en cero la bandera **del** cero (ZF = 0) y envía el carácter al AL y su código **de** rastreo al AH; el carácter ingresado permanece en el búfer. Si no está presente algún carácter, la operación pone en uno la bandera del cero y no espera. Observe que la operación proporciona una característica de anticipación, ya que el carácter permanece en el búfer del teclado hasta que la función OOH **lo** lee.

### Función 02H de la INT 16H: Regresa el estado actual de las teclas shift

Esta operación regresa a AL el estado de la tecla shift del teclado desde el área de datos del BIOS en la localidad 417H (40:17H). (Una sección anterior describe el byte de estado.) El código siguiente examina si la tecla shift izquierda (bit 1) o derecha (bit 0) están presionadas:

```
MOV  AH,02H          Petición de estado del shift
INT  1SH             Llama al BIOS
OR   AL,00000011B    ¿Se presionó el shift izq. o der?
JE   xxxx            -sí
```

Véase la función 11H para manejo del estado del shift en la localidad 418H para funciones extendidas en el teclado ampliado.

### Función 05H de la INT 16H: Escritura en el teclado

Esta operación permite que su programa inserte caracteres en el búfer del teclado como si el usuario hubiera presionado alguna tecla. Cargue el carácter ASCII al CH y su código de rastreo al CL. La operación le permitirá ingresar caracteres en el búfer hasta que esté lleno.

### Función 10H de la INT 16H: Lectura de un carácter del teclado

La operación es la misma que la de la función OOH, salvo que también acepta las teclas adicionales de función extendidas (como Fl 1 y F12) desde el teclado ampliado, mientras que la función OOH no lo permite.

La operación verifica el búfer del teclado para un carácter ingresado. Si ninguno está presente, la operación espera a que el usuario presione una tecla. Si un carácter está presente, la operación lo regresa en el AL y su código de rastreo en el AH. Si la tecla presionada es una tecla de función extendida, como Inicio o Fl, el carácter en el AL es OOH. En el teclado ampliado, Fl 1 y F12 también regresan OOH en el AL, pero otras teclas de control (duplicados), como Inicio y RePág, regresan EOH. Aquí están las dos posibilidades:

| Tecla presionada | AH | AL |
|---|---|---|
| Carácter ASCII normal: | Código de rastreo | Carácter ASCII |
| Tecla de función extendida: | Código de rastreo | OOH o EOH |

Puede examinar el AL contra 00H o EOH para determinar si el usuario ha presionado una tecla de función extendida:

```
MOV  AH,10H   ;Petición al BIOS para una entrada del teclado
INT  16H      ;Llama al BIOS
CMP  AL,00H   ;¿Es una tecla de función extendida?
JE   G4   0   ; -sí
CMP  AL,OEOH  ;¿Es una tecla de función extendida?
JE   G40      ; -sí
```

Ya que la operación no repite el carácter en la pantalla, debe emitir una interrupción de despliegue en pantalla para ese propósito.

### Función 11H de la INT 16H: Determina si está presente un carácter

Esta operación es la misma que la función 01H, excepto que reconoce las funciones extendidas del teclado ampliado, mientras que 01H no lo hace.

### Función 12H de la INT 16H: Regresa el estado presente del shift del teclado

Esta operación es semejante a la función 02H, que regresa al AL el estado del shift del teclado desde el área de datos del BIOS en la localidad 417H (40:17H). La operación también envía el estado del shift extendido a AL:

| Bit | Acción | Bit | Acción |
|---|---|---|---|
| 7 | SysReq presionada | 3 | Alt derecha presionada |
| 6 | BloqMayús presionada | 2 | Ctrl derecha presionada |
| 5 | BloqNum presionada | 1 | Alt izquierda presionada |
| 4 | ScrollLock presionada | 0 | Ctrl izquierda presionada |

## TECLAS DE FUNCIÓN EXTENDIDAS Y CÓDIGOS DE RASTREO

Una tecla de función extendida como F i o Inicio solicita una acción en lugar de enviar un carácter. No existe nada en el diseño del sistema que obligue a estas teclas a realizar una acción específica: como programador, usted determina, por ejemplo, que presionando la tecla Inicio se coloque el cursor en la esquina superior izquierda de la pantalla o que presionando la tecla Fin coló- que el cursor al final del texto de la pantalla. Podría programar con facilidad estas teclas para que realicen operaciones sin relación alguna.

Cada tecla tiene un *código de rastreo* diseñado, empezando con 01 para Esc. (Véase en el apéndice F una lista completa de estos códigos.) Por medio de los códigos de rastreo, un programa puede determinar el origen de cualquier tecleo. Por ejemplo, un programa podría emitir la función 10H de la INT 16H para solicitar la entrada de un carácter. La operación responde en una

de dos formas, dependiendo de si presiona una tecla de carácter o una tecla de función extendida. Para un carácter, como la letra A, la operación envía estos dos elementos:

1. En el registro AL, el carácter ASCII de la A (41H).
2. En el registro AH, el código de rastreo para la letra A, 1EH.

| AH | AL |
|---|---|
| IE | 41 |

El teclado tiene dos teclas para caracteres tales como -, + y *. Por ejemplo, presionando la tecla del asterisco se establece el código del carácter en 2AH en el AL y uno de dos códigos de rastreo en el AH, dependiendo de qué tecla fue presionada: 09H para el asterisco que está arriba del número 8, o 29H para el asterisco del panel numérico.

El código siguiente prueba el código de rastreo para determinar qué asterisco fue presionado:

```
CMP  AL,2AH      ;¿Es un asterisco?
JNE  EXIT1       ,- no, entonces salir
CMP  AH,09H      ¿Cuál es el código de rastreo?
JE   EXIT2
```

Si presiona una tecla de función extendida, como Ins, la operación envía estos dos elementos:

1. En el registro AL: Cero, o EOH para una nueva tecla de control en teclado ampliado.
2. En el registro AH: El código de rastreo para Ins, 52H.

| AH | AL |
|---|---|
| 52 | 00 |

Por tanto, luego de una operación INT 16H (y algunas operaciones de la INT 21H), se puede examinar el AL. Si contiene OOH o EOH, la petición es para una función extendida; de otra manera, la operación ha enviado un carácter. Lo siguiente prueba una tecla de función extendida:

```
MOV   AH, 10H      ,-Petición para entrar desde el teclado
INT   16H          :Llama al BIOS
CMP   AL,OOH       ;¿Es una función extendida?
JZ    salir        ; sí, entonces salir
CMP   AL.0E0H      ;¿Es una función extendida?
JZ    salir        ; sí, entonces salir
```

En el código siguiente, si un usuario presiona la tecla Inicio (código de rastreo 47H), el cursor se coloca en el renglón 0, columna 0:

```
MOV  AH,10H   ;Petición de entrada
INT  1SH      ,• Llama al BIOS
CMP  AL,00H   ; ¿Es una función extendida?
JE   G30      ; sí entonces pasarlo
CMP  AL,EOOH  ; ¿Es una función extendida?
JNE  G90      ; no entonces salir
CMP  AH,47H   ; ¿Es el código de rastreo de
JNE  G90      ; no entonces salir
MOV  AH,02H   ;Petición
MOV  BH,00    ; para colocar el cursor
MOV  DX,00    ; en 0,0
INT  10H      ;Llama al BIOS
```

Las teclas de función programable F1-F10 generan códigos de rastreo 3BH-44H, respectivamente, y FU y F12 generan 85H y 86H. El código siguiente prueba la tecla de función programable FIO:

```
CMP AH,44H   ;¿Es la tecla de función FIO?
JE  EXIT1    ; sí, entonces salir
```

En EXIT1, el programa podría realizar cualquier acción necesaria.

### Ejercicio del teclado

El ejercicio siguiente con DEBUG examina los efectos de introducir varios caracteres con el teclado. Para un teclado de 83 teclas, utilice la función 00H; para un teclado de 101 teclas, emplee la función 10H. Utilice el comando A 100 para introducir estas instrucciones:

```
MOV  AH.00 CN MOV AH.10
INT  1S
JMP  10 0
```

Utilice el comando P (Proceder) para ejecutar la operación INT. Teclee varios caracteres y compare los resultados en el AX con el listado del apéndice F.

## SELECCIÓN DE UN MENÚ

El programa parcial de la figura 11-1 ilustra el despliegue de un menú y permite al usuario presionar las teclas direccionales (hacia arriba y hacia abajo) para seleccionar un elemento de él. El menú está definido en el segmento de datos dentro de una caja con dobles líneas (como se explicó en el capítulo 10). Los procedimientos y las acciones que realizan son los siguientes:

```
          page    60,132
TITLE     P11SELMU  (EXE)  Selección de una opción del menú

          .MODEL  SMALL
          .STACK  64

          .DATA
TOPROW    EQU     00                  Hilera superior del menú
BOTROW    EQU     07                  Hilera inferior del menú
LEFCOL    EQU     16                  Columna izquierda del menú
COL       DB      00                  Columna de pantalla
ROW       DB      00                  Hilera de pantalla
COUNT     DB      ?                   Caracteres por línea
LINES     DB      7                   Líneas exhibidas
ATTRIB    DB      ?                   Atributo de pantalla
NINTEEN   DB      1                   Ancho del menú
MENÚ      DB      0C9H,  17 DUP(0CDH),  0BBH
          DB      0BAH,  '  Add records      ',  0BAH
          DB      0BAH,  '  Delete records   ',  0BAH
          DB      0BAH,  '  Enter orders     ',  0BAH
          DB      0BAH,  '  Print report     ',  0BAH
          DB      0BAH,  '  Update accounts  ',  0BAH
          DB      0BAH,  '  View records     ',  0BAH
          DB      0C8H,  17 DUP(OCDH),  0BCH

PROMPT    DB      OS,  'To select an item,  use up/down arrow'
          DB      '  and press Enter.
          DB      13,  10,  09,  'Press Esc  to  exit.'

          .CODE
BEGIN     PROC    FAR
          MOV     AX,@data            ;Iniciar registros
          MOV     DS,AX                    de  segmento
          MOV     ES,AX
          CALL    Q10CLR              Despejar  pantalla
          MOV     ROW,BOTROW+2
          MOV     COL,00
          CALL    Q20CURS             Fijar  cursor
          MOV     AH,40H              •Petición de exhibición
          MOV     BX, 01              •Manejo de pantalla
          MOV     CX, 75              •Número de caracteres
          LEA     DX,PROMPT           ;Indicación
          INT     21H
A10LOOP:
          CALL    B10MENU             /Exhibición de menú
          MOV     COL,LEFCOL+l
          CALL    Q20CURS              Fijar  cursor
          MOV     ROW,TOPROW+l         Fijar hilera a opción superior
          MOV     ATTRIB,16H           Fijar video inverso
          CALL    H10DISP              Resaltar la línea de menú
          CALL    D10INPT              Proporcionar para la selección de menú
          CMP     AL, ODH              ¿Enter  presionado?
          JE      Al0LOOP                   sí,  continuar
          MOV     AX,0600H             Esc presionado  (indica fin)
          CALL    Q10CLR               Despejar  pantalla
          MOV     AX,4C00H             Salida a DOS
          INT     21H
BEGIN     ENDP
          Mostrar todo el menú:

B10MENU   PROC    NEAR
          MOV     ROW, TOPROW         ,-Fijar hilera superior
          MOV     LINES,08            /Número de líneas
```

Figura **11-1** Selección de un elemento desde el menú

**Selección de un menú**

```
                LEA     SI,MENÚ
                MOV     ATTRIB,71H      •Azul sobre blanco
B20 :
                MOV     COL,LEFCOL      •Fijar columna izquierda del menú
                MOV     COUNT,19
B30 :
                CALL    Q20CURS         |Fijar cursor en la siguiente columna
                MOV     AH,09H          |Petición de exhibición
                MOV     AL, [SI]        |Obtener carácter del menú
                MOV     BH, 00          |Página 0
                MOV     BL,71H          |Nuevo atributo
                MOV     CX, 01          |Un carácter
                INT     10H
                INC     COL             |Siguiente columna
                INC     SI              |Fijar siguiente carácter
                DEC     COUNT           |¿Último carácter?
                JNZ     B30             |No, repetir
                INC     ROW             |Siguiente hilera
                DEC     LINES
                JNZ     B20             ¿Se imprimieron todas las líneas?
                RET                     ;Si es así, regresar
B10MENU         ENDP
[                       Aceptar entrada a pedido
DIOINPT         PROC    NEAR
                MOV     AH.10H          |Petición de entrada
                INT     16H             |     del teclado
                CMP     AH,50H          |¿Flecha hacia abajo?
                JE      D2 0
                CMP     AH,48H          ¿Flecha hacia arriba?
                JE      D3 0
                CMP     AL,ODH          ¿Tecla Enter?
                JE      D90
                CMP     AL,1BH          ¿Tecla escape?
                JE      D90
                JMP     DIOINPT         |Ninguna, procesar de nuevo
D20 :           MOV     ATTRIB, 71H     |Azul sobre blanco
                CALL    HIODISP         |Fijar la línea anterior a video normal
                INC     ROW
                CMP     ROW,BOTROW-l    •¿Se pasó la hilera interior?
                JBE     D4 0            • no, muy bien
                MOV     ROW,TOPROW+l    • sí, restablecer
                JMP     D4 0
D30 :           MOV     ATTRIB,71H      ;Video normal
                CALL    HIODISP         Fijar línea anterior a video normal
                DEC     ROW
                CMP     ROW,TOPROW+l    ¿Abajo de la hilera superior?
                JAE     D4 0             no, muy bien
                MOV     ROW,BOTROW-l     sí, restablecer
D40 :           CALL    Q2 0CURS        Fijar cursor
                MOV     ATTRIB,16H      Video inverso
                CALL    HIODISP         Fijar nueva línea a video inverso
                JMP     DIOINPT
D90:            RET
DIOINPT         ENDP
<                       Fijar línea de menú a normal/resaltada
HIODISP         PROC    NEAR
                MOV     AH, 00
                MOV     AL,ROW          |La hilera dice qué línea fijar
                MUL     NINTEEN         |Multiplica por la longitud de la línea
                LEA     SI,MENU+1       |     por la línea de menú seleccionada
                ADD     SI,AX           |
                MOV     COUNT, 17       ,-Caracteres a exhibir
```

Figura **11-1** (continuación)

```
H20:
          CALL    Q2 0CURS          ,-Fijar cursor en segmento columna
          MOV     AH,09H            ;Petición de exhibición
          MOV     AL, [SI]          /Obtener carácter del menú
          MOV     BH, 00            ;Página 0
          MOV     BL,ATTRIB         ;Nuevo atributo
          MOV     CX, 01            ;Un carácter
          INT     10H
          INC     COL               ;Siguiente columna
          INC     SI                ;Fijar para el siguiente carácter
          DEC     COUNT             ;¿Último carácter?
          JNZ     H20               ;No, repetir
          MOV     COL,LEFCOL+l      /Restablecer columna a la izquierda
          CALL    Q20CTJRS          ;Fijar cursor
          RET
H10DISP   ENDP

                  Despejar pa

Q10CLR    PROC    NEAR
          MOV     AX,0600H
          MOV     BH,61H            ;Azul sobre café
          MOV     CX,0000
          MOV     DX,184FH
          INT     10H               ;Llamar a BIOS
          RET
Q10CLR    ENDP

                Fijar cursor hilera:columna

Q20CURS   PROC    NEAR
          MOV     AH,02H
          MOV     BH, 00            Página 0
          MOV     DH,ROW            Hilera
          MOV     DL,COL            Columna
          INT     10H
          RET
Q2 0CURS  ENDP
          END     BEGIN
```

Figura **11-1 (continuación)**

- BEGIN llama a Q10CLR para limpiar la pantalla, llama a B10MENU para desplegar lo: elementos del menú y establecer el primer elemento en video inverso y llama a D10INP1 para aceptar entradas del teclado.
- B10MENU muestra el conjunto completo de selecciones del menú.
- D10INPT utiliza la INT 16H para entrada: La flecha hacia abajo para bajar por el menú, 1; flecha hacia arriba para subir por el menú. Enter para aceptar un elemento del menú y Esi para salir. Las demás entradas del teclado son ignoradas. La rutina da vuelta alrededor de cursor, de manera que tratar de mover el cursor por arriba de la primera línea del menú k coloca en la última línea y viceversa. La rutina también llama a H10DISP para restaurar 1¡ línea anterior del menú a video normal y la nueva línea del menú (seleccionada) a vide( inverso.
- H10DISP muestra la línea actualmente seleccionada de acuerdo con un atributo (normal c en video inverso) que haya sido proporcionado.
- Q10CLR limpia toda la pantalla y la establece en primer plano azul y fondo café.

El programa ilustra la selección de menú en una forma sencilla; un programa complete ejecutaría una rutina para cada elemento seleccionado. Entenderá mejor este programa tecleando lo y verificándolo.

## INTERRUPCIÓN 09H Y EL BÚFER DEL TECLADO

Cuando presiona una tecla, el procesador del teclado genera el código de rastreo de la tecla y solicita la INT 09H. Esta interrupción (en la posición 36 de la tabla de servicios de interrupción) apunta a una rutina de manejo de interrupción en el BIOS de ROM. La rutina emite una petición de entrada desde el puerto 96 (60H):

```
IN  AL.60H
```

La rutina de BIOS lee el código de rastreo y lo compara con entradas en una tabla de códigos de rastreo para el carácter ASCII asociado (si existe). La rutina combina el código de rastreo con su carácter ASCII asociado y envía los dos bytes al búfer del teclado. La figura 11-2 ilustra este procedimiento.

Observe que la INT 09H maneja los bytes de estado del teclado en 40:17H, 40:18H y 40:96H para Shift, Alt y Ctrl, respectivamente. Sin embargo, aunque la presión de estas teclas genera la INT 09H, la rutina de interrupción establece los bits apropiados en los bytes de estado, pero no envía ningún carácter al búfer del teclado. También, la INT 09H ignora combinaciones de tecleo no definidas.

Cuando se *presiona* una tecla, el procesador del teclado de manera automática genera un código de rastreo y la INT 09H. Cuando se *suelta* o *libera* la tecla en un período de medio segundo, genera un segundo código de rastreo [el valor del primer código sumado a 128 (1000 0000B), lo que pone en uno el bit de la extrema izquierda] y emite otra INT 09H. El segundo código de rastreo indica a la rutina de interrupción que ha liberado la tecla. Si mantiene oprimida la tecla por más de medio segundo, el proceso de teclado se convierte en tecleo automático, y repite de manera automática la operación de la tecla.

### El búfer del teclado

El búfer del teclado necesita una dirección para indicar a la INT 09H en dónde insertar el siguiente carácter y otra dirección para indicar a la INT 16H de dónde extraer el carácter siguiente. Las dos direcciones tienen desplazamientos dentro del segmento 40[0]H. Lo siguiente describe el contenido del búfer:

| DIRECCIÓN | EXPLICACIÓN |
|---|---|
| 41AH | Dirección del inicio actual del búfer, la posición siguiente para la INT 16H para leer. |
| 41CH | Dirección del final actual del búfer, la posición siguiente para la INT 09H para almacenar un carácter ingresado. |
| 41EH | Dirección del inicio del búfer del teclado: 16 palabras (32 bytes), aunque puede ser más largo. El búfer retiene los caracteres del teclado y los códigos de rastreo como son introducidos para lectura posterior por medio de la INT 16H. Se necesitan dos bytes para cada carácter y su código de rastreo asociado: |

| Dirección de la parte inicial | Dirección de la parte final | Dirección del búfer |
|---|---|---|
| 41A | 41C | 41E ... |

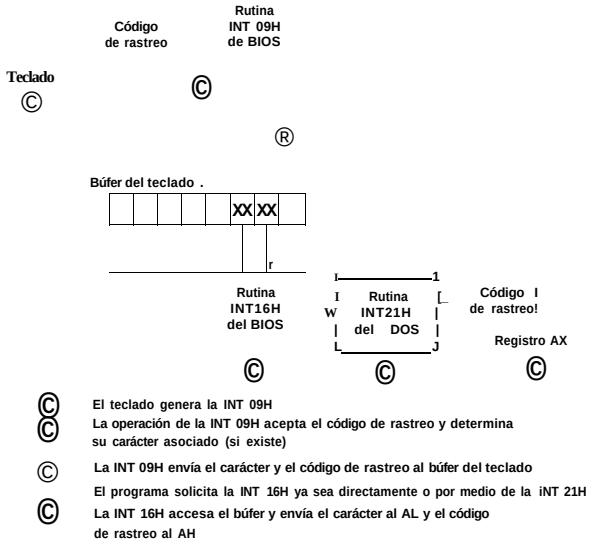



Figura 11-2 Búfer del teclado

Cuando se teclea un carácter, la INT 09H avanza la parte final. Cuando la INT 16H lee un carácter, avanza la parte inicial. De esta manera, el proceso es circular, con la parte inicial siguiendo de manera continua a la parte final.

Cuando el búfer está *vacío,* la parte inicial y la parte final están en la misma dirección. En el ejemplo siguiente, un usuario tecleó 'abcd < Enter > '. La INT 09H ha almacenado los caracteres en el búfer y ha avanzado la parte final a 428H. (Por simplicidad, el ejemplo no muestra los códigos de rastreo asociados.) El programa ha emitido la INT 16H cinco veces para leer todos los caracteres y ha avanzado la parte final a 428H, de manera que el búfer está vacío ahora:

```
.   a    b    c    d    <0DH>  . . .

    I    I    I    I    I      I
  41E  420  422  424  426    428
```

Cuando el búfer está *lleno,* la parte final está inmediatamente atrás de la parte inicial. Para verlo suponga que ahora teclea 'fghijklmnopqrs'. Entonces la INT 09H almacena los caracteres empezando en la parte final en 428H y dando vuelta para almacenar la 's' en 424H, inmediatamente antes de la parte inicial en 426H.

```
p   q   r   s <0DH>  e   f   g   h   i   j   k   l     m   n   o

I   I   I   I   I     I   I   I   I   I   I   I   I     I   I   I
41E 420 422 424 426   428 42A 42C 42E 430 432 434 436   438 43A 43C
```

En este punto, la INT 09H no acepta ningún carácter más que se teclee por adelantado, y aunque el búfer tiene 16, acepta sólo 15 a lo más. (¿Puede decir por qué?) Si la INT 09H captara

otro carácter, avanzaría la parte final a la misma dirección de la parte inicial y la INT 16H supondría que el búfer está vacío.

### Las teclas Ctrl, Shift y Alt

La INT 09H también maneja el byte de estado del shift en 40:17H en el área de datos del BIOS [Shift derecho (bit 0), shift izquierdo (bit 1), Ctrl (bit 2) y Alt (bit 3)], así como 40:18 y 40:96 para el teclado ampliado. Cuando presiona una de estas teclas, la rutina del BIOS pone en uno el bit apropiado, y cuando libera la tecla pone en cero el bit.

Su programa puede examinar si alguna de las teclas anteriores están presionadas ya sea por medio de la INT 16H (función 02H) o por referencia directa a la byte de estado. El siguiente programa parcial .COM ilustra el uso directo de la referencia directa al byte de estado:

```
BIODATA   SEGMENT AT 40H          Posiciona el área de datos del BIOS
          ORG 17H                  y
KBSTATE   DB ?                     el byte de estado
BIODATA   ENDS
CODESG    SEGMENT  PARA
          ASSUME   CS:CODESG,  DS:BIODATA
          ORG      100H
BEGIN:
          MOV      AX,BIODATA      Inicializa la dirección de
          MOV      DS, AX           BIODATA en DS
          MOV      AL,KBSTATE      Obtiene el byte de estado del teclado
          TEST     AL,00000011B    Prueba si algún shift se presionó
          JNZ      xxx              sí, entonces-saltar
```

El programa utiliza la característica SEGMENT AT para definir el área de datos del BIOS como, en realidad, un segmento ficticio. KBSTATE identifica la posición del byte de estado del teclado en 40:17H. El segmento de código inicializa la dirección de BIODATA en el DS y almacena el byte de estado del teclado en el AL. Una operación OR prueba si alguna de las teclas shift fue presionada.

Puede modificar este código para examinar también los bytes de estado de teclado ampliado en 40:18H y 40:96H.

## CÓMO INGRESAR EL CONJUNTO COMPLETO DE CARACTERES ASCII

El conjunto completo ASCII consta de 256 caracteres numerados desde el 0 hasta el 255 (FFH). Muchos de éstos son caracteres estándar desplegables, desde el ASCII 20H (espacio) hasta el ASCII 7EH (el carácter de tilde, —). Como el teclado está limitado a 83 o 101 teclas, la mayoría

de los 256 caracteres ASCII no están representados en él. Sin embargo puede introducir cualquiera de los códigos desde 01 hasta 255 manteniendo oprimida la tecla Alt e ingresando el código apropiado como un valor decimal por medio del panel numérico. El sistema almacena los valores que ingresó como dos bytes en el búfer del teclado: el primero es el carácter ASCII generado y el segundo, es cero. Por ejemplo, Alt+001 envía 01H, y Alt+255 envía FFH. Puede utilizar DEBUG para examinar el efecto de introducir diferentes números:

```
100  MOV  AH.10
102  INT  16
104  JMP  100
```

## PUNTOS CLAVE

- Los bytes de estado del shift en el área de datos del BIOS indican el estado actual de Ctrl, Alt, Shift, BloqMayús, BloqNum y ScrollLock.
- Las operaciones de la INT 21H del DOS proporcionan diferentes servicios con o sin repetición en la pantalla, para reconocer o ignorar Ctrl+Break y para aceptar códigos de rastreo.
- La INT 16H del BIOS proporciona la operación básica del BIOS para el teclado para aceptar caracteres desde el búfer del teclado. Para una tecla de carácter, la operación envía el carácter al AL y el código de rastreo de la tecla al AH. Para una tecla de función extendida, la operación envía cero al AL y el código de rastreo al AH.
- El código de rastreo es un número único asignado a cada tecla, que le permite al sistema identificar el origen de una tecla presionada y permite a un programa verificar las teclas de función extendidas tales como Inicio, AvPág y las flechas.
- El área de datos del BIOS en 40:1EH contiene el búfer del teclado. Esta área le permite teclear hasta 15 caracteres antes que el programa solicite una entrada.
- Cuando presiona una tecla, el procesador del teclado genera el código de rastreo de la tecla (su único número asignado) y solicita la INT 09H. Cuando suelta la tecla genera un segundo código de rastreo (el primero más 128: pone en uno el bit de la extrema izquierda) para indicarle a la INT 09H que la tecla ha sido soltada.
- La INT 09H del BIOS obtiene un código de rastreo del teclado, y o bien genera un carácter ASCII asociado y envía el código de rastreo al área del búfer del teclado, o establece el estado de Ctrl, Alt, Shift.

## PREGUNTAS

**11-1.** (a) ¿Cuál es la localidad en el área de datos del BIOS, del primer byte del estado del shift del teclado': (b) ¿Qué significa el contenido 00001100? (c) ¿Qué significa el contenido 00000010?

**11-2.** Explique las características de las funciones siguientes para entrada desde el teclado con la INT 21H (a) 01H; (b) 07H; (c) 08H; (d) OAH.

**11-3.** Explique las diferencias entre las funciones OOH, 01H y 10H de la INT 16H.

**11-4.** Proporcione los códigos de rastreo para las funciones extendidas siguientes: (a) Flecha hacia arriba; (b) tecla de función programable; (c) inicio (Home); (d) RePág (PgUp).

**11-5.** Utilice DEBUG para examinar los efectos de los tecleos introducidos. Para solicitar entrada de una instrucción en lenguaje ensamblador, teclee A 100 e introduzca las instrucciones siguientes:

```
MOV   AH, 00  (o AH, 10)
INT   16
JMP   100
```

Utilice U 100,104 para desensamblar el programa, y utilice el comando P para hacer que DEBUG ejecute toda la interrupción. La ejecución se detiene en espera de su entrada. Presione cualquier tecla para examinar los registros AH y AL. Continúe introduciendo diferentes teclas. Presione Q para salir de DEBUG.

**11-6.** Codifique las instrucciones para introducir un solo tecleo; si la tecla es AvPág(PgDn), coloque el cursor en el renglón 24, columna 0.

**11-7.** Corrija el programa de la figura 11-1 para proporcionar las características siguientes: (a) Después del borrado inicial de la pantalla, mostrar una petición que pida al usuario presionar Fl para un menú de pantalla, (b) Cuando se presione F1, desplegar el menú, (c) También permitir a los usuarios seleccionar elementos del menú presionando el primer carácter (mayúscula o minúscula) de cada elemento, (d) **A** solicitud de un elemento, mostrar un mensaje para esa selección en particular, como "Procedimiento para eliminar registros", (e) Permitir a los usuarios presionar la tecla Esc para regresar al menú principal de la rutina seleccionada.

**11-8.** ¿Bajo qué circunstancias ocurre una INT 09H?

**11-9.** Explique en términos sencillos cómo la INT 09H maneja las teclas Ctrl y Shift de manera diferente a la forma de manejar las teclas del teclado estándar.

**11-10.** (a) ¿En dónde está la posición en memoria del BIOS del búfer del teclado? (b) En bytes, ¿cuál es el tamaño del búfer? (c) ¿Cuántos caracteres de teclado puede tener?

**11-11.** ¿Qué significa que la dirección de la cabeza y de la cola en el búfer del teclado sean iguales? (b) ¿Qué significa que la dirección de la cola siga de manera inmediata de la cabeza?

# PARTE D — Manipulación de datos

## CAPÍTULO 12

# Operaciones con cadenas de caracteres

OBJETIVO

Explicar las instrucciones especiales utilizadas para procesar datos de cadenas de caracteres.

## INTRODUCCIÓN

En este punto, las instrucciones presentadas han manejado datos definidos como un solo byte, palabra o palabra doble. Sin embargo, a veces es necesario mover o comparar campos de datos que excedan estas longitudes. Por ejemplo, puede querer comparar las descripciones o nombres a fin de clasificarlas en orden ascendente. Los elementos en este formato son conocidos como *datos de cadena de caracteres* (o sólo *datos de cadena)* y puede ser de carácter o numérico. Para procesar una cadena de caracteres, el lenguaje ensamblador proporciona cinco instrucciones para cadenas:

| | |
|---|---|
| MOVS | Mueve un byte, palabra o palabra doble desde una localidad en memoria a otra. |
| LODS | Carga desde memoria un byte en el AL, una palabra en el AX o una palabra doble en el EAX. |
| STOS | Almacena el contenido de los registros AL, AX o EAX en memoria. |
| CMPS | Compara localidades de memoria de un byte, palabra o palabra doble. |
| SCAS | Compara el contenido de AL, AX o EAX con el contenido de una localidad de memoria. |

Una instrucción asociada, el prefijo REP, provoca que una instrucción para cadena se realice de manera repetitiva un número específico de veces.

## CARACTERÍSTICAS DE LAS OPERACIONES CON CADENAS DE CARACTERES

Una instrucción de cadena puede especificar el procesamiento repetitivo de un byte, palabra o (en el 80386 y procesadores posteriores) palabra doble a un tiempo. Así, puede seleccionar una operación de byte para una cadena con un número impar de bytes y una operación de palabra para una cadena con un número par de bytes. Cada instrucción de cadena tiene una versión para byte, palabra o palabra doble y supone el uso de los registros ES:DI o DS:SI. El DI y SI deben contener direcciones de desplazamiento válidas.

Básicamente existen dos maneras de codificar instrucciones de cadena. En la tabla siguiente, la segunda columna muestra el formato básico para cada operación, la cual utiliza los operandos implicados listados en la tercer columna (por ejemplo, si codifica una instrucción MOVS, incluya operandos como MOVS BYTE1 ,BYTE2, en donde la definición de los operandos indican la longitud del movimiento):

| Operación | Instrucción básica | Operandos implicados | Operación con bytes | Operación con palabra | Operación con palabra doble |
|---|---|---|---|---|---|
| Mover | MOVS | ES:DI,DS:SI | MOVSB | MOVSW | MOVSD |
| Cargar | LODS | AX,DS:SI | LODSB | LODSW | LODSD |
| Almacenar | STOS | ES:DI,AX | STOSB | STOSW | STOSD |
| Comparar | CMPS | DS:SI,ES:DI | CMPSB | CMPSW | CMPSD |
| Rastrear | SCAS | ES:DI,AX | SCASB | SCASW | SCASD |

La segunda manera de codificar instrucciones de cadena es la práctica usual, como se mostró en las columnas cuarta, quinta y sexta. Usted carga las direcciones de los operandos en los registros DI y SI y codifica, por ejemplo, MOVSB, MOVSW y MOVSD sin operandos.

Las instrucciones de cadena suponen que el DI y el SI contienen direcciones de desplazamiento válidas que hacen referencia a bytes en memoria. El registro SI está asociado por lo común con el DS (segmento de datos) como DS:SI. El registro DI siempre está asociado con el registro ES (segmento extra) como ES:DI. En consecuencia, MOVS, STOS, CMPS y SCAS necesitan que un programa .EXE inicialice el registro ES en general pero no necesariamente, con la misma dirección que la del registro DS:

```
MOV   AX,@data      ;Obtiene la dirección del segmento de datos
MOV   DS,AX         ;Lo almacena en DS
MOV   ES,AX         ; y en ES
```

## REP: PREFIJO DE REPETICIÓN DE CADENA

El prefijo REP inmediatamente antes de una instrucción de cadena, como REP MOVSB, proporciona una ejecución repetida con base en un contador inicial que usted establece en el registro CX. REP ejecuta la instrucción de cadena, disminuye el CX y repite la operación hasta que el contador en el CX sea cero. De esta manera, puede manejar cadenas de caracteres de casi cualquier longitud.

La bandera de dirección (DF) determina la dirección de la operación que se repite:

- Para procesamiento de izquierda a derecha (la manera normal de procesar), utilice CLD para poner en cero a DF.
- Para procesamiento de derecha a izquierda, utilice STD para poner uno en DF.

El ejemplo siguiente mueve (o mejor, copia) los 20 bytes de STRING1 a STRING2 (suponga que el DS y ES ambos han sido inicializados con la dirección del segmento de datos, como ya se mostró):

```
STRINGl   DB    20 DUP('*')
STRING2   DB    20 DUP(' ')

          CLD               ;Pone en cero la bandera de dirección
          MOV   CX,20       ;Inicializa para 20 bytes
          LEA   DI,STRING2  ,-Inicializa el nombre receptor
          LEA   SI,STRINGl  ;Inicializa la dirección emisora
          REP   MOVSB       ;Copia STRINGl en STRING2
```

Durante la ejecución, las instrucciones CMPS y SCAS también establecen las banderas de estado, de modo que la operación puede terminar de manera inmediata al encontrar una condición especificada. Las variaciones de REP para este propósito son las siguientes:

- REP Repite la operación hasta que el CX llegue a cero.
- REPE o REPZ Repite la operación mientras la bandera de cero (ZF) indique igual o cero. Se detiene cuando la ZF indica diferente o cero o cuando CX llega a cero.
- REPNE o REPNZ Repite la operación mientras la ZF indica diferente o cero. Se detiene cuando la ZF indica igual o cero o cuando CX llega a cero.

Para el 80286 y procesadores más avanzados, el uso de las operaciones con palabra o palabra doble puede proporcionar un procesamiento más rápido. Ahora examinaremos en detalle las operaciones de cadena.

## MOVS: MOVER UNA CADENA DE CARACTERES

MOVS combinada con un prefijo REP y una longitud en el CX puede mover cualquier número de caracteres. Aunque usted no codifica los operandos, la instrucción se parece a esto:

```
[etiqueta:] REP MOVSn [ES:DI,DS:SI]
```

Para la cadena receptora, los registros segmento:desplazamiento son ES:DI; para la cadena emisora los registros segmento desplazamiento son DS:SI. Como resultado, al inicio de un programa .EXE inicialice el registro ES junto con el registro DS y, antes de ejecutar el MOVS, utilice LEA para inicializar los registros DI y SI. Dependiendo de la bandera de dirección, MOVS incrementa o disminuye los registros DI y SI en 1 para un byte, en 2 para una palabra y en 4 para una palabra doble. El código siguiente es ilustrativo:

```
        MOV   CX,número          ;Número de byte/palabras
        LEA   DI,STRING2         ;Dirección de STRING2
        LEA   SI,STRING1         ;Dirección de STRING1
        REP   MOVSn              ,-Mueve n bytes/palabras
```

Las instrucciones equivalentes para REP MOVSB son:

```
          JCXZ  LABEL2          ,-Salta, si CX es cero
LABEL1:   MOV   AL, [SI]        ,-Obtiene el carácter de STRING1
          MOV   [DI], AL        ,-Almacena el carácter en STRING2
          INC   DI              ,-0 DEC DI
          INC   SI              ,-0 DEC SI
          LOOP  LABEL1
LABEL2:
```

La figura 6-2 ilustró cómo mover un campo de 9 bytes. El programa también pudo haber utilizado MOVSB para este objetivo. En la figura 12-1 el procedimiento C10MVSB utiliza MOVSB para mover de byte en byte un campo de 10 bytes NAME1 a NAME2. La primer instrucción, CLD, pone en cero la bandera de dirección de modo que el MOVSB procesa los datos de izquierda a derecha. Al inicio de la ejecución, por lo regular la bandera de dirección se encuentra en cero, pero aquí por precaución está codificado CLD.

Las dos instrucciones LEA cargan los registros SI y DI con los desplazamientos de NAME1 y NAME2, respectivamente. Ya que el cargador del DOS para un programa .COM de manera automática inicializa los registros DS y ES, las direcciones segmento:desplazamiento son correctas para ES:DI y DS:SI. Una instrucción MOV inicializa el CX con 10 (la longitud de NAME1 y de NAME2). Ahora la instrucción REP MOVSB realiza lo siguiente:

- Mueve el byte de la extrema izquierda de NAME1 (direccionado por DS:SI) al byte de extrema izquierda de NAME2 (direccionado por ES:DI).
- Incrementa el DI y SI en uno para los siguientes bytes a la derecha.
- Disminuye el CX en 1.
- Repite esta operación, 10 ciclos en total, hasta que el CX se convierte en cero.

Puesto que la bandera de dirección es cero y MOVSB incrementa DI y SI, cada iteración procesa un byte más a la derecha, como NAME1 +1 a NAME2+1, y así en forma sucesiva. Al final de la ejecución, el CX contiene 00, el DI contiene la dirección de NAME2+10, y el SI contiene la dirección de NAME1 +10 —ambos un byte después del final del nombre.

Si la bandera de dirección es uno, MOVSB disminuiría DI y SI, provocando que el procesamiento ocurriera de derecha a izquierda. Pero en ese caso, para mover los contenidos de manera adecuada tendría que inicializar el SI con NAME1 +9 y el DI con NAME2+9.

El procedimiento siguiente de la figura 12-1, D10MVSW, utiliza MOVSW para mover cinco palabras desde NAME2 a NAME3. Al final de la ejecución, el CX contiene 00, el DI contiene la dirección de NAME3 + 10, y el SI contiene la dirección de NAME2+10.

```
TITLE   P12MOVST (COM)   Operaciones de cadenas con MOVS
        .MODEL SMALL
        .CODE
        ORG     100H
BEGIN:  JMP     SHORT MAIN

NAME1   DB      'Assemblers'        ,-Elementos de datos
ÑAME2   DB      10 DUP(' ')
ÑAME3   DB      10 DUP(' ')

MAIN    PROC    NEAR                Procedimiento principal
        CALL    C10MVSB             Subrutina MVSB
        CALL    D10MVSW             Subrutina MVSW
        MOV     AX,4C00H            Salir a DOS
        INT     21H
MAIN    ENDP
;               Use of MOVSB:
C10MVSB PROC    NEAR
        CLD                         Izquierda a derecha
        MOV     CX,10               Mover los bytes
        LEA     DI,ÑAME2              de NAME1 a ÑAME2
        LEA     SI,NAME1
        REP MOVSB
        RET
C10MVSB ENDP
;                Use of MOVSW:
D10MVSW PROC    NEAR
        CLD                         /Izquierda a derecha
        MOV     CX,05               /Mover 5 palabras
        LEA     DI,NAME3            /  de NAME2 a ÑAME3
        LEA     SI,NAME2
        REP MOVSW
        RET
D10MVSW ENDP
        END     BEGIN
```

Figura 12-1 Uso de operaciones con cadena MOVS

Ya que MOVSW incrementa los registros DI y SI en 2, la operación sólo necesita de cinco ciclos. Para procesar de derecha a izquierda, inicialice el SI con NAME1+8 y el DI con NAME2+8.

## LODS: CARGA UNA CADENA DE CARACTERES

LODS carga el AL con un byte, el AX con una palabra o el EAX con una palabra doble desde la memoria. La dirección de memoria está sujeta a los registros DS:SI, aunque puede pasar por alto el SI. Dependiendo de la bandera de dirección, la operación también incrementa o disminuye el SI en 1 para byte, en 2 para palabra y en 4 para palabra doble.

Ya que una operación LODS llena el registro, no existe razón práctica para utilizar con ella el prefijo REP. Para la mayor parte de los propósitos, una sencilla instrucción MOV es adecuada. Pero MOV genera 3 bytes de código de máquina, mientras que LODS sólo genera uno, aunque necesita que inicialice el registro SI. Podría utilizar LODS para recorrer una cadena un byte, una palabra o una palabra doble a la vez, examinándola de forma sucesiva contra un valor particular.

```
TITLE   P12LODST (COM)  Uso de LODSB en operaciones de cadenas
        .MODEL  SMALL
        .CODE
        ORG     100H
BEGIN:  JMP     SHORT MAIN

FIELDA  DB      'Assemblers'        Elementos de datos
FIELDB  DB      10 DUP(20H)

MAIN    PROC    NEAR                Procedimiento principal
        CLD                         Izquierda a derecha
        MOV     CX,10
        LEA     SI,FIELDA           Cargar dirección de FIELDA
        LEA     DI,FIELDB+9         Cargar dirección de FIELDB+9
A20:    LODSB                       Obtener carácter en AL,
        MOV     [DI],AL               se almacena en FIELDB,
        DEC     DI                    izquierda a derecha
        LOOP    A20                 ¿10 caracteres?
        MOV     AX,4C00H              sí, salida
        INT     21H
MAIN    ENDP
        END     BEGIN
```

Figura 12-2 Uso de la operación de cadena de caracteres LODSW

Las instrucciones equivalentes a LODSB son:

```
MOV   AL, [SI]     ,-Carga un byte en AL
INC   SI           ;Incrementa SI al byte siguiente
```

En la figura 12-2 el área de datos define un campo de 10 bytes llamado FIELDA, con el valor "Assemblers" y otro campo de 10 bytes llamado FIELDB. El objetivo es transferir los bytes de FIELDA a FIELDB en secuencia inversa, de manera que FIELDB contenga "srelbmessA". LODSB es utilizada para accesar un byte a la vez de FIELDA al AL y la instrucción MOV [DI],AL transfiere los bytes a FIELDB de derecha a izquierda.

## STOS: ALMACENAR UNA CADENA DE CARACTERES

STOS almacena los contenidos del registro AL, AX o EAX en un byte, palabra o palabra doble en memoria. La dirección de memoria siempre está sujeta a los registros ES:DI. Dependiendo de la bandera de dirección, STOS también incrementa o disminuye el registro DI en 1 para byte, 2 para palabra y 4 para palabra doble.

Un uso práctico de STOS con un prefijo REP es para inicializar el área de datos a cualquier valor especificado, tal como limpiar el área de despliegue a blancos. Puede establecer el número de bytes, palabras o palabras dobles en el CX. Las instrucciones equivalentes a REP STOSB son:

```
        JCXZ     LABEL2      ;Si CX es cero, entonces salta
LABEL1: MOV      [DI],AL     /Almacena AL en memoria
        INC/DEC  DI          /Incrementa o disminuye
        LOOP     LABEL1
LABEL2
```

```
TITLE   P12STOST (COM)   Operaciones de cadenas con STOSW
        .MODEL   SMALL
        .CODE
        ORG     100H
BEGIN:  JMP     SHORT  MAIN

NAME1   DB      'Assemblers        /Elementos de datos

MAIN    PROC   NEAR                Procedimiento principal
        CLD                        Izquierda a derecha
        MOV     AX,2020H           Mover
        MOV     CX,05                5 blancos
        LEA     DI,NAME1             a NAME1
        REP  STOSW
        MOV     AX,4C00H           /Salir a DOS
        INT     21H
MAIN    ENDP
        END     BEGIN
```

Figura 12-3 Uso de la operación de cadena de caracteres STOSW

La instrucción STOSW en la figura 12-3 almacena de forma repetida una palabra con 2020H (blancos) cinco veces en NAME1. La operación almacena el AL en el primer byte y el AH en el byte siguiente (esto es, en orden inverso). Al final, NAME1 está en blanco, el CX contiene 00 y el DI contiene la dirección de NAME1 +10.

## CÓMO TRANSFERIR DATOS CON LODS Y STOS

El programa de la figura 12-4 ilustra el uso de ambas instrucciones, LODS y STOS. El ejemplo es semejante al del programa de la figura 10-4, que transfiere caracteres y atributos de manera directa al área de despliegue de video, excepto que en la figura 12-4 contiene estas diferencias:

- Para el área de video, utiliza la página número 02, en lugar de la 01.
- En C10PROC utiliza STOSW para almacenar caracteres y atributos asociados en el área de video, en lugar de esta instrucción y sus dos intrucciones DEC acompañantes que disminuyen el DI:

```
MOV WORD PTR [VIDAREA + DI*], AX
```

- En el segmento de datos, define un elemento llamado PROMPT, solicita al usuario "Presionar cualquier tecla para ser utilizada al final del procesamiento.
- Al terminar el procesamiento, el procedimiento D10PROMPT transfiere la indicación definida al área de despliegue de video. Para este fin, utiliza LODSB para accesar caracteres, uno a la vez, desde PROMPT al AL y utiliza STOSW para transferir cada carácter y su atributo asociado desde el AX al área de video.

## CMPS: COMPARAR CADENAS

CMPS compara el contenido de una localidad de memoria (direccionada por DS:SI) con el de otra localidad de memoria (direccionada por ES:DI). Dependiendo de la bandera de dirección, CMPS incrementa o disminuye también los registros SI y DI en 1 para bytes, en 2 para palabras y en 4

```
TITLE     P12DRVID (EXE)   Exhibición de video directo
          .MODEL SMALL

VIDSEG    SEGMENT AT 0BA00H    /Página 2 del área de video
VIDAREA   DB      1000H DUP(?)
VIDSEG    ENDS

          .DATA
PROMPT    DB      'Press any key...'

          .STACK 64

          .CODE
BEGIN     PROC    FAR
          MOV     AX,@data        Direccionamiento
          MOV     DS, AX            del segmento de datos,
          MOV     AX, VIDSEG        y del
          MOV     ES, AX            área de video
          ASSUME  ES/VIDSEG
          MOV     AH, OFH         Petición obtiene
          INT     10H               y guarda
          PUSH    AX                modo y
          PUSH    BX                página presente
          MOV     AH,00H          Petición fija
          MOV     AL, 03            modo 03,  despejar pantalla
          INT     10H
          MOV     AH,05H          Petición fija
          MOV     AL,02H            página 02
          INT     10H
          CALL    C10PROC         Procesar área de exhibición
          CALL    DIOPROMPT       Mostrar indicación al usuario
          CALL    E10INPT         Proporcionar para entrada
          MOV     AH, 05H         Restaurar
          POP     BX                número de página
          MOV     AL, BH            original
          INT     10H
          POP     AX              /Restaurar video
          MOV     AH,00H          /  modo (en AL)
          INT     10H
          MOV     AX,4C00H        /Salir a DOS
          INT     21H
BEGIN     ENDP
          Almacenar carácter y atributo en área de vi

C10PROC   PROC    NEAR
          MOV     AL.41H          Caráter a mostrar
          MOV     AH,01H          •Atributo
          MOV     DI,660          •Inicio de área de exhibición
C30:      MOV     CX, 60          •Caracteres por hilera
C40:      STOSW                   •AX en área de exhibición
          LOOP    C40             •Repetir 60 veces
          INC     AH              /Siguiente atributo
          INC     AL              /Siguiente carácter
          ADD     DI,40           /Sangrar para siguiente hilera
          CMP     AL,51H          /¿Último carácter a mostrar?
          JNE     C30               no, repetir
          RET                     /  sí, regresar
CIOPROC   ENDP
          Indicación a usuario para presionar tecla

DIOPROMPT PROC    NEAR
          MOV     CX, 16          /Caracteres a exhibir
          LEA     SI,PROMPT       /Dirección de la indicación
```

Figura 12-**4** Despliegue directo en video

```
            MOV     DI,3840         Ubicación en el área de exhibición
            MOV     AH,03H          Nuevo atributo en AH
D20:        LODSB                   Carácter en AL
            STOSW                   Almacenar en área de exhibición
            LOOP    D20             16 veces
            RET                     Regresar
D10PROMPT   ENDP
                    Aceptar entrada

E10INPT     PROC    NEAR
            MOV     AH,10H          ;Petición del teclado
            INT     16H             ,- entrada
            RET
E10INPT     ENDP
            END     BEGIN
```

Figura 12-4B (continuación)

para palabras dobles. La operación establece las banderas AF, CF, OF, PF, SF y ZF. Cuando s< combinan con un prefijo REP y una longitud en el CX, de manera sucesiva CMPS puede compa rar cadenas de cualquier longitud.

Pero observe que CMPS proporciona una comparación *alfanumérica,* esto es, una compa ración de acuerdo con los valores ASCII. La operación no es adecuada para comparacione algebraicas, que consisten en números con signo. Considere la comparación de dos cadenas qu contienen JEAN y JOAN. Una comparación de izquierda a derecha, tiene el resultado siguiente

J:J Iguales
E:0 Diferentes (E es menor)
A: A Iguales
N:N Iguales

Una comparación de los cuatro bytes termina con una comparación de N con N (iguales). Ahora ya que los dos nombres no son idénticos, la operación debe terminar tan pronto como la compa ración entre dos caracteres sea diferente. Para este propósito, REP tiene una variación, REP1 (Repite cuando sea igual), que repite la operación mientras la comparación entre caracteres se igual, o hasta que el registro CX sea igual a cero. El código para la comparación repetida de ui byte es REPE CMPSB.

La figura 12-5 consta de dos ejemplos que utilizan CMPSB. El primero compara ÑAME con NAME2, que contienen los mismos valores. Por tanto, la operación CMPSB se realiza coi los 10 bytes. Al final de la ejecución, el CX contiene 00, el DI contiene la dirección de NAME2 +10 el SI contiene la dirección de NAME1 +10, la bandera de signo es positiva y la bandera de cen indica igual o cero.

El segundo ejemplo compara NAME2 con NAME3, que contienen valores diferentes. L operación CMPSB termina después de comparar el primer byte resultando una condición alta **i** diferente: El CX contiene 09, el DI contiene la dirección de NAME3 + 1, el SI contiene 1 dirección de NAME2+1, la bandera del signo es positiva y la bandera del signo indica diferente

El primer ejemplo resulta igual o cero y (sólo por razones de ilustración) mueve 01 a registro BH. El segundo ejemplo resulta diferente y mueve 02 al registro BL. Si utiliza DEBU( para rastrear las instrucciones, al final de la ejecución verá 0102 en el registro BX.

*¡Advertencia!:* Estos ejemplos utilizan CMPSB para comparar datos de byte en byte. Inicialic CX en 5, si utiliza CMPSW para comparar datos una palabra a la vez. Pero éste no es el problema

```
TITLE   P12CMPST (COM)   Uso de CMPS para operaciones en cadenas
        .MODEL   SMALL
        .CODE
        ORG     100H
BEGIN:  JMP     SHORT  MAIN

NAME1   DB      'Assemblers'        ;Elementos de datos
ÑAME2   DB      'Assemblers'
ÑAME3   DB      10 DUP(' ')

MAIN    PROC    NEAR                /Procedimiento principal
        CLD                        /Izquierda a derecha
        MOV     CX, 10              Iniciar para 10 bytes
        LEA     DI,NAME2
        LEA     SI,NAME1
        RE PE CMPSB                •Compare NAME1:  NAME2
        JNE     G2 0                 no es igual, saltarlo
        MOV     BH, 01               igual,  fijar BH
G20 :
        MOV     CX,10               Iniciar para 10 bytes
        LEA     DI, NAME3
        LEA     SI,NAME2
        REPE  CMPSB                Compare NAME2:  ÑAME3
        JE      G3 0                 igual,  salir
        MOV     BL,02                no es igual,  fijar BL
G3 0 :
        MOV     AX,4C00H           Salir a DOS
        INT     21H
MAIN    ENDP
        END     BEGIN
```

Figura 12-5 Uso de las operaciones de cadena de caracteres CMPS

Cuando compara palabras, CMPSW invierte los bytes. Por ejemplo, compare los nombres SAMUEL y ARNOLD. Para la comparación inicial de las palabras, en lugar de comparar SA con AR la operación compara AS con RA. Así, en lugar de que el nombre SAMUEL indique un valor mayor, será menor, e incorrecto. CMPSW funciona de manera correcta sólo si las cadenas comparadas contienen datos numéricos sin signo definido como DW, DD o DQ.

## SCAS: BÚSQUEDA EN CADENAS

SCAS difiere de CMPS en que SCAS busca una cadena por un valor de byte, palabra o palabra doble específico. SCAS compara el contenido de la localidad de memoria (direccionado por ES:DI) con el contenido del registro AL, AX o EAX. Dependiendo de la bandera de dirección, SCAS también incrementa o disminuye el registro DI en 1 para byte, 2 para palabra y 4 para palabra doble. Al final de la ejecución, SCAS establece las banderas AF, CF, OF, PF, SF y ZF. Cuando se combina con el prefijo REP y una longitud en el CX, SCAS puede buscar en cadenas con cualquier longitud.

SCAS es útil en particular para aplicación de edición de texto, en la que el programa tiene que buscar signos de puntuación, como puntos, comas y blancos.

El código en la figura 12-6 rastrea NAME1 por la letra minúscula 'm\ La operación en este caso es REPNE SCASB, ya que la operación SCASB es para una búsqueda continua, mientras la comparación no sea igual o hasta que CX sea cero.

Como NAME1 contiene "Assemblers", SCASB encuentra una coincidencia en la quinta comparación. Si utiliza DEBUG para rastrear las instrucciones, al final de la ejecución de la

```
TITLE   P12SCAST (COM)  Operaciones de cadenas con SCAS
        .MODEL  SMALL
        .CODE
        ORG     100H
BEGIN:  JMP     SHORT MAIN
-------------------------------------------------------
NAME1   DB      'Assemblers'        .•Elementos de datos
-------------------------------------------------------
MAIN    PROC    NEAR .              /Procedimiento principal
        CLD                         /Izquierda a derecha
        MOV     AL,'m'
        MOV     CX, 10              /Escudriñar 'm'
        LEA     DI, NAME1           /  en NAME1
        REPNE SCASB
        JNE     H20                 /Si se encontró
        MOV     AL, 03              /  almacenar 03 en AL
H20:
        MOV     AH,4CH
        INT     21H                 /Salir a DOS
MAIN    ENDP
        END     BEGIN
```

Figura 12-6 Uso de la operación de cadena de caracteres SCASB

operación REP SCASB verá que la bandera del cero muestra cero, el CX está disminuido en 05 y el DI está aumentado en 05. (El DI está incrementado en un byte pasando la posición actual de la 'm'.)

El programa almacena 03 en el registro AL (por razones ilustrativas) para indicar que se encontró una "m".

SCASW busca una palabra en memoria que coincida con la palabra en el registro AX. Si utiliza LODSW o MOV para transferir una palabra al registro AX, el primer byte estaría en el AL y el segundo en el AH. Como SCASW compara los bytes en orden inverso, la comparación funciona de manera correcta.

## BUSCAR Y REEMPLAZAR

También puede necesitar reemplazar un carácter específico con otro carácter, por ejemplo, para borrar de un documento caracteres de edición, como símbolos de párrafo y de fin de página. El siguiente programa parcial busca en STRING un ampersán (&) y lo reemplaza con un blanco. Si SCASB localiza un ampersán, termina la operación. En este ejemplo, existe uno en STRING+8, er donde se inserta un blanco, aunque al final SCASB haya incrementado el registro DI a STRING+9. Disminuir el DI en uno proporciona la dirección correcta para insertar el blanco que reemplaza al carácter. El código es el siguiente:

```
STRLEN  EQU 15                  /Longitud de STRING
STRING  DB      'The time&is now'


        CLD                     /De izquierda a derecha
        MOV                     AL,/Busca el carácter
```

```
MOV   CX,STRLEN          ;Longitud de STRING
LEA   DI,STRING          ,-Dirección de STRING
REPNE SCASB              /Busca
JNZ K2 0                 /¿Se encontró el carácter?
DEC   DI                 / sí, ajusta dirección
MOV   BYTE PTR[DI],20H   /Reemplace con un blanco
```

## CODIFICACIÓN ALTERNA PARA INSTRUCCIONES DE CADENA DE CARACTERES

Como vimos, si codifica de manera explícita con una instrucción para byte, palabra o palabra doble, como MOVSB, MOVSW o MOVSD, el ensamblador supone la longitud correcta y no necesita operandos. También puede utilizar los formatos básicos de la instrucción para las operaciones con cadenas de caracteres. Para instrucciones tales como MOVS, que no tienen sufijo para indicar byte, palabra o palabra doble, debe indicar la longitud de los operandos. Por ejemplo, si FLDA y FLDB están definidas como byte (DB), la instrucción

```
REP MOVS FLDA,FLDB
```

implica un movimiento repetido del byte que inicia en FLDB al byte que inicia en FLDA. Si carga los registros DI y SI con las direcciones de FLDA y FLDB, también puede codificar la instrucción MOVS como

```
REP MOVS ES:BYTE PTR[DI],DS:[SI]
```

Pocos programas están codificados de esta manera, y el formato se trata aquí sólo para información.

## CÓMO DUPLICAR UN PATRÓN

La instrucción STOS es útil para codificar un área de acuerdo con un valor de byte, palabra o palabra doble específico. Sin embargo, para repetir un patrón que exceda estas longitudes puede utilizar MOVS con una pequeña modificación. Digamos que tiene que establecer una línea de despliegue al siguiente patrón:

```
***###***#######***#######***###...
```

En lugar de definir el patrón de manera repetitiva, sólo necesita definir los primeros seis bytes que están al inicio de la línea de despliegue. Aquí está la codificación necesaria:

```
PATTERN  DB   '***###'
DISAREA  DB   42 DUP(?)
```

```
CLD                     ,-De izquierda a derecha
MOV   CX,21             ;21 palabras
LEA   DI,DISAREA        /Destino
LEA   SI,PATTERN        /Origen
REP   MOVSW             /Mueve los caracteres
```

En la ejecución, MOVSW mueve la primer palabra de PATTERN (**) a la primer palabra de DISAREA y después mueve la segunda (*#) y tercer (##) palabras:

```
                ***###***###
                 I     I
           PATTERN     DISAREA
```

En este punto, el DI contiene la dirección de DISAREA+6, y el SI contiene la dirección de PATTERN+6, que también es la dirección de DISAREA. Ahora la operación duplica de manera automática el patrón moviendo la primer palabra de DISAREA a DISAREA+6. DISAREA+2 a DISAREA+8, DISAREA+4 a DISAREA+10, y así sucesivamente. Final, el patrón está duplicado hasta el final de DISAREA:

```
      ***######*###**###**###***###    ...    ***###
       I          I          I              I
   PATTERN     DISAREA+6  DISAREA+12     DISAREA+42
```

Puede utilizar esta técnica para duplicar cualquier número de veces un patrón. El patrón puede ser de cualquier longitud, pero debe preceder de manera inmediata al campo destino.

## CÓMO ALINEAR A LA DERECHA EN LA PANTALLA

El programa de la figura 12-7 ilustra la mayor parte del material descrito en este capítulo. El procedimiento realiza lo siguiente:

- B10INPT acepta un nombre de hasta 30 caracteres de longitud en la parte superior de la pantalla.
- D10SCAS utiliza SCASB para barrer el nombre y evitar cualquier entrada que contenga un asterisco.
- E10RGHT utiliza MOVSB para alinear a la derecha de la pantalla cada nombre que es ingresado, uno debajo del otro. La longitud de ACTNLEN en la lista de parámetros de entrada es utilizada para calcular el carácter de más a la derecha en el nombre, como sigue:

```
                Babe Ruth
              Mickey Mantle
             Reggie Jackson
```

- F10CLNM utiliza STOSW para borrar el campo de entrada del teclado.

```
TITLE     P12RIGHT (EXE)  Nombres exhibidos justificados a la derecha
          .MODEL SMALL
          .STACK 64

          .DATA
NAMEPAR   LABEL  BYTE              ;Lista de parámetros de nombres
MAXNLEN   DB     31                /Longitud máxima
ACTNLEN   DB     ?                 ;No. de caracteres introducidos
NAMEFLD   DB     31 DUPC ')        ;Nombre

PROMPT    DB     'Ñame?', '$'
NAMEDSP   DB     31 DUP(' '), 13,  10, '$'
ROW       DB     00

          .CODE
BEGIN     PROC   FAR               .•Procedimiento principal
          MOV    AX,@data          ;Iniciar
          MOV    DS,AX             ;  segmento de datos
          MOV    ES,AX
          MOV    AX,0600H
          CALL   Q10SCR            ,-Despejar pantalla
          SUB    DX,DX              Fijar cursor en 00,00
          CALL   Q20CURS
A10LOOP:
          CALL   B10INPT           ;Petición de dar el nombre
          TEST   ACTNLEN,OFFH      ;¿No hay nombre? (indica fin)
          JZ     A90               ;  sí, salir
          CALL   D10SCAS           ,-Escudríñar asterisco
          CMP    AL,'*'            /¿Se encontró?
          JE     A10LOOP           ,- sí, saltado
          CALL   E10RGHT           /Justificar nombre a la derecha
          CALL   F10CLNM           /Despejar nombre
          JMP    A10LOOP
A90:      MOV    AX,4C00H          /Salir a DOS
          INT    21H
BEGIN     ENDP

          Indicación para entrada

B10INPT   PROC
          MOV    AH,09H
          LEA    DX, PROMPT        /Exhibir indicación
          INT    21H
          MOV    AH,OAH
          LEA    DX, NAMEPAR       ,-Aceptar entrada
          INT    21H
          RET
B10INPT   ENDP
í

          Escudriñar asterisco en nombre

DIOSCAS   PROC
          CLD                      /Izquierda a derecha
          MOV    AL,'*'            ,-Carácter a escudriñar
          MOV    CX, 30            /Fijar 30 bytes a escudriñar
          LEA    DI,NAMEFLD
          REPNE SCASB              ,-¿Se encontró un asterisco?
          JE     D20               ,- no, salir
          MOV    AL,20H            ,-  sí, despejar * en AL
D20:      RET
D10SCAS   "ENDP

/

          Justificar a la derecha y exhibir nombre

E10RGHT   PROC
```

**Figura 12-7** Justificación a la derecha en la pantalla

```
                STD                             /Izquierda a derecha
                MOV     CH,00
                MOV     CL,ACTNLEN              ,-Longitud en CX para REP
                LEA     SI,NAMEFLD              /Calcular la posición
                ADD     SI,CX                   /  más a la derecha
                DEC     SI                      /  del nombre que se ingresa
                LEA     DI,NAMEDSP+3 0          /Posición a la derecha de exhibición
ñame
                REP  MOVSB                      /Mover cadena derecha a
left
                MOV     DH,ROW
                MOV     DL, 4 8
                CALL    Q20CURS                 /Fijar cursor
                MOV     AH,09H
                LEA     DX,NAMEDSP              /Exhibir nombre
                INT     21H

                CMP     ROW,20                   iParte inferior de la pantalla?
                JAE     E20                       no,
                INC     ROW                       incrementar hilera
                JMP     E90
E20:
                MOV     AX,0601H                  Si,
                CALL    Q10SCR                    recorrer y
                MOV     DH,ROW                    fijar cursor
                MOV     DL,00
                CALL    Q20CURS
E90:            RET
E10RGHT         ENDP
                        Clear ñame:

F10CLNM         PROC
                CLD                             /Izquierda a derecha
                MOV     AX,2020H
                MOV     CX,15                   /Despejar 15 palabras
                LEA     DI,NAMEDSP
                REP  STOSW
                RET
F10CLNM         ENDP
                        Scroll screen:

Q10SCR          PROC                            /AX se fija al inicio
                MOV     BH,30                   /Atributo de color
                MOV     CX,00
                MOV     DX,184FH
                INT     10H
                RET
Q10SCR          ENDP
                        Set cursor row/col:

Q20CURS         PROC                            ,-DX se fija al inicio
                MOV     AH,02H
                SUB     BH,BH
                INT     10H
                RET
Q2 0CURS        ENDP
                END     BEGIN
```

Figura 12-7 (continuación)

PUNTOS CLAVE

- Para las instrucciones de cadenas de caracteres MOVS, STOS, CMPS y SCAS, asegúrese de que su programa .EXE inicializa el registro ES.
- Para instrucciones de cadenas, utilice los sufijos B, W o D para manejo de cadenas de byte, palabra o palabra doble.
- Ponga en unoXCLD) o en cero (STD) la bandera de dirección para la dirección necesaria de procesamiento.
- Verifique dos veces la inicialización de los registros DI y SI. Por ejemplo, MOVS implica los operandos DI,SI, mientras que CMPS implica los operandos SI,DI.
- Inicialice el registro CX de REP para procesar el número necesario de bytes, palabras o palabras dobles.
- Para procesamiento normal, utilice REP con MOVS y STOS, y utilice un REP condicional (REPE o REPNE) con CMPS y SCAS.
- CMPSW y SCASW invierten los bytes de las palabras que son comparadas.
- En donde necesite procesar de derecha a izquierda, tenga cuidado con la dirección inicial del campo de byte de la extrema derecha. Por ejemplo, si el campo es NAME1 y tiene una longitud de 10 bytes, entonces para procesar los bytes, la dirección que carga para LEA es NAME+9. Sin embargo, para procesar palabras la dirección que carga para LEA es ÑAME4-8 ya que la operación de cadena de caracteres accesa ÑAME+8 y NAME+9.

PREGUNTAS

**12-1.** Las operaciones con cadena de caracteres suponen que los operandos están relacionados con los registros DI o SI. Identifique estos registros para lo siguiente: (a) MOVS (operandos 1 y 2); (b) CMPS (operandos 1 y 2); (c) SCAS (operando 1).

**12-2.** Para operaciones con cadenas usando REP, ¿cómo define el número de repeticiones que ocurren?

**12-3.** Para operaciones con cadenas usando REP, ¿cómo establece el procesamiento de derecha a izquierda?

**12-4.** El capítulo da las instrucciones equivalentes a (a) MOVSB, (b) LODSB y (c) STOSB, cada una con prefijo REP. Para cada caso, proporcione el código equivalente para procesamiento de palabras.

**12-5.** Corrija el programa de la figura 12-1. Convierta el programa de formato .COM a .EXE, y asegúrese de inicializar el registro ES. Cambie las operaciones MOVSB y MOVSW para mover datos de derecha a izquierda. Utilice DEBUG para rastrear los procedimientos y observe el contenido del segmento de datos y de los registros.

**12-6.** Utilice la definición de datos siguiente y codifique operaciones con cadenas para las partes (a) - (f):

```
DATASG  SEGMENT PARA
        CONAME  DB 'SPACE LAUNCHES, LAUNCHES, INC
        PRLINE  DB 20 DUP(' •)
```

(a) Mover CONAME a PRLINE, de izquierda a derecha.
(b) Mover CONAME a PRLINE, de derecha a izquierda.
(c) Cargar el tercer y cuarto bytes de CONAME en el AX.
(d) Almacenar el AX empezando en PRLINE+5.

(e) Comparar CONAME con RLINE (serán diferentes).

(f) Rastrear CONAME por un carácter blanco y, si se encuentra uno, moverlo al BH.

12-7. Corregir el programa de la figura 12-6 de manera que la operación rastree en NAME1 la cadena "er". Un examen de NAME1 revela que los caracteres "er" no aparecen como una palabra, como se muestra a continuación: /As/se//mb/le/rs/. Existen dos posibles soluciones:

(a) Utilizar SCASW dos veces. El primer SCASW inicia en NAME1 y el segundo SCASW inicia en NAME1 + 1.

(b) Utilizar SCASB y, al encontrar una "e", comparar el siguiente byte contra una "r".

**12.8.** Definir un campo de cuatro bytes con el valor hexadecimal 030405B4. Utilice MOVSW para duplicar este campo 20 veces en un área de 80 bytes y despliegue el resultado.

CAPÍTULO 13

# Aritmética: I—Procesamiento de datos binarios de datos

OBJETIVO

Cubrir los requisitos para la suma, resta, multiplicación y división de datos binarios.

## INTRODUCCIÓN

Este capítulo estudia la suma, resta, multiplicación y división, y el uso de datos con y sin signo. También ofrece muchos ejemplos y advertencias sobre varios errores al viajero inexperto en el reino de los microprocesadores. El capítulo 14 cubre los requisitos especiales para la conversión entre formato de datos binarios y ASCII.

Aunque estamos acostumbrados a realizar aritmética en formato decimal (base 10), un microprocesador realiza su aritmética sólo en binario (base 2). Además, la limitación es de registros de 16 bits en procesadores anteriores al 80386 exige un tratamiento especial para números grandes.

Las instrucciones introducidas en este capítulo son:

| | | | |
|---|---|---|---|
| ADD | Suma | SUB | Resta |
| MUL | Multiplica sin signo | IMUL | Multiplica con signo |
| DIV | Divide sin signo | IDIV | Divide con signo |
| CBW | Convierte byte en word | NEG | Niega |

## SUMA Y RESTA

Las instrucciones ADD y SUB realizan sumas y restas sencillas de datos binarios. Como se describió en capítulos anteriores, los números binarios negativos están representados en la forma de complemento a dos: Invierta todos los bits del número positivo y sume 1. Los formatos generales para las instrucciones ADD y SUB son:

```
[etiqueta:]   ADD/SUB     {registro,registro}
[etiqueta:]   ADD/SUB     {memoria,registro}
[etiqueta:]   ADD/SUB     {registro,memoria}
[etiqueta:]   ADD/SUB     {registro,inmediato}
[etiqueta:]   ADD/SUB     {memoria,inmediato}
```

Como con otras instrucciones, no existen operaciones directas de memoria a memoria. El ejemplo siguiente utiliza el registro AX para sumar WORDA a WORDB:

```
WORDA   DW    123          /Define WORDA
WORDB   DW    25           ;Define  WORDB

        MOV   AX,WORDA     /Mueve WORDA al AX
        ADD   AX,WORDB     /Suma WORDB al AX
        MOV   WORDB,AX     ,-Mueve AX a WORDB
```

La figura 13-1 proporciona ejemplos de ADD y SUB para el procesamiento de valores en un byte y en una palabra. El procedimiento B10ADD utiliza ADD para procesar bytes y el procedimiento C10SUB utiliza SUB para procesar palabras.

### Desbordamientos

Esté alerta con los desbordamientos en las operaciones aritméticas. Ya que un byte sólo permite el uso de un bit de signo y siete bits de datos (desde -128 hasta +127), una operación aritmética, puede exceder con facilidad la capacidad de un registro de un byte. Y una suma en el registro AL que exceda su capacidad puede provocar resultados inesperados. Por ejemplo, suponga que el AL contiene 60H. Entonces la instrucción

```
ADD AL, 20H
```

genera una suma de 80H en el AL. Comq hemos sumado dos números positivos, esperamos qué la suma sea positiva, pero la operación pone en uno la bandera de desbordamiento y la bandera de signo en negativa. ¿La razón? El valor 80H, o 10000000 binario, es un número negativo; en lugar de +128 la suma es -128. El problema es que el registro AL es muy pequeño para la suma, que debe estar en el registro AX completo, como se muestra en la sección siguiente.

```
TITLE   P13ADD (COM)   Operaciones ADD y SUB
        .MODEL SMALL
        .CODE
        ORG     100H
BEGIN:  JMP     SHORT MAIN

BYTEA   DB      64H                 ;Datos
BYTEB   DB      4OH
BYTEC   DB      16H
WORDA   DW      4000H
WORDB   DW      2000H
WORDC   DW      1000H

MAIN    PROC    NEAR            | Procedimiento principal:
        CALL    B10ADD          | Llama a la rutina ADD
        CALL    C10SUB          | Llama a la rutina SUB
        MOV     AX,4C00H        | Sale al DOS
        INT     21H
MAIN    ENDP

        Ejemplos de SUMA (ADD) de bytes:

B10ADD  PROC
        MOV     AL,BYTEA
        MOV     BL,BYTEB
        ADD     AL,BL           | Registro a registro
        ADD     AL,BYTEC        | Memoria a registro
        ADD     BYTEA,BL        | Registro a memoria
        ADD     BL,10H          | Inmediato a registro
        ADD     BYTEA,25H       | Inmediato a memoria
        RET
B10ADD  ENDP

1       Ejemplos de RESTA (SUB) de palabras:

¿10SUB  PROC
        MOV     AX,WORDA
        MOV     BX,WORDB
        SUB     AX,BX           | Registro de registro
        SUB     AX,WORDC        | Memoria de registro
        SUB     WORDA,BX        | Registro de memoria
        SUB     BX,1000H        | Inmediato de registro
        SUB     WORDA,256H      | Inmediato de memoria
        RET
C10SUB  ENDP
        END     BEGIN
```

Figura 13-1 Ejemplos del uso de ADD y de SUB

### Extensión de un número en un registro

En la sección anterior vimos cómo al sumar 20H al número 60H en el AL provoca una suma incorrecta. Una mejor solución sería que el AX representara la suma de manera adecuada. La instrucción para este propósito es CBW (convierte byte en palabra), que de forma automática envía el bit de signo del AL (0 o 1) al AH. Observe que el CBW está restringido para el uso del AX.

En el ejemplo siguiente, CBW extiende el signo (0) en el AL al AH, que genera 0060H en el AX. Después, el código suma 20H al AX (en lugar de al AL) y genera el resultado correcto en el AX:0080H, o +128:

```
                                                        AH    AL
                                                        XX    SOH
CBW                 /Extiende el signo de AL al AH      00    60
ADD    AX.20H/      /Suma al AX                         00    80
```

El resultado numérico en el segundo ejemplo es el mismo, pero la operación en el AX no lo trata como desbordamiento o negativo. Aun así, aunque una palabra completa en el AX permite un bit de signo y 15 bits de datos, el AX está limitado a números desde -32,768 hasta +32,767. La sección siguiente examina cómo manejar números que excedan estos límites.

## ARITMÉTICA CON PALABRAS MÚLTIPLES

Como hemos visto, valores numéricos grandes pueden exceder la capacidad de una palabra, y en realidad se necesita la capacidad de palabras múltiples. Un requisito principal en aritmética de palabras múltiples es el byte y palabra en secuencia inversa. Recuerde que el ensamblador convierte de manera automática el contenido de las palabras numéricas definidas en secuencia inversa de bytes, así que, por ejemplo, una definición de 0134H se convierte en 3401H. Pero en los valores en palabras dobles, es responsabilidad de *usted* definir el par relacionado de palabras en secuencia inversa de palabras. Digamos que un par de palabras dobles es como éste:

```
Hex | 01 23 | BC 62 |
```

Entonces usted tiene que definir las *palabras* en orden inverso:

```
DW   0BC62H
DW   0123H
```

Entonces el ensamblador convierte estas definiciones en secuencia inversa de bytes, adecuada para aritmética con palabras dobles:

```
Hex | 62 BC | 23 01 |
```

Examinemos dos maneras de realizar aritmética de palabras múltiples. La primera es sencilla y específica, mientras que la segunda es más elaborada y general.

En la figura 13-2, el procedimiento D10DWD ilustra la suma de un par de palabras (WORD1A y WORD IB) a un segundo par (WORD2A y WORD2B) y almacena la suma en un tercer par (WORD3A y WORD3B). En efecto, la operación es para sumar los números, tal como lo siguiente:

```
Número inicial:   0123    BC62H
Sumar:            0012    553AH
Total:            0136    119CH
```

A causa de la secuencia inversa de bytes en memoria, el programa define los números con las palabras al revés: BC62 0123 y 553A 0012, respectivamente. Entonces el ensamblador almacena en la memoria valores de palabras dobles en la secuencia inversa de bytes correcta:

```
TITLE   P13DBADD (COM)   Suma de palabras dobles
        .MODEL  SMALL
        .CODE
        ORG     100H
BEGIN:  JMP     SHORT MAIN

WORDlA  DW      0BC62H              /Datos
WORDIB  DW      0123H
WORD2A  DW      553AH
WORD2B  DW      0012H
WORD3A  DW      ?
WORD3B  DW      ?

MAIN    PROC    NEAR                Procedimiento principal
        CALL    D10DWD              Llama al primer ADD
        CALL    E10DWD              Llama al segundo ADD
        MOV     AX,4C00H            Sale al DOS
        INT     21H
MAIN    ENDP
        Ejemplo de SUMA (ADD) de palabras dobles:

D10DWD  PROC
        MOV     AX.WORD1A            Suma la palabra de extrema izquierda
        ADD     AX,WORD2A
        MOV     WORD3A,AX
        MOV     AX,W0RD1B           ;Suma la palabra de extrema derecha
        ADC     AX,WORD2B             con acarreo
        MOV     WORD3B,AX
        RET
D10DWD  ENDP
        Operación de suma generalizada:

E10DWD  PROC
        CLC                         Pone en cero la bandera de acarreo
        MOV     CX,02               Designa el contador del ciclo
        LEA     SI,W0RD1A           Palabra de la izquierda
        LEA     DI,WORD2A           Palabra de la izquierda
        LEA     BX,WORD3A           Palabra de la izquierda de la suma
E20:
        MOV     AX,[SI]             Mueve la palabra al AX
        ADC     AX,[DI]             Suma con acarreo al AX
        MOV     [BX],AX             Almacena la palabra
        INC     SI                  Ajusta las direcciones para
        INC     SI                    la siguiente palabra de la derecha
        INC     DI
        INC     DI
        INC     BX
        INC     BX
        LOOP    E20                 ,-Repite para la palabra siguiente
        RET
E10DWD  ENDP
        END     BEGIN
```

Figura 13-2 Suma de palabras múltiples

```
WORD1A y WORD1B:    G2BC   2301

WORD2A y WORD2B:    3A55   1200
```

El primer procedimiento suma WORD2A a WORDlA en el AX (en realidad son las partes de bajo orden) y almacena la suma en WORD3A. A continuación suma WORD2B a WORD1B (las partes de orden superior) en el AX, junto con el acarreo de la suma anterior. Después almacena la

suma en WORD3B. Examinemos las operaciones en detalle. El primer MOV y la operación ADD invierten los bytes en el AX y suman las palabras de la extrema izquierda:

```
WORD1A:      BCS2H
WORD2A:     +553AH
Total:     (D119CH     (9C11H es almacenado en WORD3A)
```

Ya que la suma de WORD1A más WORD2A excede la capacidad del AX, ocurre un acarreo y la bandera de acarreo es puesta en uno. Ahora, el ejemplo suma las palabras de la derecha, pero esta vez utilizando ADC (sumar **con** acarreo) en lugar de ADD. ADC suma los dos números y ya que la bandera de acarreo está en uno, suma uno a la suma:

```
WORD IB:            0123H
WORD2B:            +0012H
Más el acarreo:    +   1H
Total:              0136H     (3601H es almacenado en WORD3B)
```

Por medio de DEBUG rastree la aritmética: puede ver la suma 0136H en el AX y los valores en orden inverso 9C11H en WORD3A y 3601H en WORD3B.

También en la figura 13-2, el procedimiento más elaborado E10DWD proporciona un enfoque para sumar números de cualquier longitud aunque aquí, como antes, se suma la misma pareja de palabras W0RD1A:W0RD1B y WORD2A:WORD2B. El procedimiento utiliza el SI, DI y BX como registros base para las direcciones de WORD1A, WORD2A y WORD3A, respectivamente. Se realiza una iteración a través de las instrucciones por cada par de palabras que se suman —en este caso, dos veces. El primer ciclo suma las palabras de la extrema izquierda, y el segundo suma las de la extrema derecha. Ya que el segundo ciclo es para procesar las palabras de la derecha, las direcciones en los registros SI, DI y BX se incrementan en 2. Para cada registro, dos instrucciones INC realizan esta operación. Se emplea INC (en lugar de ADD) por una buena razón: la instrucción reg,02 limpiaría la bandera de acarreo y causaría una respuesta incorrecta, mientras que INC no afecta la bandera de acarreo.

A causa del ciclo, sólo existe una instrucción ADC. Al inicio, una instrucción CLC (pone en cero el acarreo) asegura que la bandera de acarreo esté inicialmente en cero. Para hacer que este método funcione, asegúrese de (1) definir palabras adyacentes una de otra, (2) procesar palabras de izquierda a derecha y (3) inicializar el CX al número de palabras que serán sumadas.

Para resta de múltiples palabras, la instrucción equivalente a ADC es SBB (restar con préstamo). En el procedimiento E10DWD, sólo reemplace ADC con SBB.

### Aritmética en registros de 32 bits

El 80386 y procesadores posteriores proveen registros de 32 bits para aritmética con palabras dobles. Por ejemplo, para sumar el EBX al EAX sólo codifique

```
ADD EAX, EBX  ,-registros de 32 bits
```

**Puede** sumar palabras cuádruples utilizando la técnica estudiada antes para sumar palabras múltiples.

## DATOS CON SIGNO Y SIN SIGNO

Algunos campos numéricos carecen de signo; por ejemplo, un número de cliente y una dirección de memoria. Otros campos numéricos pueden tener números positivos o negativos; por ejemplo, el saldo de un cliente y un número algebraico. Y otros campos numéricos con signo —por ejemplo, el sueldo de un empleado, el día del mes y el valor de pi— se supone que siempre son positivos.

Para datos sin signo, todos los bits tienen el propósito de ser bits de datos; de aquí que, en lugar de un máximo de 32,767, un registro de 16 bits puede contener 65,535. Para datos con signo, el bit de la extrema izquierda es un bit de signo. Pero observe que las instrucciones ADD y SUB no distinguen ente datos con y sin signo: en realidad, sólo suman y restan bits. El ejemplo siguiente ilustra la suma de dos números binarios, con los valores tomados sin signo, primero, y después con signo. El número de arriba tiene un bit en 1 a la izquierda; para datos sin signo, los bits representan 249, mientras que para datos con signo los bits representan -7. La suma no pone en uno las banderas de acarreo ni de desbordamiento:

| BINARIO | DECIMAL SIN SIGNO | DECIMAL CONSIGNO | OF | CF |
|---|---|---|---|---|
| 11111001 | 249 | -7 | | |
| +00000010 | +2 | +2 | | |
| 11111011 | 251 | -5 | 0 | 0 |

El resultado binario de la suma en este ejemplo es el mismo tanto para datos con signo como datos sin signo. Sin embargo, los bits en el campo sin signo representan el 251 decimal, mientras que los bits en el campo con signo representan el -5 decimal. En realidad, el contenido de un campo significa cualquier cosa que usted quiera que signifique.

### Aritmética con acarreo

Una operación aritmética que causa un acarreo externo (hacia afuera) del bit de signo también pone en uno la bandera de acarreo. Si ocurre un acarreo en datos sin signo, el resultado no es válido. El ejemplo siguiente de una suma provoca un acarreo:

| BINARIO | DECIMAL SIN SIGNO | DECIMAL CONSIGNO | OF | CF |
|---|---|---|---|---|
| 11111100 | 252 | -4 | | |
| +00000101 | +5 | +5 | | |
| (1)00000001 | 1 | 1 0 | | 1 |
| | (no válido) | (válido) | | |

La operación sobre los datos sin signo no es válida a causa del acarreo externo de un bit de datos, mientras que la operación en datos con signo es válida.

### Desbordamiento aritmético

Una operación aritmética pone en uno la bandera de desbordamiento cuando se tiene un acarreo *hacia* el bit de signo (acarreo interno) y no se tiene un acarreo hacia afuera, o bien ocurre un acarreo *externo* sin acarreo interno. En donde ocurra un desbordamiento en datos con signo, el resultado es no válido (a causa de un desbordamiento en el bit de signo), como lo muestra este ejemplo:

| BINARIO | DECIMAL SIN SIGNO | DECIMAL CONSIGNO | OF | CF |
|---|---|---|---|---|
| 01111001 | 121 | +121 | | |
| +00001011 | +11 | +11 | | |
| 10000100 | 132 | -124 | 1 | 0 |
| | (válido) | (no válido) | | |

Una suma puede poner en uno las dos banderas, la de acarreo y la de desbordamiento. En el ejemplo siguiente, el acarreo hace que la operación sin signo sea no válida, y así mismo el desbordamiento hace que la operación con signo sea no válida:

| BINARIO | DECIMAL SIN SIGNO | DECIMAL CONSIGNO | OF | CF |
|---|---|---|---|---|
| 11110110 | 246 | - 10 | | |
| +10001001 | +137 | -119 | | |
| (1)01111111 | 127 | +127 | | |
| | (no válido) | (no válido) | | |

El resultado de todo esto es que usted debe tener una buena idea de cuál es la magnitud de los números que su programa procesará, y debe definir el tamaño de los campos de acuerdo con esto.

## MULTIPLICACIÓN

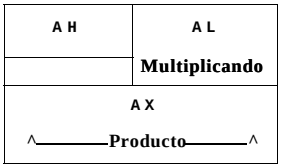
Para la multiplicación, la instrucción MUL maneja datos sin signo y la instrucción IMUL (multiplicación entera) maneja datos con signo. Ambas instrucciones afectan las banderas de acarreo y de desbordamiento. Como programador, usted tiene el control sobre el formato de los datos que procesa, y tiene la responsabilidad de seleccionar la instrucción de multiplicación apropiada. El formato general para MUL e IMUL es

```
[etiqueta:]  MUL/IMUL  registro/memoria
```

Las operaciones de multiplicación básicas son byte por byte, palabra por palabra y (para el 80386 y procesadores posteriores) palabras dobles por palabras dobles.

### Byte por byte

Para multiplicar dos números de un byte, el multiplicando está en el registro AL y el multiplicador es un byte en memoria o en otro registro. Para la instrucción MUL DL, la operación multiplica el contenido del AL por el contenido del DL. El producto generado está en el registro AX. La operación ignora y borra cualquier información que pueda estar en el AH.



### Palabra por palabra

Para multiplicar dos números de una palabra, el multiplicando está en el registro AX y el multiplicando es una palabra en memoria o en otro registro. Para la instrucción MUL DX, la operación multiplica el contenido del AX por el contenido del DX. El producto generado es una palabra doble que necesita dos registros: la parte de orden alto (más a la izquierda) en el DX y la parte de orden bajo (más a la derecha) en el AX. La operación ignora y borra cualquier información que pueda estar en el DX.

| | DX | AX |
|---|---|---|
| Antes de multiplicar: | Ignorado | Multiplicando |
| Después de multiplicar: | Parte alta de producto | Parte baja de producto |

### Palabra doble por palabra doble

Para multiplicar dos números de palabras dobles, el multiplicando está en el registro EAX y el multiplicador es una palabra doble en memoria o en otro registro. El producto es generado en el par EDX:EAX. La operación ignora y borra cualquier información que ya esté en el EDX.

| | EDX | EAX |
|---|---|---|
| Antes de multiplicar: | Ignorado | Multiplicando |
| Después de multiplicar: | Parte alta de producto | Parte baja de producto |

### Tamaño de campo

El operando de MUL o IMUL sólo hace referencia al multiplicador, que determina el tamaño del campo. En los ejemplos siguientes, el multiplicador está en un registro, el cual especifica el tipo de operación:

| INSTRUCCIÓN | MULTIPLICADOR | MULTIPLICANDO | PRODUCTO |
|---|---|---|---|
| MUL CL | byte | AL | AX |
| MUL BX | palabra | AX . | DX:AX |
| MUL EBX | palabra doble | EAX | EDX: EAX |

En los ejemplos siguientes, los multiplicadores están definidos en memoria:

```
BYTEl   DB   ?
WORDl   DW   ?
DWORD1  DD   ?
```

| OPERACIÓN | MULTIPLICADOR | MULTIPLICANDO | PRODUCTO |
|---|---|---|---|
| MUL BYTEl | BYTEl | AL | AX |
| MUL WORDl | WORDl | AX | DX:AX |
| MUL DWORD1 | DWORD1 | EAX | EDX: EAX |

### Multiplicación sin signo: MUL

El objetivo de la instrucción MUL es multiplicar datos sin signo. En la figura 13-3, C10MUL da tres ejemplos del uso de MUL: byte por byte, palabra por palabra y palabra doble por palabra doble. El primer ejemplo multiplica 80H (128) por 40H (64). El producto en el AX es 2000H (8,192). El segundo ejemplo genera 1000 0000H en los registros DX:AX.

El tercer ejemplo multiplica una palabra por un byte y necesita extender BYTE1 a una palabra. Ya que los números se suponen sin signo, el ejemplo supone que los bits en el registro AH son cero. (Aquí el problema con el uso de CBW es que el bit de la extrema izquierda del AL podría ser uno, y la propagación de bits uno en el AH generaría en un número sin signo mayor.) El producto en el DX:AX es 0040 0000H.

### Multiplicación con signo: IMUL

El objetivo de la instrucción IMUL (multiplicación entera) es multiplicar datos con signo. En la figura 13-3, D10IMUL da los mismos tres ejemplos que C10MUL, pero reemplaza MUL con IMUL.

El primer ejemplo multiplica 80H (un número negativo) por 40H (un número positivo). El producto en el registro AX es E000H. Usando los mismos datos, MUL genera un producto de 2000H, así que puede ver la diferencia entre el uso de MUL y de IMUL. MUL trata 80H como +128, mientras que IMUL lo trata como -128. El producto de -128 por +64 es -8192H, que es igual a E000H. (Intente convirtiendo E000H a bits, invierta los bits, sume 1 y sume los valores de los bits.)

El segundo ejemplo multiplica 8000H (un número negativo) por 2000H (un número positivo). El producto en el DX:AX es F000 0000H, que es el negativo del producto generado por MUL.

El tercer ejemplo extiende BYTE1 a una palabra en el AX. Ya que los números se suponen con signo, el ejemplo utiliza CBW para extender el bit del signo de la extrema izquierda en el registro AH: 80H en el AL se convierte en FF80H en el AX. Ya que el multiplicador, WORD1, también es negativo, el producto debe ser positivo. Y en realidad así es: 0040 0000H en el DX:AX, el mismo resultado que MUL, que multiplicó dos números sin signo.

En efecto, si el multiplicando y el multiplicador tienen el bit del mismo signo, IMUL y MUL generan el mismo producto. Pero si el multiplicando y el multiplicador tienen bits de signos diferentes, MUL produce un producto positivo e IMUL produce un producto negativo. El resultado es que su programa debe conocer el formato de los datos y utilizar las instrucciones apropiadas.

Puede encontrar útil usar DEBUG para rastrear estos ejemplos.

## MULTIPLICACIÓN DE PALABRAS MÚLTIPLES

La multiplicación convencional consiste en la multiplicación byte por byte, palabra por palabra, o bien, palabra doble por palabra doble. Como ya se ha viso, el número máximo con signo en una palabra es +32,767. La multiplicación de números mayores en procesadores anteriores al 80386 exige pasos adicionales. El enfoque en estos procesadores es multiplicar cada palabra por separado y después sumar cada producto. El ejemplo siguiente multiplica un número decimal de cuatro dígitos por un número de dos dígitos:

```
TITLE   P13MULT (COM)  Operaciones MUL e IMUL
        .MODEL SMALL
        .CODE
        ORG    100H
BEGIN:  JMP    SHORT MAIN

BYTEl   DB     80H
BYTE2   DB     4 OH
WORDl   DW     8000H
W0RD2   DW     2000H

MAIN    PROC   NEAR              /Procedimiento principal
        CALL   ClOMUL            /Llama a la rutina MUL
        CALL   DIOIMUL           /Llama a la rutina IMUL
        MOV    AX,4C00H          /Sale al DOS
        INT    21H
MAIN    ENDP
                Ejemplos de MUL:

ClOMUL  PROC
        MOV    AL,BYTEl          /Byte por byte
        MUL    BYTE2             / el producto en AX

        MOV    AX,WORDl          /Palabra por palabra
        MUL    WORD2             /  el producto en DX:AX

        MOV    AL,BYTEl          ,-Byte por palabra
        SUB    AH, AH            /  extiende el multiplicando en AH
        MUL    WORDl             /  el producto queda en DX:AX
        RET
ClOMUL  ENDP
1               Ejemplos de IMUL:

DIOIMUL PROC
        MOV    AL, BYTEl         ,-Byte por byte
        IMUL   BYTE2             /  el producto en AX

        MOV    AX,WORDl          /Palabra por palabra
        IMUL   WORD2             /  el producto en DX:AX

        MOV    AL,BYTEl          /Byte por palabra
        CBW                        extiende el multiplicando en AH
        IMUL   WORDl             /  el producto queda en DX:AX
        RET
DIOIMUL ENDP
        END    BEGIN
```

Figura 13-3 Multiplicación con signo y sin signo

$$\begin{array}{r} 1{,}365 \\ \times \quad 12 \\ \hline 16{,}380 \end{array}$$

¿Qué pasa si usted sólo puede multiplicar números de dos dígitos? Entonces podría multiplicar por separado el 13 y el 65 por 12, como:

$$\begin{array}{rr} 13 & 65 \\ \times \ 12 & \times \ 12 \\ \hline 156 & 780 \end{array}$$

Y después sumar los dos productos; pero recuerde, ya que el 13 son los cientos, su producto en realidad es 15,600:

```
  15,600      (13 x 12 x 100)
+    780      (65 x 12)
  16,380
```

Un programa en ensamblador puede usar esta misma técnica, salvo que los datos consisten de palabras (cuatro dígitos) en formato hexadecimal. Examinemos ahora los requisitos para multiplicar una palabra doble por una palabra y una palabra doble por una palabra doble.

### Palabra doble por palabra

En la figura 13-4, E10XMUL multiplica una palabra doble por una palabra. El multiplicando MULTCND, consiste en dos palabras con 3206H y 2521H, respectivamente. La razón de definir dos DW (palabras) en lugar de una DD (palabra doble) es para facilitar el direccionamiento para las instrucciones MOV que mueven palabras al registro AX. Los números están definidos en secuencia inversa de palabra, y el ensamblador almacena cada palabra en secuencia inversa de byte. Así MULTCND, que tiene un valor definido de 32062521H, es almacenado como 21250632H.

```
TITLE     P13DWMUL (COM)   Multiplicación de palabras dobles
          .MODEL SMALL
          .CODE
          ORG     100H
BEGIN:    JMP     SHORT MAIN
-------------------------------------------------------------------
MULTCND   DW      2521H              Datos
          DW      3206H
MULTPLR   DW      0A26H
          DW      64 OOH
PRODUCT   DW      0
          DW      0
          DW      0
          DW      0
-------------------------------------------------------------------
MAIN      PROC    NEAR               Procedimiento principal
          CALL    E10XMUL            Llama a la primera multiplicación
          CALL    Z10ZERO            limpia el producto
          CALL    F10XMUL            Llama a la segunda multiplicación
          MOV     AX.4C00H           Sale al DOS
          INT     21H
MAIN      ENDP
                  Palabra doble por palabra
-------------------------------------------------------------------
E10XMUL   PROC
          MOV     AX,MULTCND         /Multiplica la palabra de la izquierda
          MUL     MULTPLR+2          ;  del multiplicando
          MOV     PRODUCT,AX         /Almacena el producto
          MOV     PRODUCT+2,DX

          MOV     AX,MULTCND+2       •Multiplica la palabra de la derecha
          MUL     MULTPLR+2            del multiplicando
          ADD     PRODUCT+2,AX       •Suma el producto almacenado
          ADC     PRODUCT+4,DX
          RET
E10XMUL   ENDP
```

Figura 13-4 Multiplicación de palabras múltiples

```
                    Palabra doble por palabra doble:

F10XMUL   PROC
          MOV     AX,MULTCND        ,-Palabra uno del multiplicando por
          MUL     MULTPLR           ;  palabra uno del multiplicador
          MOV     PRODUCT+0,AX      /Almacena el producto
          MOV     PRODUCT+2,DX

          MOV     AX,MULTCND        /Palabra uno del multiplicando por
          MUL     MULTPLR+2         /  palabra dos del multiplicador
          ADD     PRODUCT+2,AX      /Suma al producto almacenado
          ADC     PRODUCT+4,DX
          ADC     PRODUCT+6,00      /Suma con acarreo

          MOV     AX,MULTCND+2      /Palabra dos del multiplicando por
          MUL     MULTPLR           /  palabra uno del multiplicador
          ADD     PRODUCT+2,AX      /Suma al producto almacenado
          ADC     PRODUCT+4,DX
          ADC     PRODUCT+6,00      /Suma con acarreo

          MOV     AX,MULTCND+2      /Palabra dos del multiplicando por
          MUL     MULTPLR+2         /  palabra dos del multiplicador
          ADD     PRODUCT+4,AX      /Suma al producto
          ADC     PRODUCT+6,DX
          RET
F10XMUL   ENDP
                    Limpia el área del producto:

Z10ZERO   PROC
          MOV     PRODUCT,0000      /Limpia las palabras
          MOV     PRODUCT+2,0000    /  de izquierda a derecha
          MOV     PRODUCT+4,0000
          MOV     PRODUCT+6,0000
          RET
ZlOZERO   ENDP
          END     BEGIN
```

Figura 13-4B (continuación)

El multiplicador, MULTPLR+2, contiene 6400H. El campo para el producto generado, PRODUCT, mantiene tres palabras. La primera operación MUL multiplica MULTPLR+2 y la palabra izquierda de MULTCND; el producto es 0E80 E400H hexadecimal, almacenado en PRODUCT+2 y PRODUCT+4. El segundo MUL multiplica MULTPLR+2 y la palabra derecha de MULTCND; el producto es 138A 5800H. Entonces, la rutina suma los dos productos así:

```
Producto 1:     0000    0E80    E400
Producto 2:    +138A    5800
Total:          138A    6680    E400
```

Como el primer ADD puede provocar un acarreo, la segunda suma es ADC (suma con acarreo). Ya que los datos numéricos están almacenados en formato inverso de bytes, PRODUCT en realidad contiene 00E4 8066 8A13. La rutina necesita que la primer palabra de PRODUCT al principio contenga cero.

### Palabra doble por palabra doble

La multiplicación de dos palabras dobles en procesadores anteriores al 80386 implica cuatro multiplicaciones:

| MULTIPLICANDO | | MULTIPLICADOR |
|---|---|---|
| palabra 2 | x | palabra 2 |
| palabra 2 | x | palabra 1 |
| palabra 1 | x | palabra 2 |
| palabra 1 | x | palabra 1 |

Se suma cada producto en el DX y AX para la palabra apropiada en el producto final. En la figura 13-4, F10XMUL da un ejemplo. MULTCND contiene 3206 2521H, MULTPLR contiene 6400 0A26H y PRODUCT mantiene cuatro palabras.

Aunque la lógica es semejante a la multiplicación de una palabra doble por una palabra, este problema necesita una característica adicional. Después de la pareja ADD/ADC está otro ADC que suma cero a PRODUCT. El primer ADC puede provocar un acarreo, que instrucciones siguientes limpiarían. Por lo tanto, el segundo ADC suma cero si no hay acarreo y uno si existe alguno. La última pareja ADD/ADC no necesita un ADC adicional: Ya que PRODUCT es suficientemente grande para la respuesta final generada, no existe acarreo.

El producto final es 138A 687C 8E5C CCE6, almacenado en PRODUCT con los bytes invertidos. Trate de usar DEBUG para rastrear este ejemplo.

## INSTRUCCIONES ESPECIALES DE MULTIPLICACIÓN

El 80286 y procesadores posteriores tienen formatos adicionales para IMUL que proporcionan operandos inmediatos y permiten generar productos en registros distintos del AX. Puede utilizar estas instrucciones para muliplicar datos con y sin signo, ya que los resultados son los mismos. Todos los números deben tener la misma longitud: 16 o, para el 80386 y procesadores posteriores, 32 bits.

### Operación IMUL en 16 bits

Para el IMUL en 16 bits el primer operando (un registro) contiene el multiplicando y el segundo operando (un número inmediato) es el multiplicador. El producto es generado en el primer operando. Un producto que excede el registro causa que las banderas de acarreo y de desbordamiento se pongan en uno. El formato general para esta operación de IMUL de 16 bits es

```
[etiqueta:]   IMUL  registro,inmediato
```

### Operación IMUL en 32 bits

El IMUL en 32 bits tiene tres operandos: el segundo operando (memoria) contiene el multiplicando y el tercer operando (un número inmediato) contiene el multiplicador. El producto es generado en el primer operando (un registro). El formato general para el IMUL de 32 bits es

```
[etiqueta:]   IMUL  registro,memoria,inmediato
```

### Operación IMUL en 16/32 bits

El 80386 y procesadores posteriores proporcionan otro formato IMUL para las operaciones de 16 o 32 bits. El primer operando (un registro) contiene el multiplicando y el segundo operando (registro/memoria) contiene el multiplicador. El producto es generado en el primer operando.

```
[etiqueta:]  IMUL  registro,{registro/memoria}
```

He aquí ejemplos de estas tres instrucciones IMUL:

| | | Multiplicando | Multiplicador | Producto |
|---|---|---|---|---|
| 16-bit IMUL: | IMUL DX,25 | DX | 25 | DX |
| 32-bit IMUL: | IMUL ECX,MULTCAND,25 | MULTCAND | 25 | ECX |
| 16/32-bit IMUL: | IMUL BX,CX | BX | CX | BX |

## MULTIPLICACIÓN POR CORRIMIENTO

Para multiplicar por una potencia de 2 (2, 4, 8, etc.) es más eficiente sólo correr el número necesario de bits a la izquierda. Para el 8088/8086, un corrimiento mayor a uno necesita que cargue el número de corrimientos en el registro CL. En los ejemplos siguientes, el multiplicando está en el AX:

```
Multiplicar por 2 (un corrimiento a la izquierda):  SHL AX.01
Multiplicar por 8 (tres corrimientos a la izquierda): MOV CL,03   ;8088/8086
                                                     SHL AX,CL

Multiplicar por 8 (tres corrimientos a la izquierda): SHL AX,03   ;80286 y posteriores
```

### Corrimiento en los registros DX:AX

La rutina siguiente puede ser útil para obtener un producto por corrimientos a la izquierda en los registros DX:AX. Puede idear un método más eficiente, pero este ejemplo es generalizado a cualquier número de ciclos (y corrimientos) en el CX. Observe que un bit 1 corrido fuera del registro entra a la bandera de acarreo, la cual es utilizada por RCL:

```
        MOV   CX,04     ;Inicializa para cuatro ciclos
C20:    SHL   AX,01     /Corrimiento del AX
        RCL   DX,01     /Rota el DX a la izquierda
        LOOP  C20       /Repite
```

El método siguiente para corrimientos a la izquierda necesita de un 80286 o procesador posterior y no requiere de ciclos. Aunque es específico para un corrimiento de cuatro bits, puede ser adaptado a otros valores:

```
SHL     DX,04     Corrimiento del DX 4 bits a la izquierda
MOV     BL,AH     Almacena el AH en el BL
SHL     AX,04     Corrimiento del AX 4 bits a la izquierda
SHR     BL,04     Corrimiento del BL 4 bits a la derecha
OR      DL,BL     Inserta el BL 4 bits en el DL
```

## DIVISIÓN

Para la división, la instrucción DIV (dividir) maneja datos sin signo y la IDIV (división entera) maneja datos con signo. Usted es responsable de seleccionar la instrucción apropiada. El formato general para DIV/IDIV es

```
[etiqueta:]  IDIV/DIV  {registro/memoria}
```

Las operaciones de división básicas son palabra entre byte, palabra doble entre palabra y (para 80386 y posteriores) palabra cuádruple entre palabra doble.

### Palabra entre byte

Aquí, el dividendo está en el AX y el divisor es un byte en memoria o en otro registro. Después de la división, el residuo está en el AH y el cociente está en el AL. Ya que un cociente de un byte es muy pequeño —si es sin signo, un máximo de +255 (FFH) y con signo +127 (7FH)— esta operación tiene un uso limitado.

| | | |
|---|---|---|
| **Antes de la división:** | **AX** | |
| | **- Dividendo -** | |
| **Después de la división:** | **AH** | **AL** |
| | **Residuo** | **Cociente** |

### Palabra doble entre palabra

Para esta operación, el dividendo está en el par DX: AX y el divisor es una palabra en memoria c en otro registro. Después de la división, el residuo está en el DX y el cociente está en el AX. El cociente de una palabra permite para datos sin signo un máximo de +32,767 (FFFFH) y con signe + 16,383 (7FFFH). Tenemos:

| | | |
|---|---|---|
| **Antes de la división:** | **DX** | **AX** |
| **Después de la división:** | **Parte alta del dividendo** | **Parte baja del dividendo** |
| | **Residuo** | **Cociente** |

### Palabra cuádruple entre palabra doble

Al dividir una palabra cuádruple entre una palabra doble, el dividendo está en el par EDX:EAX j el divisor está en una palabra doble en memoria o en otro registro. Después de la división, e residuo está en el EDX y el cociente en el EAX.

| | | |
|---|---|---|
| **Antes de la división:** | **DX** | **AX** |
| **Después de la división:** | **Parte alta del dividendo** | **Parte baja del dividendo** |
| | **Residuo** | **Cociente** |

### Tamaños del campo

El operando de DIV o de IDIV hace referencia al divisor, que especifica el tamaño del campo. En los ejemplos siguientes de DIV, los divisores están en un registro, que determina el tipo de operación:

| OPERACIÓN | DIVISOR | DIVIDENDO | COCIENTE | RESIDUO |
|---|---|---|---|---|
| DIV CL | byte | AX | AL | AH |
| DIV CX | palabra | DX:AX | AX | DX |
| DIV EBX | palabra doble | EDX: EAX | EAX | EDX |

En los ejemplos siguientes de DIV, los divisores están definidos en memoria:

```
BYTEl    DB
WORDl    DW
DWORD1   DD
```

| | | DIVISOR | DIVIDENDO | COCIENTE | RESIDUO |
|---|---|---|---|---|---|
| DIV | BYTEl | BYTEl | AX | AL | AH |
| DIV | WORDl | WORDl | DX:AX | AX | DX |
| DIV | DWORD1 | DWORD1 | EDX:EAX | EAX | EDX |

**Residuo.** Si divide 13 entre 3, el resultado es *4*\, donde el cociente es 4 y el residuo es 1. Note que una calculadora (y un lenguaje de programación de alto nivel) enviaría como cociente 4.333..., que consiste en una parte entera (4) y una parte fraccionaria (.333...). Los números **3**y .333 son fracciones, mientras que 1 es un residuo.

### División sin signo: DIV

El objetivo de la operación DIV es dividir datos sin signo. La figura 13-5 da cuatro ejemplos de DIV en el procedimiento D10DIV: una palabra entre un byte, un byte entre un byte, una palabra doble entre una palabra y una palabra entre una palabra. El primer ejemplo divide 2000H (8092) entre 80H (128). El residuo en el AH es 00H y el cociente en el AL es 40H (64).

El segundo ejemplo necesita extender BYTEl a una palabra. Como el valor se supone sin signo, el ejemplo supone que los bits en el registro AH son cero. El residuo en el AH es 12H y el cociente en el AL es 05H.

En el tercer ejemplo, el residuo en el DX es 1000H y el cociente en el AX es 0080H.

El cuarto ejemplo necesita extender WORDl a una palabra doble en el registro DX. Después de la división, el residuo en el DX es 0000H y el cociente en el AX es 0002H.

### División con signo: IDIV

El objetivo de la instrucción IDIV es dividir datos con signo. En la figura 13-5, E10IDIV da los mismos cuatro ejemplos que D10DIV, pero reemplazando DIV con IDIV. El primer ejemplo divide 2000H (positivo) entre 80H (negativo). El residuo en el AH es 00H, y el cociente en el AL es COH (-64) . (Con los mismos datos, DIV dio como resultado un cociente de +64.)

```
TITLE   P13DIV (COM)   Operaciones DIV e IDIV
        .MODEL  SMALL
        .CODE
        ORG     100H
BEGIN:  JMP     SHORT MAIN
;-----------------------------------------------------------------------
BYTE1   DB      80H                 /Datos
BYTE3   DB      16H
WORD1   DW      2000H
WORD2   DW      0010H
WORD3   DW      1000H
;-----------------------------------------------------------------------
MAIN    PROC  NEAR                  |Procedimiento principal
        CALL    D10DIV              |Llama a la rutina DIV
        CALL    E10IDIV             |Llama a la rutina IDIV
        MOV     AX,4C00H            |Sale al DOS
        INT     21H
MAIN    ENDP
                Ejemplos de DIV:
;-----------------------------------
D10DIV  PROC
        MOV     AX,WORDl            |Palabra / byte
        DIV     BYTE1               |  residuo:cociente en AH AL
        MOV     AL,BYTE1            |Byte / byte
        SUB     AH, AH              |  extiende el dividendo en DX:AX
        DIV     BYTE3               |  residuo cociente en AH:AL

        MOV     DX,WORD2            |Palabra doble / palabra
        MOV     AX,WORD3            |  dividendo en DX:AX
        DIV     WORD1               |  residuo:cociente en DX AX
        MOV     AX,WORDl            |  Palabra / palabra
        SUB     DX,DX               |  extiende el dividendo en DX
        DIV     WORD3               |  residuo:cociente en DX AX
        RET
D10DIV  ENDP
;       Ejemplos de IDIV:
;-----------------------------------
E10IDIV PROC
        MOV     AX.WORD1            |Palabra / byte
        IDIV    BYTE1               |  residuo:cociente en AH AL
        MOV     AL,BYTE1            |Byte / byte
        CBW                         |  extiende el dividendo en AH
        IDIV    BYTE3               |  residuo:cociente en AH AL

        MOV     DX,WORD2            |•Palabra doble / palabra
        MOV     AX,WORD3            |  dividendo en DX:AX
        IDIV    WORD1               |• • residuo:cociente en DX:AX
        MOV     AX.WORDl            |
        CWD                         |•Palabra / palabra
        IDIV    WORD3               |  extiende el dividendo en DX
        RET                         |/  residuo:cociente en DX:AX
E10IDIV ENDP
        END     BEGIN
```

Figura 13-5 División con signo y sin signo

Los resultados en hexadecimal de los tres ejemplos restantes de IDIV son:

| EJEMPLO DE IDIV | RESIDUO | COCIENTE |
|---|---|---|
| 2 | EE (-18) | FB (-5) |
| 3 | 1000 (4096) | 0080 (128) |

Sólo el cuarto ejemplo da el mismo resultado que el que dio DIV. En efecto, si el dividendo y el divisor tienen el mismo bit de signo, DIV e IDIV generan el mismo resultado. Pero si el dividendo y el divisor tienen bits de signo diferentes, DIV genera un cociente positivo, mientras que IDIV genera un cociente negativo.

Puede encontrar útil usar DEBUG para rastrear estos ejemplos.

### Desboradamientos e interrupciones

Las operaciones DIV e IDIV suponen que el cociente es mucho menor que el dividendo original. Como consecuencia, la operación puede causar con facilidad un desbordamiento; cuando lo hace, ocurre una interrupción con resultados impredecibles. La división entre cero siempre provoca una interrupción. Pero la división entre uno genera un cociente igual al dividendo y también podría causar una interrupción.

Ésta es una regla útil: si el divisor es un byte, su contenido debe ser mayor que el byte izquierdo (AH) del dividendo; si el divisor es una palabra, su contenido debe ser mayor que la palabra izquierda (DX) del dividendo; si el divisor es una palabra doble, su contenido debe ser mayor que la palabra doble izquierda (EDX) del dividendo. Veamos un ejemplo que utiliza 1 como divisor, aunque también pueden servir otras cifras:

| LA OPERACIÓN DIVIDE | DIVIDENDO | DIVISOR | COCIENTE |
|---|---|---|---|
| Palabra entre byte: | 0123 | 01 | (1)23 |
| Palabra doble entre palabra: | 0001 4026 | 0001 | (1)4026 |

En ambos casos, el cociente generado excedería su espacio disponible. Puede ser prudente incluir una prueba antes de las operaciones DIV o IDIV, como se muestra en los dos ejemplos siguientes. En el primero, DIVBYTE es un divisor de un byte, y el dividendo ya está en el AX:

```
CMP  AH, DIVBYTE                ;Compara el AH con el divisor
JNB  Rutina-desbordamiento      ;Si no es menor salta
DIV  DIVBYTE                    /Divide una palabra entre un byte
```

En el segundo ejemplo, DIVWORD es un divisor de una palabra y el dividendo está en el DX:AX:

```
CMP   DX,DIVWORD                /Compara el DX con el divisor
JNB   Rutina-desbordamiento     /Si no es menor salta
DIV   DIVWORD                   /Divide una palabra DOBLE entre una palabra
```

Para IDIV, la lógica debe tener en cuenta que el dividendo o el divisor pueden ser negativos. Ya que el valor absoluto del divisor debe ser el menor de los dos, puede utilizar la instrucción NEG para convertir temporalmente un número negativo en positivo y después de la división restaurar el signo.

### División por medio de restas

Si un cociente es demasiado grande para el divisor, puede realizar la división por medio de restas sucesivas. Esto es, restar el divisor del dividendo, incrementar en uno el cociente y continuar

restando hasta que el dividendo sea menor que el divisor. En el ejemplo siguiente, el dividendo está en el AX, el divisor está en el BX y el cociente se desarrolla en el CX:

```
        SÜB   CX, CX    ;Inicia el cociente en cero
C20:    CMP   AX, BX    ;Si dividendo < divisor,
        JB    C30       ; entonce salir
        SUB   AX, BX    ,-Restar el divisor del dividendo
        INC   CX        /Sumar uno al cociente
        JMP   C20       /Repetir
C30:    RET             ,-El cociente está en CX, el residuo
```

Al final de la rutina, el CX contiene el cociente y el AX, el residuo. Con toda intención el ejemplo es muy simple para demostrar sólo la técnica. Si el cociente está en la pareja DX:AX, incluya estas dos operaciones:

1. En C20, comparar AX con BX sólo si DX es cero.
2. Después de la instrucción SUB, insertar SBB DX,00.

Observe que un cociente muy grande y un divisor muy pequeño pueden provocar que se realicen miles de ciclos a un gran costo en tiempo de procesamiento.

## DIVISIÓN POR MEDIO DE CORRIMIENTOS

Para la división entre una potencia de dos (2, 4, 8, etcétera), es más eficiente realizar sólo corrimientos a la derecha el número necesario de bits. Para el 8088/8086, un corrimiento mayor que 1 necesita un valor de corrimiento en el registro CL. Los ejemplos siguientes suponen que el dividendo está en el AX:

Divide entre 2 (1 corrimiento a la derecha): **SHR AX**,oi

Divide entre 8 (3 corrimientos a la derecha): **MOV CL,03 ,-8088/8086**

**SHR AX,CL**

Divide entre 8 (3 corrimientos a la derecha): **SHR CL,03 ,-80286** y posteriores

### Corrimientos en los registros DX:AX

La rutina siguiente puede ser útil para obtener una división por corrimientos a la derecha en los registros DX:AX. Puede idear un método más eficiente, pero este ejemplo es general para cualquier número de ciclos (y corrimientos) en el CX. Observe que un bit 1 desplazado fuera del registro entra en la bandera de acarreo, la cual es utilizada por RCR:

```
        MOV   CX.04     Inicializa para cuatro ciclos
D20:    SAR   DX,01     Corrimiento del DX
        RCR   AX.01     Rota el AX a la derecha
        LOOP  D20       Repite
```

## CAMBIO (INVERSIÓN) DEL SIGNO

La operación NEG (negar) invierte el signo de un número binario, de positivo a negativo y viceversa. En realidad, NEG invierte los bits, igual que NOT, y después suma 1 para una correcta notación en complemento a dos. El formato general para NEG es:

```
[etiqueta:]  NEG  {registro/memoria}
```

Veamos algunos ejemplos:

```
NEG   AX        15 bits
NEG   BL        8 bits
NEG   BINAMT    Byte o palabra en memoria
NEG   ECX       32 bits
```

Invertir el signo de un número de 32 (o más) bits implica más pasos. Suponga que el par DX:AX contiene un número binario de 32 bits. NEG no puede actuar sobre el par DX:AX de manera concurrente, y usarla en ambos registros significaría sumar 1 a ambos. En lugar de eso, utilice NOT para cambiar los bits, utilice ADD y ADC para sumar el uno para el complemento a dos:

```
NOT  DX      Cambia los bits
NOT  AX      Cambia los bits
ADD  AX, 1   Suma 1 al AX
ADC  DX,0    Suma con acarreo al DX
```

Queda un problema menor: todo está muy bien para realizar aritmética con datos binarios que el programa se define o con datos que ya están en forma binaria den un archivo de disco. Sin embargo, los datos que introduce un programa desde una terminal están en formato ASCII. Aunque los datos ASCII son adecuados para desplegar e imprimir información, requieren de un ajuste especial para la aritmética, un tema que se estudia en el capítulo siguiente.

## PROCESADORES NUMÉRICOS DE DATOS (COPROCESADORES)

Esta sección da una introducción general a los procesadores numéricos de datos; un estudio completo queda fuera del alcance de este libro. La tarjeta de sistema tiene un enchufe para un Procesador Numérico de Datos de Intel, conocido como coprocesador. El coprocesador 8087 opera en conjunción con un 8088/86, el 80287 con un 80286, el 80387 con un 80386, y así sucesivamente.

El coprocesador tiene su característico conjunto de instrucciones y hardware para punto flotante a fin de realizar operaciones como exponenciaciones y operaciones logarítmicas y trigonométricas. Los ocho registros de 80 bits de punto flotante puede representar números hasta 10 elevado al exponente 400, es decir, $10^{400}$. El procesamiento matemático del coprocesador es alrededor de 100 veces más rápido que el procesador normal.

El 8087 consta de ocho registros de 80 bits, R1-R8, en el formato siguiente:

| s | exponente | | mantisa | |
|---|---|---|---|---|
| 79 | 78 | **64** | **63** | **0** |

Cada registro tiene asociado un indicador de 2 bits, que indica su estado:

00 Contiene un número válido
01 Contiene un valor cero
10 Contiene un número no válido
11 Está vacío

El coprocesador reconoce siete tipos de datos numéricos:

**1.** *Word integer (palabra):* 16 bits de datos binarios.

| **s** | **número** | |
|---|---|---|
| **15** | **14** | **0** |

**2.** *Short integer (entero corto):* 32 bits de datos binarios.

| **s** | **número** | |
|---|---|---|
| **31** | **30** | **0** |

3. *Long integer (entero largo):* 64 bits de datos binarios.

| **s** | **número** | |
|---|---|---|
| **63** | **62** | **0** |

**4.** *Short real (real corto):* 32 bits de datos de punto flotante.

| s | exponente | | mantisa | |
|---|---|---|---|---|
| **31** | **30** | **23** | **22** | **0** |

5. *Long real (real largo):* 64 bits de datos de punto flotante.

| **s** | **exponente** | | **mantisa** | |
|---|---|---|---|---|
| **63** | **62** | **52** | **51** | **0** |

**6.** *Temporary real (real temporal):* 80 bits de datos de punto flotante.

| **s** | **exponente** | | **mantisa** | |
|---|---|---|---|---|
| 79 | 78 | **64** | **63** | **0** |

7. *Packed decimal (decimal empacado):* 18 dígitos decimal significativos.

| s | ceros | mantisa |
|---|---|---|
| 79 | 78 72 | 71 0 |

Los tipos 1, 2 y 3 son los formatos comunes de binarios en complemento a dos. Los tipos 4, 5 y 6 representan números de punto flotante. El tipo 7 contiene 18 dígitos decimales de 4 bits cada uno. Puede cargar cualquiera de estos formatos desde memoria a un registro del coprocesador y almacenar el contenido de un registro en la memoria. Sin embargo, el coprocesador convierte para sus cálculos todos los formatos en sus registros a real temporal. Los datos están almacenados en memoria en secuencia inversa de byte.

El procesador solicita una operación específica y envía datos numéricos al coprocesador, que realiza la operación y regresa el resultado. Para ensamblar, utilice la directiva .80x86 apropiada.

La instrucción INT 11H puede ayudar a determinar la presencia de un coprocesador. La operación envía el estado del equipo al AX, en donde un bit en 1 significa que está presente un coprocesador.

## PUNTOS CLAVE

- Los números con signo máximos para acumuladores de un byte son +127 y -128.
- Para sumar en varias palabras múltiples, utilice ADC para tomar en cuenta cualquier acarreo de un ADD anterior. Si la operación se realiza dentro de un ciclo, utilice CLC para inicializar la bandera de acarreo en cero.
- Utilice MUL para datos sin signo e IMUL para datos con signo.
- Con MUL, si un multiplicador está definido como un byte, el multiplicando es AL; si el multiplicador es una palabra, el multiplicando es AX; si el multiplicador es una palabra doble, el multiplicando es EAX.
- Utilice corrimiento a la izquierda (SHL o SAL) para multiplicar por potencias de 2.
- Utilice DIV para datos sin signo e IDIV para datos con signo.
- En la división tenga cuidado especial del desbordamiento. El divisor debe ser mayor que el contenido del AH si el divisor es un byte, que el DX si el divisor es una palabra, o que el EDX si el divisor es una palabra doble.
- Con DIV, si el divisor está definido como un byte, el dividendo es AX; si el divisor es una palabra, el dividendo es DX:AX; si el divisor es una palabra doble, el dividendo es EDX:EAX.
- Utilice corrimiento a la derecha para dividir entre potencias de 2:—SHR para campos sin signo y SAR para campos con signo.

## PREGUNTAS

**13-1.** (a) ¿Cuáles son los números máximos en un byte para datos con signo y para datos sin signo? (b) ¿Cuál es el número máximo en una palabra para datos con signo y sin signo?

**13-2.** Escriba la diferencia entre un acarreo y un desbordamiento.

Las preguntas 13-3 hasta la 13-7 se refieren a los datos siguientes, con palabras definidas en orden inverso:

```
DATAX  DW  0148H
       DW  2316H
DATAY  DW  0237H
       DW  4052H
DATAZ  DW  0
       DW  0
       DW  0
```

13-3. Codifique las instrucciones para sumar lo siguiente: (a) la palabra DATAX a la palabra DATAY; (b) la palabra doble que empieza en DATAX a la palabra doble en DATAY.

13-4. Explique el efecto de las instrucciones siguientes relacionadas:

```
STC
MOV  BX, DATAX
ADC  BX, DATAY
```

13-5. Codifique las instrucciones para multiplicar (MUL) lo siguiente: (a) la palabra DATAX por la palabrz DATAY; (b) la palabra doble que empieza en DATAX por la palabra doble en DATAY. Almacene e producto en DATAZ.

13-6. Además del cero, ¿qué divisores provocan un error por desbordamiento?

13-7. Codifique las instrucciones para dividir (DIV) lo siguiente: (a) la palabra DATAX entre 23; **(b)** *h* palabra doble que empieza en DATAX entre la palabra DATAY.

13-8. Corrija el programa de la figura 13-2 de modo que la rutina sume tres pares de palabras en lugar di dos. Ponga por nombre WORD3 y WORD3B, a las palabras adicionales.

13-9. Refiérase a la sección "Multiplicación por corrimiento". La segunda parte contiene un método má; eficiente de corrimiento a la izquierda de cuatro bits. Corrija el ejemplo para un corrimiento a 1¿ derecha de cuatro bits.

CAPÍTULO 14

# Aritmética: II—Procesamiento de datos ASCII y BCD

OBJETIVO

Examinar los formatos de datos ASCII y BCD para realizar aritmética, y estudiar las conversiones entre estos formatos y el binario.

## INTRODUCCIÓN

En las computadoras el formato natural para la aritmética es el binario. Como se vio en el capítulo 13, el formato binario no causa mayores problemas, siempre y cuando el programa defina sus datos. Sin embargo, para muchos propósitos, los datos numéricos se introducen desde el teclado como caracteres ASCII, en formato de base 10. De manera similar, el despliegue de valores numéricos en la pantalla es en formato ASCII.

Un formato relacionado, *decimal codificado en binario (BCD),* tiene uso ocasional y aparece como desempaquetado y empaquetado. La PC proporciona varias instrucciones que facilitan la aritmética sencilla y la conversión entre formatos. Este capítulo también trata las técnicas para la conversión de datos ASCII a binario para aplicar la aritmética, así como las técnicas para convertir los resultados binarios de regreso a formato ASCII para su visualización. El programa final del capítulo combina mucho del material que se ha estudiado en los capítulos 1 a 13.

Si ha programado en un lenguaje de alto nivel, como C, usted está acostumbrado a que el compilador tome en cuenta el punto base (decimal o binario). Sin embargo, la computadora no reconoce un punto base en un campo aritmético, así que como programador tiene que tener en cuenta su posición.

Las instrucciones introducidas en este capítulo son:

AAA Ajusta ASCII después de sumar
AAS Ajusta ASCII después de restar
AAM Ajusta ASCII después de multiplicar
AAD Ajusta ASCII para dividir
DAA Ajusta decimal después de sumar
DAS Ajusta decimal después de restar

## DATOS EN FORMATO DECIMAL

Hasta este punto, hemos manejado valores numéricos en formatos binario y ASCII. El sistema de la PC también permite usar formato *decimal codificado en binario (BCD),* que facilita algunas operaciones aritméticas limitadas. Dos usos del formato BCD son:

1. EL BCD permite un redondeo apropiado de números sin pérdida de precisión, una característica que es particularmente útil para manejo de cantidades monetarias (pesos y centavos). (El redondeo de números binarios que representan pesos y centavos puede provocar una pérdida en la precisión.)
2. Con frecuencia es más sencillo realizar aritmética con números pequeños introducidos desde el teclado o que son escritos en la pantalla o en la impresora.

Un dígito BCD consiste en cuatro bits que pueden representar los dígitos decimales desde el 0 hasta el 9:

| Binario | Dígito BCD | Binario | Dígito BCD |
|---|---|---|---|
| 0000 | 0 | 0101 | 5 |
| 0001 | 1 | 0110 | 6 |
| 0010 | 2 | 0111 | 7 |
| 0011 | 3 | 1000 | 8 |
| 0100 | 4 | 1001 | 9 |

Puede almacenar dígitos BCD como desempaquetado o empaquetado:

1. *BCD desempaquetado* tiene un solo dígito BCD en los cuatro bits inferiores de cada byte, con ceros en los cuatro bits superiores. Observe que aunque el formato ASCII también es "desempaquetado" no se le llama así.
2. *BCD empaquetado* contiene dos dígitos BCD, uno en los cuatro bits superiores y uno en los cuatro bits inferiores. Este formato es muy común para la aritmética que utiliza coprocesador numérico, definido como 10 bytes con la directiva DT.

Examinemos la representación del número decimal 1,527 en los tres formatos decimales:

- ASCII 31 35 32 37 (cuatro bytes)
- BCD desempaquetado 01 05 02 07 (cuatro bytes)
- BCD empaquetado 15 27 (dos bytes)

El procesador realiza aritmética en valores ASCII y BCD un dígito a la vez. Usted tiene que usar instrucciones especiales para convertir de un formato al otro.

## PROCESAMIENTO DE DATOS ASCII

Ya que los datos que usted ingresó desde un teclado están en formato ASCII, la representación en memoria de un número decimal ingresado tal como 1234 es 31323334H. Pero realizar aritmética sobre tal número implica un tratamiento especial. Las instrucciones AAA y AAS realizan aritmética de manera directa sobre números ASCII:

| | | |
|---|---|---|
| [etiqueta:] | AAA | /Ajusta ASCII después de sumar |
| [etiqueta:] | AAS | /Ajusta ASCII después de restar |

Estas instrucciones están codificadas sin operandos y ajustan de manera automática un valor ASCII que se encuentre en el registro AX. El ajuste ocurre porque un número ASCII representa un número de base 10 desempaquetado, mientras que el procesador realiza aritmética en base dos.

### Suma ASCII

Considere el efecto de sumar los números ASCII 8 (38H) y 4 (34H):

```
38  hex
34  hex
6C  hex
```

La suma 6CH no es correcta ni en ASCII ni en binario. Sin embargo, ignore el 6 de la extrema izquierda, y sume 6 al C hex: C más 6 hex = 12 hex, la respuesta correcta en términos de números decimales. ¿Por qué añadir 6? Porque ésa es la diferencia entre hexadecimal (16) y decimal (10). Esto es muy simple, pero indica la forma en la que AAA realiza su ajuste.

La operación AAA verifica el dígito hex en la extrema derecha (cuatro bits) del registro AL. Si el dígito está entre A y F o la bandera de acarreo auxiliar es 1, la operación suma 6 al registro AL, suma 1 al registro AH y pone en uno las banderas de acarreo y acarreo auxiliar. En todos los casos, AAA pone en cero el dígito hexadecimal en la extrema izquierda del AL.

Como ejemplo, suponga que el AX contiene 0038H y el BX contiene 0034H. El 38 en el AL y el 34 en el BL representan dos bytes ASCII que serán sumados. La suma y el ajuste son como sigue:

```
ADD  AL,BL          /Suma 34H a 38H,  igual a 006CH
AAA                 /Ajusta para suma ASCII,  igual a 0102H
```

Ya que el dígito hexadecimal en la extrema derecha del AL es C, AAA suma 6 al AL, suma 1 al AH, pone en uno las banderas de acarreo y de acarreo auxiliar y pone en cero el dígito hexadecimal en la extrema izquierda del AL. El resultado en el AX ahora es 0102H.

Para restaurar la representación ASCII, sólo inserte 3 en los dígitos hexadecimal en la extrema izquierda del AH y del AL para obtener 3132H o 12 decimal:

```
OR  AX.3030H  /El resultado ahora es 3132H
```

Todo esto está muy bien para sumar números ASCII de un byte. Sin embargo sumar números ASCII de varios bytes necesita un ciclo que procese de derecha a izquierda (de orden bajo a

```
TITLE   P14ASCAD (COM)   Suma de números ASCII
        .MODEL SMALL
        .CODE
        ORG     100H
BEGIN:  JMP     SHORT MAIN

ASCI    DB      '578'               ;Datos
ASC2    DB      '694'
ASCSUM  DB      '0000'

MAIN    PROC    NEAR
        CLC                         |Limpia bandera de acarreo
        LEA     SI,ASCl+2           |Inicialización de
        LEA     DI,ASC2+2           |  números ASCII
        LEA     BX, ASCSUM+3
        MOV     CX, 03              ;Inicialización de 3 ciclos
A20:
        MOV     AH, 00              |Limpia el AH
        MOV     AL, [SI]            |Carga un byte ASCII
        ADC     AL, [DI]            |Suma (con acarreo)
        AAA                         |Ajusta para ASCII
        MOV     [BX] , AL           |Almacena la suma
        DEC     SI
        DEC     DI
        DEC     BX
        LOOP    A20                 Realiza el ciclo 3 veces
        MOV     [BX], AH            ,-Al final, almacena el acarreo

        LEA     BX,ASCSUM+3         ;Convierte ASCSUM
        MOV     CX, 04                a ASCII
A30:
        OR      BYTE PTR[BX] ,30H
        DEC     BX
        LOOP    A30                 ,-Realiza el ciclo 4 veces
        MOV     AX,4C00H            ;Sale al DOS
        INT     21H
MAIN    ENDP
        END     BEGIN
```

Figura 14-1 Suma ASCII

alto) y tome en cuenta los acarreos. El código en la figura 14-1 suma dos números ASCII de tres bytes cada uno, ASCI y ASC2, y produce una suma de cuatro bytes, ASCSUM. Observe los puntos siguientes:

- Una instrucción CLC al empezar inicializa la bandera de acarreo en cero.
- A continuación en A20, ADC es utilizada para sumar ya que un ADD puede provocar un acarreo que debe ser añadido al siguiente byte (de la izquierda).
- Una instrucción MOV limpia el AH en cada ciclo, ya que cada AAA puede sumar uno al AH. Sin embargo, ADC toma en cuenta cualquier acarreo. Note que el uso de XOR o SUB para limpiar el AH cambiaría la bandera de acarreo.
- Cuando el ciclo se ha completado, la rutina mueve el AH (que contiene un 00 final o 01) al byte en la extrema izquierda de ASCSUM.
- Al final, ASCSUM contiene 01020702H. Para insertar el 3 ASCII en cada byte, el programa pasa a través de ASCSUM en memoria y realiza un OR en cada byte con 30H. El resultado es 31323732H o 1272 decimal.

La rutina no utilizó OR después de AAA para insertar los 3 de más a la izquierda, ya que OR pone en uno la bandera de acarreo y cambia el resultado para las instrucciones ADC. Una solución que guarda la configuración de las banderas es enviarla (PUSHF) al registro de banderas, ejecutar el OR y después sacar (POPF) las banderas para restaurarlas:

```
ADC      AL,[DI]       ;Suma con acarreo
AAA                    /Ajusta para ASCII
PUSHF                  ,-Guarda las banderas
OR       AL.30H        ;Inserta el 3 ASCII
POPF                   ;Restaura las banderas
MOV      [BX] ,AL      ,-Almacena la suma
```

### Resta ASCII

La instrucción AAS funciona igual que AAA. El AAS verifica el dígito hexadecimal (cuatro bits) de más a la derecha del AL. Si el dígito está entre A y F o la bandera auxiliar de acarreo está en uno, la operación resta 6 del AL, resta uno del AH y pone en uno las banderas auxiliar (AF) y de acarreo (CF). En todos los casos, AAS pone en cero el dígito de más a la izquierda del AL.

Los dos ejemplos siguientes suponen que ASCI contiene 38H y ASC2 contiene 34H. El primer ejemplo resta ASC2 (34H) de ASCI (38H). AAS no necesita hacer un ajuste, ya que el dígito de la derecha es menor que A:

```
                      AX      AF
MOV   AL,ASCI   ,-003 8
SUB   AL,ASC2   ;0004   0
AAS             ,-0004  0
OR    AL,30H    ;0034
```

El segundo ejemplo resta ASCI (38H) de ASC2 (34H). Como el dígito de más a la derecha es C hex, AAS resta 6 del AL, resta uno del AH y pone en uno las banderas AF y CF. La respuesta, que debe ser -4, es FF06H, su complemento a 10, que tiene valor pequeño:

```
                      AX      AF
MOV   AL,ASC2   ,-0034
SUB   AL,ASCI   ;00FC   1
AAS             ;FF0  6 1
```

## PROCESAMIENTO DE DATOS BCD DESEMPAQUETADOS

La multiplicación y división de números ASCII necesita que primero los números sean convertidos al formato BCD desempaquetado. Las instrucciones AAM y AAD realizan aritmética de forma directa sobre números BCD desempaquetados:

```
[etiqueta:]  AAM  /Ajusta ASCII después de multiplicar
[etiqueta:]  AAD  /Ajusta ASCII antes de dividir
```

### Multiplicación ASCII

La instrucción AAM corrige el resultado de la multiplicación de datos ASCII en el registro AX. Sin embargo, usted primero debe limpiar el 3, de cada byte, en el dígito hexadecimal de más a la izquierda, así se convierte el valor en BCD desempaquetado. Por ejemplo, el número ASCII 31323334 se convierte en 01020304 como BCD desempaquetado. También, ya que el ajuste no es sino de un byte a la vez, sólo puede multiplicar campos de un byte y tiene que realizar la operación de forma repetida en un ciclo. Sólo utilice la operación MUL, no la operación IMUL.

AAM divide el AL entre 10 (OAH) y almacena el cociente en el AH y el residuo en el AL. Por ejemplo, suponga que el AL contiene 35H y el CL contiene 39H. El código siguiente multiplica el contenido del AL por el de CL y convierte el resultado a formato ASCII:

```
INSTRUCCIÓN          COMENTARIO                      AX     CL
AND  CL.0FH     /Convierte CL a 09                    0035   09
AND  AL, OFH    /Convierte AL a 05                    0005
MUL  CL         /Multiplica AL por CL                 002D
AAM             /Convierte a BCD desempaquetado       0405
OR   AX.3030H   /Convierte a ASCII                    3435
```

La operación MUL genera 45 (002DH) en el AX. AAM divide este número entre 10, generando un cociente de 04 en el AH y un residuo de 05 en el AL. Después, la instrucción OR convierte el valor BCD desempaquetado a formato ASCII.

La figura 14-2 describe la multiplicación de un multiplicando de cuatro bytes por un multiplicador de un byte. Ya que AAM tiene capacidad para operaciones con un byte, la rutina pasa por el multiplicando un byte a la vez, de derecha a izquierda. Al final, el producto BCD desempaquetado es 0108090105, que un ciclo convierte a un formato real ASCII como 3138393135, o 18,915 decimal.

Si el multiplicador es mayor que un byte, tiene que proporcionar otro ciclo más que pase por el multiplicador. Puede ser más sencillo convertir el dato ASCII a formato binario, como se estudia en una sección posterior.

### División ASCII

La instrucción AAD proporciona una corrección de un dividendo ASCII antes de hacer la división. Igual que con AAM, primero usted debe limpiar los 3 de la izquierda de los bytes ASCII para crear un formato BCD desempaquetado. ADD permite un dividendo de dos bytes en el AX. El divisor sólo puede ser un único byte con 01 a 09.

Suponga que el AX contiene el valor ASCII 28 (3238H) y el CL contiene al divisor, 7 ASCII (37H). Las instrucciones siguientes realizan el ajuste y la división:

```
TITLE   P14ASCMU (COM)  Multiplicación de números ASCII
        .MODEL SMALL
        .CODE
        ORG     100H
BEGIN:  JMP     MAIN

MULTCND DB      '3783'              Datos
MULTPLR DB      '5'
PRODUCT DB      5 DUP(0)

MAIN    PROC    NEAR
        MOV     CX, 04              Inicializa 4 ciclos
        LEA     SI,MULTCND+3
        LEA     DI,PRODUCT+4
        AND     MULTPLR, OFH        /Limpia el 3 ASCII
A20:
        MOV     AL, [SI]            Carga el carácter ASCII
        AND     AL,OFH              Limpia el 3 ASCII
        MUL     MULTPLR             Multiplica
        AAM                         Ajusta para ASCII
        ADD     AL, [DI]            Suma para
        AAA                           almacenar
        MOV     [DI], AL              el producto
        DEC     DI
        MOV     [DI], AH            Almacena el producto con acarreo
        DEC     SI
        LOOP    A20                 /Realiza el ciclo 4 veces

        LEA     BX,PRODUCT+4        •Convierte PRODUCT
        MOV     CX, 05              •  a ASCII
A30:
        OR      BYTE PTR[BX], 3OH
        DEC     BX
        LOOP    A30                 /Realiza el ciclo 4 veces
        MOV     AX,4C00H            /Sale al DOS
        INT     21H
MAIN    ENDP
        END     BEGIN
```

Figura **14-2** Multiplicación ASCII

| INSTRUCCIÓN | COMENTARIO | AX | CL |
|---|---|---|---|
| AND CL,0FH | Convierte a BCD desempaquetado | 3238 | 07 |
| AND AX,0F0FH | Convierte a BCD desempaquetado | 0208 | |
| AAD | Convierte a binario | 001C | |
| DIV CL | Divide entre 7 | 0004 | |

AAD multiplica el AH por 10 (OAH), suma el producto 20 (14H) al AL y limpia el AH. El resultado, 001CH, es la representación hexadecimal del 28 decimal.

La figura 14-3 permite hacer la división entre un divisor de un byte y un dividendo de cuatro bytes. La rutina pasa por el dividendo de izquierda a derecha. LODSB obtiene un byte de DIVDND para el AL (vía el SI) y STOSB almacena bytes del AL en QUOTNT (vía el DI). El residuo permanece en el registro AH de modo que AAD lo ajustará en el AL. Al final, el cociente, en formato BCD desempaquetado, es 00090204 y el residuo en el AH es 02. Otro ciclo (no codificado) podría convertir el cociente a formato ASCII como 30393234.

Si el divisor es mayor de un byte, usted tiene que proporcionar otro ciclo más para pasar por el divisor. Mejor aún, vea la sección posterior, "Conversión de formato ASCII a binario".

```
TITLE     P14ASCDV (COM)   División de números ASCII
          .MODEL SMALL
          .CODE
          ORG    100H
 BEGIN:   JMP    SHORT MAIN

 DIVDND   DB     '3698'              /Datos
 DIVSOR   DB     '4'
 QUOTNT   DB     4 DUP(0)

 MAIN     PROC   NEAR
          MOV    CX,04               Inicialización para 4 ciclos
          SUB    AH, AH              Limpia el byte izquierdo del dividendo
          AND    DIVSOR,0FH          Limpia el divisor del 3 ASCII
          LEA    SI,DIVDND
          LEA    DI,QUOTNT
 A20:
          LODSB                      Carga el byte ASCII/
          AND    AL,0FH              Limpia el 3 ASCII
          AAD                        Ajusta para dividir
          DIV    DIVSOR              Divide
          STOSB                      Almacena el cociente
          LOOP   A20                 ¿Ya son cuatro veces?
          INT    21H                   sí, entonces salir al DOS
 MAIN     ENDP
          END    BEGIN
```

Figura 14-3 División ASCII

## PROCESAMIENTO DE DATOS BCD EMPAQUETADOS

En el ejemplo precedente de división ASCII, el cociente fue 00090204. Si tuviera que condensar este valor conservando el dígito derecho de cada byte, el resultado sería 0924, ahora en formato BCD empaquetado. También puede realizar sumas y restas sobre datos BCD empaquetados. Para este objetivo, existen dos instrucciones de ajuste:

```
[etiqueta:]  DAA   /Ajuste decimal después de la suma
[etiqueta:]  DAS   /Ajusté decimal después de la resta
```

DAA corrige el resultado de la suma de dos números BCD empaquetados en el AL, y DAS corrige el resultado de su resta. Una vez más, tiene que procesar los campos un byte a la vez.

El programa en la figura 14-4 ejemplifica la suma BCD. El procedimiento B10CONV convierte los números ASCII ASCI y ASC2 a números BCD empaquetados BCD1 y BCD2, respectivamente. El procesamiento, que es de derecha a izquierda, podría ser tan fácil de izquierda a derecha. También el procesamientotle palabras es más fácil que el procesamiento de bytes, ya que necesita dos bytes ASCII para generar un byte BCD empaquetado. Sin embargo, el uso de palabras requiere de un número par de bytes en el campo ASCII.

El procedimiento C10ADD realiza un ciclo tres veces para sumar los números BCD empaquetados a BCDSUM. El total final es 00127263.

```
TITLE     P14BCDAD (COM)   Conversión de ASCII a BCD y suma
          .MODEL  SMALL
          .CODE
          ORG     100H
BEGIN:    JMP     SHORT MAIN
-------------------------------------------------------------
ASCI      DB      '057836'          ;DatOS
ASC2      DB      '069427'
BCD1      DB      '000'
BCD2      DB      •000'
BCDSUM    DB      4 DUP(0)
-------------------------------------------------------------
MAIN      PROC    NEAR
          LEA     SI,ASCl+4         ;Inicializa ASCI
          LEA     DI,BCDl+2
          CALL    B10CONV           ;Llama la rutina para convert
          LEA     SI,ASC2+4          Inicializa ASC2
          LEA     DI,BCD2+2
          CALL    B10CONV           ;Llama la rutina para convert
          CALL    C10ADD            ,-Llama la rutina para sumar
          MOV     AX,4C00H          ;Sale al DOS
          INT     21H
MAIN      ENDP
          Convierte ASCII a BCD
-------------------------------------------------------------
B10CONV   PROC
          MOV     CL, 04            ,-Factor de corrimiento
          MOV     DX, 03            /Núm. de palabras a convertir
B20:
          MOV     AX, [SI]          /Obtiene la pareja ASCII
          XCHG    AH, AL
          SHL     AL, CL            /Corrimiento de
          SHL     AX, CL            /  3 ASCII
          MOV     [DI], AH          /Almacena los dígitos BCD
          DEC     SI
          DEC     SI
          DEC     DI
          DEC     DX
          JNZ     B20               /¿Son tres veces?
          RET                       /  sí, entonces regresar
B10CONV   ENDP
          Suma de números BCD:
-------------------------------------------------------------
¿10ADD    PROC
          XOR     AH, AH             /Limpia el AH
          LEA     SI,BCD1+2          ,• Inicializa
          LEA     DI.BCD2+2          /  direcciones de
          LEA     BX,BCDSUM+3        /  BCD
          MOV     CX, 03             /campos de 3 bytes
          CLC
C20:
          MOV     AL, [SI]           Obtiene BCD1  (o LODSB)
          ADC     AL, [DI]           Suma BCD2
          DAA                        Ajusta el decimal
          MOV     [BX], AL           Almacena en BCDSUM
          DEC     SI
          DEC     DI
          DEC     BX
          LOOP    C2 0               /Realiza el ciclo 3 veces
          RET
C10ADD    ENDP
          END     BEGIN
```

Figura 14-4 Conversión y aritmética en BCD

## CONVERSIÓN DE FORMATO ASCII A BINARIO

Realizar aritmética en formato ASCII o BCD sólo es adecuado para campos pequeños. Para muchos propósitos aritméticos, es más práctico convertir tales números a formato binario. De hecho, es más fácil convertir desde ASCII a binario, de manera directa, que convertir de ASCII a BCD y luego a binario.

El método de conversión está basado en el hecho de que un número ASCII está en base 10 y la computadora realiza la aritmética en base 2. Aquí está el procedimiento:

1. Inicie con el byte de más a la derecha del número ASCII y procese de derecha a izquierda.
2. Quite el 3 del dígito hexadecimal de la izquierda de cada byte ASCII, para formar un número BCD empaquetado.
3. Multiplique el primer dígito BCD por 1, el segundo por 10 (OAH), el tercero por 100 (64H) y así sucesivamente, y sume los productos.

El ejemplo siguiente convierte el número ASCII 1234 a binario:

| | Decimal | | Hexadecimal | |
|---|---|---|---|---|
| | Paso | Producto | Paso | Producto |
| | 4 x 1 = | 4 | 4 x 01H = | 4H |
| | 3 x 10 = | 30 | 3 x OAH = | 1EH |
| | 2 x 100 = | 200 | 2 x 64H = | C8H |
| | 1 x 1000 = | 1000 | 1 x 3E8H = | 3E8H |
| Total: | | 1234 | 04D2H | |

Verifique que la suma 04D2H sea en realidad igual a 1234 decimal. En la figura 14-5, el programa convierte el número ASCII 1234 a su equivalente binario. Una instrucción LEA inicializa la dirección del byte más a la derecha del campo ASCII, ASCVAL+3, en el registro SI. La instrucción en B20 que mueve el byte ASCII al AL es

```
MOV AL, [SI]
```

La operación utiliza la dirección de ASCVAL+3 para copiar el byte de la extrema derecha de ASCVAL en el AL. Cada iteración del ciclo disminuye en uno el SI y se refiere al siguiente byte a la izquierda. El ciclo se repite para cada uno de los cuatro bytes de ASCVAL. Además cada iteración multiplica MULT10 por 10 (OAH), dando los multiplicadores 1, 10 (OAH), 100 (64H) y así sucesivamente. Al final, BINVAL contiene el número binario correcto, D204H, en secuencia inversa de byte.

La rutina está codificada en términos de claridad; para un procesamiento más rápido, el multiplicador puede ser almacenado en el registro DI.

## CONVERSIÓN DE FORMATO BINARIO A ASCII

Para imprimir o desplegar el resultado de aritmética binaria, tiene que convertirlo en formato ASCII. La operación implica el inverso de los pasos anteriores: En lugar de multiplicar, se debe dividir de manera continua entre 10 (OAH) hasta que el cociente sea menor que 10. Los residuos,

```
TITLE   P14ASCBI (COM)  Conversión de formato ASCII a binario
        .MODEL SMALL
        .CODE
        ORG     100H
BEGIN:  JMP     SHORT MAIN

ASCVAL  DB      '1234'              •Datos
BINVAL  DW      0
ASCLEN  DW      4
MULT10  DW      1

MAIN    PROC    NEAR                Procedimiento principal
        MOV     BX,10               Factor de multiplicación
        MOV     CX,04               Contador del ciclo
        LEA     SI,ASCVAL+3         Dirección de ASCVAL
B20:
        MOV     AL,[SI]             Selecciona el carácter ASCII
        AND     AX,000FH            Borra la zona 3
        MUL     MULT10              •Multiplica por un factor 10
        ADD     BINVAL,AX           •Suma al binario
        MOV     AX.MULT10           •Calcula el siguiente
        MUL     BX                    factor de 10
        MOV     MULT10,AX
        DEC     SI                  ¿Es el último carácter ASCII?
        LOOP    B20                  no, entonces continuar
        MOV     AX,4C0OH
        INT     21H                 •Salir al DOS
MAIN    ENDP
        END     BEGIN
```

Figura **14-5** Conversión de formato ASCII a binario

que sólo puede ser del 0 al 9, generan de manera sucesiva el número ASCII. Como un ejemplo, convierta 4D2H de regreso a formato decimal:

| DIVIDE ENTRE 10 | COCIENTE | RESIDUO |
|---|---|---|
| A ! 4D2 | 7B | 4 |
| A ¡7B~~ | C | 3 |
| A fC | 1 | 2 |

Como el cociente (1) ahora es menor que el divisor (OAH) la operación está terminada. Los residuos, junto con el último cociente, forman el resultado BCD, de derecha a izquierda: 1234. Todo lo que resta por hacer es almacenar estos dígitos en memoria, con los 3 ASCII, como 31323334.

El programa de la figura 14-6 convierte el número binario 04D2H a formato ASCII. La rutina divide el número binario de manera sucesiva entre 10, hasta que el cociente que queda sea menor que 10 (OAH) y almacena los dígitos hexadecimales generados en formato ASCII como 31323334. Si no completamente divertido, puede encontrar útil reproducir este programa y rastrear su ejecución paso por paso.

## CORRIMIENTO Y REDONDEO

Suponga que usted está redondeando a dos decimales un producto que contiene tres posiciones decimales. Si el producto es 12.345, sume 5 a la posición decimal de más a la derecha y recorra un dígito a la derecha:

```
TITLE   P14BINAS (COM)   Conversión de formato binario a ASCII
        .MODEL  SMALL
        .CODE
        ORG     100H
BEGIN:  JMP     SHORT MAIN

ASCVAL  DB      4 DUP(' ')          Datos
BINVAL  DW      04D2H

MAIN    PROC    NEAR                Procedimiento principal
        MOV     CX,0010             Factor de división
        LEA     SI,ASCVAL+3         Dirección de ASCVAL
        MOV     AX,BINVAL           Obtiene campo binario
C20:
        CMP     AX, CX              ¿El número es menor a 10?
        JB      C30                   sí, entonces salir
        XOR     DX.DX               Limpiar el cociente superior
        DIV     CX                  Divide entre 10
        OR      DL,30H
        MOV     [SI],DL             •Almacena el carácter ASCII
        DEC     SI
        JMP     C20
C30:
        OR      AL,30H              Almacena el último cociente
        MOV     [SI] ,AL              como carácter ASCII
        MOV     AX.4C00H            Sale al DOS
        INT     21H
MAIN    ENDP
        END     BEGIN
```

Figura 14-6 Conversión de formato binario a ASCII

| | |
|---|---|
| Producto: | 12.345 |
| Sumar 5 : | + 5 |
| Producto redondeado: | 12.350 = 12.35 |

Si el producto es 12.3455, sume 50 y recorra dos dígitos, y si el producto es 12.34555, sume 500 y recorra tres dígitos:

| | |
|---|---|
| 12.3455 | 12.34555 |
| +___50 | + 500 |
| 12.3505 = 12.35 | 12.35055 = 12.35 |

Además, un número con seis lugares decimales necesita sumar 5,000 y recorrer cuatro dígitos, y así sucesivamente. Ahora, ya que por lo regular una computadora procesa datos binarios, 12.345 aparece como 3039H. Sumando 5 a 3039H da 303EH, o 12350 en formato decimal. Hasta ahora todo va bien. Pero, del corrimiento de un dígito binario resulta 181FH, o 6175, de hecho el corrimiento sólo divide entre dos al número. Nosotros necesitamos un corrimiento que sea equivalente a recorrer a la derecha un dígito decimal. Puede realizar este corrimiento dividiendo el valor redondeado entre 10, o A hex: 303E hex dividido entre A hex = 4D3H, o 1235 decimal. La conversión de 4D3H a un número decimal da 1235. Ahora sólo inserte un punto decimal en la posición correcta y puede desplegar un valor redondeado como 12.35.

De esta manera, puede redondear y recorrer cualquier número binario. Para tres lugares decimales, sume 5 y divida entre 10, para cuatro lugares decimales, sume 50 y divida entre 100.

Tal vez haya notado un patrón: el factor de redondeo (5, 50, 500, etc.) siempre es la mitad del factor de corrimiento (10, 100, 1,000, etcétera).

Por supuesto, el punto decimal en un número binario es implicado y en realidad no está presente.

## PROGRAMA PARA CONVERTIR DATOS ASCII

El programa de la figura 14-7 permite a los usuarios ingresar el número de horas trabajadas y el sueldo por hora de los empleados y despliega el salario calculado. Por brevedad, el programa omite algunas verificaciones de error que de otra forma serían incluidas. Los procedimientos son como sigue:

| | |
|---|---|
| B10INPT | Desde el teclado, acepta horas y sueldo por hora en formato ASCII. Estos valores pueden tener punto decimal. |
| D10HOUR | Inicializa la conversión de horas ASCII a binario. |
| E10RATE | Inicializa la conversión del sueldo ASCII a binario |
| F10MULT | Realiza la multiplicación, redondeo y corrimiento. Un salario con cero, uno |

```
          page 60,132
TITLE     P14SCREMP (EXE) Introduzca horas y sueldo, despl
          .MODEL SMALL
          .STACK 64
;-----------------------------------------------------------
          .DATA
LEFCOL    EQU    28                ¡Equivalencia para la p;
RITCOL    EQU    52
TOPROW    EQU    10
BOTROW    EQU    14

HRSPAR    LABEL  BYTE              ,• Lista de parámetros de
MAXHLEN   DB     6                 ; .-_____________________
ACTHLEN   DB     ?
HRSFLD    DB     6 DUP(?)

RATEPAR   LABEL  BYTE              ;Lista de parámetros de
MAXRLEN   DB     6                 ;_____________________
ACTRLEN   DB     ?                 ;
RATEFLD   DB     6 DUP(?)          .

MESSG1    DB     'Horas trabajadas
MESSG2    DB     'Sueldo por hora
MESSG3    DB     'Salario =
ASCWAGE   DB     10 DUP(30H), 13, ;10, '$'
MESSG4    DB     'Presione cualquier tecla para continuar

ADJUST    DW     ?                 ;Datos
BINVAL    DW     00
BINHRS    DW     00
BINRATE   DW     00
COL       DB     00
DECIND    DB     00
MULT10    DW     01
NODEC     DW     00
ROW       DB     00
SHIFT     DW
TENWD     DW     10
;-----------------------------------------------------------
          .CODE
BEGIN     PROC   FAR
          MOV    AX,@data          ,-Inicializa los
          MOV    DS, AX            ,•  registros DS y ES
          MOV    ES, AX
          CALL   Q10SCR            ;Limpia la pantalla
```

Figura 14-7 Despliegue de los salarios de los empleados

```
A20LOOP:
          CALL    Q15WIN              Limpia la ventana
          CALL    Q20CURS             Coloca el cursor
          CALL    B10INPT             Acepta las horas y el sueldo por hora
          CALL    D10HOUR             Convierte las horas a binario
          CALL    ElORATE             Convierte el sueldo a binario
          CALL    F10MULT             Calcula el salario, redondeado
          CALL    G10WAGE             Convierte salario a ASCII
          CALL    K10DISP             Despliega el salario
          CALL    L10PAUS             Pausa para el usuario
          CMP     AL,1BH              ¿Presionó Esc?
          JNE     A20LOOP               no, entonces continuar
                                        sí, entonces fin de la entrada
          CALL    Q10SCR              Limpia la pantalla
          MOV     AX,4C00H            Sale al DOS
          INT     21H
BEGIN     ENDP
          Ingreso de horas y sueldo por hora

 B10INPT   PROC   NEAR
           MOV    ROW,TOPROW+l        ,-Coloca el cursor
           MOV    COL,LEFCOL+3
           CALL   Q2 0CURS
           INC    ROW
           MOV    AH,09H
           LEA    DX,MESSG1           /Indicación del número de horas
           INT    21H
           MOV    AH,OAH
           LEA    DX,HRSPAR           /Acepta el número de horas
           INT    21H
           MOV    COL.LEFCOL+3        /Designa la columna
           CALL   Q20CURS
           INC    ROW
           MOV    AH,09H
           LEA    DX,MESSG2           /Indicación del sueldo por hora
           INT    21H
           MOV    AH,OAH
           LEA    DX,RATEPAR          /Acepta el sueldo por hora
           INT    21H
           RET
 B10INPT   ENDP
           Procesa las horas:

 D10HOUR   PROC   NEAR
           MOV    NODEC,0 0
           MOV    CL,ACTHLEN
           SUB    CH,CH
           LEA    SI,HRSFLD-1         /Designa la posición derecha
           ADD    SI,CX               /  de horas
           CALL   M10ASBI             /Convierte a binario
           MOV    AX,BINVAL
           MOV    BINHRS,AX
           RET
 D10HOUR   ENDP
           Procesa el sueldo por hora:

 E10RATE   PROC   NEAR
           MOV    CL, ACTRLEN
           SUB    CH,CH
           LEA    SI,RATEFLD-1        /Designa la posición derecha
           ADD    SI,CX               /  de sueldo por hora
           CALL   M10ASBI             /Convierte a binario
           MOV    AX,BINVAL
           MOV    BINRATE, AX
           RET
```

Figura 14-7 (continuación)

```
            ENDP
                    Multiplica,  redondea y recorre:
F10MULT     PROC    NEAR
            MOV     CX,05
            LEA     DI.ASCWAGE          /Designa el salario ASCII
            MOV     AX,3030H            ; a los 3 0
            CLD
            REP  STOSW

            MOV     SHIFT,10
            MOV     ADJUST,00
            MOV     CX,NODEC
            CMP     CL,06               ;Si hay más de 6
            JA      F40                 ,- decimales, error
            DEC     CX
            DEC     CX
            JLE     F30                 ,-Si hay 0, 1, 2 decimales, saltar
            MOV     NODEC,02
            MOV     AX,01
F20:
            MUL     TENWD               ;Calcula el factor de corrimiento
            LOOP    F20
            MOV     SHIFT,AX
            SHR     AX,1                ;Calcula el valor redondeado
            MOV     ADJUST,AX
F30:
            MOV     AX,BINHRS
            MUL     BINRATE             ,-Calcula el salario
            ADD     AX,ADJUST           /Redondea el salario
            ADC     DX,00
            CMP     DX,SHIFT            ;¿El producto es muy grande
            JB      F50                 ;  para DIV?
F40:
            SUB     AX,AX
            JMP     F70
F50:
            CMP     ADJUST,00           ¿Se requiere corrimiento?
            JZ      F80                   no, entonces saltar
            DIV     SHIFT               /Corrimiento de salario
F70:        SUB     DX,DX               ,-Limpiar el residuo
F80:        RET
F10MULT     ENDP
                    Conversión a ASCII
G10WAGE     PROC    NEAR
            LEA     SI,ASCWAGE+7        /Fija el punto decimal
            MOV     BYTE PTR[SI],' '
            ADD     SI,NODEC            /Fija la inicial derecha de inici
G30:
            CMP     BYTE PTR[SI],'.'
            JNE     G40                 Si está en la posición dec, enton
            DEC     SI
G40:
            CMP     DX,00               /Si DX:AX < 10,
            JNZ     G50
            CMP     AX,0010             /  operación terminada
            JB      G60
G50:
            DIV     TÉNWD               /El residuo es un dígito ASCII
            OR      DL,30H
            MOV     [SI],DL             /Almacenar el carácter ASCII
            DEC     SI
            SUB     DX.DX               /Limpiar el residuo
            JMP     G30
```

Figura **14-7** (continuación)

```
G60:
          OR      AL,30H              /Almacena el último
          MOV     [SI],AL             /  carácter ASCII
          RET
G10WAGE   ENDP
                  Despliega el salario:

K10DISP   PROC    NEAR
          MOV     COL,LEFCOL+3        /Designa la columna
          CALL    Q20CURS
          MOV     CX,09
          LEA     SI,ASCWAGE
K20:                                  /Elimina los ceros iniciales
          CMP     BYTE PTR[SI],30H
          JNE     K30                 /  cambiándolos por blancos
          MOV     BYTE PTR[SI],20H
          INC     SI
          LOOP    K20
K30:
          MOV     AH,09H              /Petición de despliegue
          'LEA    DX,MESSG3           ;Salario
          INT     21H
          RET
K10DISP   ENDP
                  Pausa para el usuario:

LIOPAUS   PROC    NEAR
          MOV     COL,20              /Coloca el cursor
          MOV     ROW,22
          CALL    Q20CURS
          MOV     AH,09H
          LEA     DX,MESSG4           /Despliega pausa
          INT     21H
          MOV     AH,10H              /Petición de despliegue
          INT     1SH
          RET
          ENDP
                  Convierte ASCII a binario:

M10ASBI   PROC    NEAR
          MOV     MULT10,0001
          MOV     BINVAL,00
          MOV     DECIND,00
          SUB     BX,BX
M20:
          MOV     AL,[SI]             /Obtiene el carácter ASCII
          CMP     AL,'.'              /Si es punto decimal, saltar
          JNE     M40
          MOV     DECIND,01
          JMP     M90
M40:
          AND     AX,000FH
          MUL     MULT10               Multiplica por factor
          ADD     BINVAL,AX            Suma al binario
          MOV     AX.MULT10            Calcula el factor
          MUL     TENWD                  siguiente  10
          MOV     MULT10,AX
          CMP     DECIND,00           ¿Se llegó al punto decimal?
          JNZ     M90
          INC     BX                  /  sí, entonces sumar a la cuenta
M90:
          DEC     SI
          LOOP    M20
          CMP     DECIND,00            Fin del ciclo
          JZ      M100                 ¿Hay algún punto decimal?
          ADD     NODEC,BX               sí, entonces sumar al total
```

Figura 14-7 (continuación)

**Programa para convertir datos ASCII**

```
M100:     RET
M10ASBI   ENDP
                    Recorre toda la pantalla:

Q10SCR    PROC    NEAR
          MOV     AX, OSOOH
          MOV     BH,3 OH             ,-Atributo
          SUB     CX,CX
          MOV     DX,184FH
          INT     10H
          RET
Q10SCR    ENDP
                    Recorre la pantalla de despliegue:

Q15WIN    PROC    NEAR
          MOV     AX,0605H            ;Cinco renglones
          MOV     BH,1SH              ,•Atributo
          MOV     CH,TOPROW
          MOV     CL,LEFCOL
          MOV     DH,BOTROW
          MOV     DL,RITCOL
          INT     10H
          RET
Q15WIN    ENDP
                    Coloca el cursor:

Q2 0CÜRS  PROC    NEAR
          MOV     AH,02H
          SUB     BH, BH
          MOV     DH.ROW              ;Designa el renglón
          MOV     DL,COL              /Designa la columna
          INT     10H
          RET
Q2 0CURS  ENDP
          END     BEGIN
```

Figura **14-7** (continuación)

El ORATE Inicializa la conversión del sueldo ASCII a binario

F10MULT Realiza la multiplicación, redondeo y corrimiento. Un salario con cero, uno o dos lugares decimales no requiere de redondeo o corrimiento.

G10WAGE Inserta el punto decimal, determina la posición de más a la derecha para empezar a almacenar caracteres ASCII y convierte el salario binario a ASCII.

K10DISP Cambia por espacios en blanco los ceros iniciales de salario y lo despliega.

L10PAUS Despliega el salario calculado hasta que el usuario presione una tecla. Presionando Esc se le indica al programa que interrumpa el proceso.

M10ASBI Convierte ASCII a binario (una rutina común para horas y sueldo por hora) y determina el número de decimales en el número ingresado.

Q10SCR Recorre toda la pantalla y la establece a negro sobre cian.

Q15WIN Recorre una ventana en la mitad de la pantalla en donde horas, sueldo por hora y salario son desplegados en café sobre azul.

Limitaciones. Una limitación de este programa es que sólo permite un total de seis lugares decimales en el salario calculado. Otra es la propia magnitud del salario y el hecho de que el corrimiento exige la división entre un múltiplo de 10 y convertir a ASCII implica división entre 10. Si horas y sueldo por hora contienen un total que exceda seis decimales o si el salario excede

a cerca de 655,350, el programa limpia el salario a cero. En la práctica, un programa imprimiría un mensaje de advertencia o tendría subrutinas para superar estas limitaciones.

Verificación de errores. Un programa diseñado para otros usuarios, además del programador, no sólo debe producir mensajes de advertencia, sino que también debe validar las horas y el sueldo por hora. Los únicos caracteres válidos son los números desde el 0 hasta el 9 y un punto decimal. Para cualquier otro carácter, el programa debe mostrar un mensaje y regresar a la petición de entrada. Una instrucción útil para la validación es XLAT, que se estudia en el capítulo 15.

Como práctica, pruebe su programa completamente para todas las posibles condiciones, como valores cero, números en extremo grandes o pequeños y números negativos.

Números negativos. Algunas aplicaciones implican cantidades negativas, en especial para invertir y corregir entradas. Usted puede permitir un signo menos después de un valor, como 12.34-, o precediendo al número como -12.34. El programa entonces puede verificar un signo menos durante la conversión a binario. Por otra parte, puede querer dejar el número binario positivo y sólo establecer un indicador para registrar el hecho de que la cantidad es negativa. Cuando la aritmética ha terminado, el programa, si se requiere, puede insertar un signo menos en el campo ASCII.

Si quiere que el número binario sea negativo, convierta la entrada ASCII a binario de la forma usual. (Véase la sección "Inversión del signo" en el capítulo 13 para cambiar el signo de un campo binario.) Y tenga cuidado al usar IMUL e IDIV para manejar datos con signo. Para redondear cantidades negativas, reste 5 en lugar de sumar 5.

## PUNTOS CLAVE

- Un campo ASCII necesita un byte para cada carácter. Para un campo numérico, la mitad derecha del byte contiene el dígito y la mitad izquierda un 3.
- Cambiando los 3 ASCII a ceros se convierte el campo a formato decimal codificado en binario (BCD) desempacado.
- Comprimir los caracteres ASCII a dos dígitos por byte convierte el campo a dato decimal codificado en binario (BCD) empacado.
- Después de una suma ASCII, utilice AAA para ajustar la respuesta: después de una resta ASCII, utilice AAS para ajustar la respuesta.
- Antes de una multiplicación ASCII, convierta el dividendo y divisor a BCD desempacado poniendo los 3 hex de la izquierda en cero. Después de una multiplicación, emplee AAM para ajustar el producto.
- Antes de una división ASCII, convierta el dividendo y el divisor a BCD desempacado limpiando los 3 hex de la extrema izquierda y use AAD para ajustar el dividendo.
- Para casi todos los propósitos aritméticos, convierta los números ASCII a binario. Cuando convierta de formato ASCII a binario, verifique que los caracteres ASCII sean válidos: de 30 hasta 39, un punto decimal y tal vez un signo menos.

## PREGUNTAS

**14-1.** Suponga que el AX contiene 9 ASCII (0039H) y que el BX contiene 7 ASCII (0037H). Explique los resultados exactos de las operaciones independientes siguientes:

```
(a)  ADD  AX.33H     (b)  ADD  AX,BX
     AAA                  AAA
(C)  SUB  AX,BX      (d)  SUB  AX,0DH
     AAS                  AAS
```

**14-2.** Un campo BCD desempacado llamado UNPAK contiene 01040705H. Codifique un ciclo que haga que su contenido sea el apropiado ASCII 31343735H.

**14**-3. Un campo llamado ASCA contiene el número decimal ASCII 173 y otro campo llamado ASCB contiene el 5 ASCII. Codifique instrucciones para multiplicar los números ASCII juntos y almacenar el producto en ASCPRO.

**14-4.** Utilice los mismos campos de la pregunta 14-3 para dividir ASCA entre ASCB y almacene el cociente en ASCQUO.

**14-5.** Proporcione los cálculos manuales para lo siguiente: (a) Convertir el número decimal ASCII 46328 a binario y mostrar el resultado en formato hexadecimal; (b) convertir el valor hexadecimal de regreso a ASCII.

**14-6.** Codifique y corra un programa que determine el tamaño de la memoria de la computadora (véase la INT 12H en el capítulo 3), convierta el tamaño a formato ASCII y despliegúelo en pantalla como se muestra:

```
El tamaño de la memoria es de nnn bytes
```

## CAPÍTULO 15

# Procesamiento de tablas

OBJETIVO

**Cubrir los requisitos necesarios para definir tablas, realizar búsquedas en tablas y ordenar tablas.**

### INTRODUCCIÓN

Muchas aplicaciones de programas necesitan *tablas* que contengan datos como nombres, descripciones, cantidades y precios. La definición y el uso de tablas requiere mucho de la aplicación de lo que usted ya ha aprendido. Este capítulo empieza por definir algunas tablas convencionales y después trata métodos para buscar en ellas. Las técnicas para esta búsqueda están sujetas a la manera en que las tablas estén definidas, y son posibles muchos otros métodos para definir y buscar además de los dados aquí. Otras características muy usadas son el ordenamiento, que reacomoda la secuencia de datos en la tabla, y el uso de listas ligadas, que utilizan apuntadores para localizar elementos en una tabla.

La única instrucción introducida en este capítulo es XLAT (Traducir).

### DEFINICIÓN DE TABLAS

Para facilitar la búsqueda en ellas, se acomoda la mayoría de las tablas de manera consistente: cada entrada se define con el mismo formato (carácter o numérico), con la misma longitud y en orden ascendente o descendente.

Una tabla que ya se ha usado a lo largo del libro es la pila, que en lo que sigue es una tabla de 64 palabras no inicializadas (el nombre STACK se refiere a la primera palabra de la tabla):

```
STACK DW 64 DUP(?)
```

Las dos tablas siguientes, MONTAB y EMPTAB, inicializan valores de carácter y numéricos, respectivamente. MONTAB define abreviaturas alfabéticas de Tos meses, mientras que EMPTAB define una tabla de números de empleado:

```
MONTAB DB 'Jan', 'Feb', 'Mar',      'Dec'
EMPTAB DB 205, 208, 209, 212, 215, 224, ...
```

Todas las entradas en MONTAB son de tres caracteres, y todas las entradas en EMPTAB son de tres dígitos. Pero observe que el ensamblador convierte los números decimales a formato binario, y si no exceden de 255, almacena cada uno de ellos en un byte.

Una tabla también puede tener una mezcla de valores numéricos y de caracteres, con tal de que estén definidos de manera consistente. En la tabla siguiente de elementos en existencia, cada entrada numérica (número de existencia) es de dos dígitos (un byte) y cada entrada de carácter (descripción de la existencia) es de nueve bytes. Los cuatro puntos que siguen a "Paper" son para mostrar que deben aparecer espacios; esto es, deben teclearse espacios, y no puntos, en la descripción:

```
STOKTBL DB 12, 'Computers',14, 'P a p e r 1 7 , 'Diskettes', ...
```

Por claridad, también puede codificar las entradas de la tabla en líneas separadas:

```
STOKTBL DB 12,  'Computers'
        DB 14,  'Paper. ...'
        DB 17,  'Diskettes'
```

El ejemplo siguiente define una tabla con 50 entradas, cada una inicializada a 20 blancos:

```
STORETAB DB 50 DUP(20 DUP(' '))
```

Un programa podría usar esta tabla para almacenar hasta 50 valores que se hayan generado de manera interna o para almacenar hasta 50 entradas que se lean de un archivo en disco.

## Tablas en disco

En situaciones del mundo real, muchos programas están dirigidos por medio de tablas. Las tablas son almacenadas en archivos en disco, que cualquier número de programas puede leer de ahí a su segmento de datos para procesamiento. La razón de esta práctica es que el contenido de las tablas cambia con el tiempo. Si cada programa define su propia tabla, cualquier cambio requeriría que todos los programas redefinieran las tablas y que se reensamblaran. Con tablas en archivos en disco, sólo necesita cambiar el contenido del archivo. En el capítulo 17 hay un ejemplo de un archivo de tabla.

Ahora examinemos maneras diferentes de utilizar tablas en programas.

## DIRECCIONAMIENTO DD1ECTO DE TABLAS

Suponga que un usuario introduce un mes numérico tal como 03 y que hay programa para convertirlo a formato alfabético —en este caso, March. La rutina para realizar esta conversión pide definir una tabla de meses alfabéticos, todos de igual longitud. La longitud de cada una de las entradas es el del nombre más largo, September:

```
MONTAB  DB    'January..'
        DB    'February.'
        DB    "March...."


        DB    'December.'
```

La entrada 'January' está en MONTAB+00, 'February' está en MONTAB+09, 'March' en MONTAB+ 18, y así sucesivamente. Para localizar el mes 03, el programa tiene que realizar las acciones siguientes:

1. Convertir el mes ingresado de ASCII 33 a binario 3.
2. Descontar 1 de este número: 3 - 1 = 2 (ya que el mes 01 está en MONTAB+00).
3. Multiplicar el nuevo número por 9 (la longitud de cada entrada): 2 x 9 = 1 8 .
4. Sumar este producto a la dirección de MONTAB; el resultado es la dirección de la descripción requerida: MONTAB + 18, en donde empieza "March".

Esta técnica es conocida como *direccionamiento directo de tabla.* Como el algoritmo calcula de forma directa la dirección de la tabla que se necesita, el programa no tiene que buscar de forma sucesiva en cada entrada de la tabla.

### Direccionamiento directo, ejemplo 1: Tabla de meses

El programa de la figura 15-1 proporciona un ejemplo de acceso directo a una tabla con los nombres de los meses. El procedimiento C10CONV utiliza 12 (December) como entrada y convierte el mes así (los números están en hexadecimal):

| | | |
|---|---|---|
| Carga el mes ASCII en AX: | 3132 | |
| Utiliza 3030 para XOR: | 3030 | |
| Desempaqueta el mes: | 0102 | |
| Si el byte de más a la izquierda nó es cero, | 0002 | |
| limpiar y sumar OAH (10 decimal) | OOOC | (12 decimal) |

El procedimiento D10LOC determina la posición actual de las entradas en la tabla:

| | |
|---|---|
| Restar 1 del mes en el AX | 000B (11 decimal) |
| Multiplicar por 9 (longitud de las entradas) | 0063 (99 decimal) |
| Sumar la dirección de la tabla (MONTAB) | MONTAB+63H |

Una manera de mejorar este programa es aceptar meses numéricos desde el teclado y verificar que sus valores estén entre 01 y 12, inclusive.

**Direccionamiento directo de tablas**

```
TITLE   P15DIREC (COM)   Direccionamiento directo de tablas
        .MODEL SMALL
        .CODE
        ORG     100H
BEGIN:  JMP     SHORT MAIN

NINE    DB      9
MONIN   DB      '12'
ALFMON  DB      9 DUP (20H), '$'
MONTAB  DB      'January  ', 'February ', 'March    '
        DB      'April    ', 'May      ', 'June     '
        DB      'July     ', 'August   ', 'September'
        DB      'October  ', 'November ', 'December '

.386
MAIN    PROC    NEAR                    Procedimiento principal
        CALL    C10CONV                 Convierte a binario
        CALL    D10LOC                  Localiza el mes
        CALL    F10DISP                 Despliega mes en forma alfabética
        MOV     AX,4C00H                Sale al DOS
        INT     21H
MAIN    ENDP
;
                Convierte ASCII a oinario:
;
¿10CONV PROC
        MOV     AH,MONIN                Configura el mes
        MOV     AL,MONIN+l
        XOR     AX,3030H                Limpia los 3 ASCII
        CMP     AH, 00                  •¿Es del mes 01 al 09?
        JZ      C20                       sí, entonces continuar
        SUB     AH, AH                    no, entonces limpiar el AH,
        ADD     AL, 10                    corregir para binario
C20:    RET
C10CONV ENDP
;
                Localizar el mes en la tabla:
;
D10LOC  PROC
        LEA     SI,MONTAB
        DEC     AL                      ,•Corregir para la tabla
        MUL     NINE                    /Multiplica AL por 9
        ADD     SI,AX
        MOVZX   CX.NINE                  Inicializa el movimiento de 9 cars.
        CLD
        LEA     DI,ALFMON
        REP MOVSB                       ¡Mueve 9 caracteres
        RET
D10LOC  ENDP
;
                Despliega el mes  (alfabético):

F10DISP PROC
        MOV     AH,09H                  •Petición de despliegue
        LEA     DX,ALFMON
        INT     21H
        RET
F10DISP ENDP
        END     BEGIN
```

Figura 15-1 Direccionamiento directo de tablas: ejemplo 1

Direccionamiento directo, ejemplo 2: Tablas de meses y días

El programa de la figura 15-2 recupera la fecha actual del DOS y la despliega. La función 2AH de la 21H del DOS envía los siguientes valores binarios:

```
TITLE     P15DISDA (EXE)  Despliega el día de la semana y el mes
          .MODEL  SMALL
          .STACK  64

          .DATA
SAVEDAY   DB      7
SAVEMON   DB      7
TEN       DB      10
ELEVEN    DB      11
TWELVE    DB      12
DAYSTAB   DB      'Sunday, $    , 'Monday, $   '
          DB      'Tuesday, $   , 'Wednesday, $'
          DB      'Thursday, $  , 'Friday, $  '
          DB      'Saturday, $'
MONTAB    DB      'January $  ', 'February $ ', 'March $    '
          DB      'April $    ', 'May $      ', 'June $     '
          DB      'July $     ', 'August $   ', 'September $'
          DB      'October $  ', 'November $ ', 'December $ '

          .CODE
BEGIN     PROC    FAR
          MOV     AX,@data        Inicializa
          MOV     DS,AX             registro de segmentos
          MOV     ES,AX
          MOV     AX,0600H
          CALL    Q10SCR          ;Limpia la pantalla
          CALL    Q20CURS         ;Coloca el cursor
          MOV     AH,2AH          ;Obtiene la fecha de hoy
          INT     21H
          MOV     SAVEMON,DH      ;Guarda el mes
          MOV     SAVEDAY,DL      ;Guarda el día del mes
          CALL    B10DAYWK        ;Despliega el día de la semana
          CALL    C10MONTH        ;Despliega el mes
          CALL    D10DAYMO        ;Despliega el día
          CALL    E10INPT         ;Espera por una entrada
          CALL    Q10SCR          ;Limpia la pantalla
          MOV     AX,4C00H        ;Sale al DOS
          INT     21H
BEGIN     ENDP

B10DAYWK  PROC    NEAR            ;Despliega el día de la semana
          MUL     TWELVE          ;Día (en AL) x 12
          LEA     DX, DAYSTAB     ;Dirección de la tabla
          ADD     DX, AX            ;más el desplazamiento
          MOV     AH,09H          ;Despliega
          INT     21H
          RET
B10DAYWK  ENDP
C10MONTH  PROC    NEAR            ;Despliega el mes
          MOV     AL,SAVEMON      ;Obtiene el mes
          DEC     AL              ;Disminuye en uno
          MUL     ELEVEN          ;Multiplica por la longitud de
          LEA     DX,MONTAB       ;Dirección de la tabla
          ADD     DX, AX            ;más desplazamiento
          MOV     AH,09H          ;Despliega
          INT     21H
          RET
C10MONTH  ENDP
.100
D10DAYMO  PROC    NEAR            ;Despliega día del mes
          MOVZX   AX,SAVEDAY      ;Obtiene día
          DIV     TEN             ;Convierte de binario
          OR      AX,3030H          a ASCII
          MOV     BX, AX          ;Guarda el día en ASCII
```

Figura 15-2 Direccionamiento directo de tablas: ejemplo 2

```
            MOV     AH,02H          /Despliega
            MOV     DL, BL            primer dígito
            INT     21H
            MOV     AH,02H          /Despliega
            MOV     DL, BH            segundo dígito
            INT     21H
            RET
D10DAYMO    ENDP

E10INPT     PROC    NEAR            Espera por entrada desde
            MOV     AH,10H          Petición de entrada
            INT     16H             Llama al BIOS
            RET
E10INPT     ENDP

QIOSCR      PROC    NEAR            /Recorre la pantalla
            MOV    AX,0600H
            MOV    BH,17H           Blanco sobre azul
            MOV    CX,0000
            MOV    DX,184FH
            INT    10H              Llama al BIOS
            RET
Q10SCR      ENDP

Q2 0CURS    PROC   NEAR
            MOV    AH,02H           /Petición para colocar el
            MOV    BH, 00           •Página
            MOV    DH, 10           /Renglón
            MOV    DL, 24           /Columna
            INT    10H
            RET
Q2 0CURS    ENDP
            END      BEGIN
```

Figura 15-2 (continuación)

AL Día de la semana (donde Sunday = 0)
CX Año (no es utilizado en este programa)
DH Mes (01-12)
DL Día del mes (01-31)

El programa utiliza estos valores para desplegar el día alfabético de la semana en la forma "Wednesday, September 12". Para este fin, el programa define una tabla de días de la semana llamada DAYSTAB, iniciando con Sunday, y una tabla de meses llamada MONTAB, iniciando en January.

Las entradas en DAYSTAB son de 12 bytes, y a cada descripción sigue una coma, un blanco, un signo $ y con blancos a la derecha. La función 09H de la INT 21H del DOS, despliega todos los caracteres hasta el signo $; la coma y el espacio en blanco son seguidos en la pantalla por el mes. El procedimiento B10DAYWK multiplica el día de la semana por 12 (la longitud de cada entrada en DAYSTAB). El producto es un desplazamiento en la tabla, donde, por ejemplo, Sunday está en DAYSTAB+0, Monday en DAYSTAB+12, y así sucesivamente. El día es desplegado directamente de la tabla.

Las entradas en MONTAB son de 11 bytes, con cada descripción seguida por un blanco y un signo $ y espacios en blanco a la derecha. El procedimiento C10MONTH primero disminuye el

mes en uno de manera que, por ejemplo, el mes 01 se convierte en la entrada cero en MONTAB. Después multiplica el mes por 11 (la longitud de cada entrada en MONTAB). El producto está en un desplazamiento de la tabla donde January está en MONTAB+0, February en MONTAB+11, etc. El mes es desplegado directamente de la tabla.

El procedimiento D10DAYMO divide el día del mes entre 10 para convertirlo de formato binario a ASCII. Como el número máximo para día es 31, tanto el cociente como el residuo sólo pueden ser de un dígito. (Por ejemplo, 31 dividido entre 10 da un cociente de 3 y un residuo de 1.) La función 02H del DOS despliega cada uno de estos dos caracteres, incluyendo el cero inicial para los días menores a 10; la supresión del cero inicial implica algunos pequeños cambios en el programa.

Al final, el programa espera a que el usuario presione una tecla antes de salir al DOS.

Aunque el direccionamiento directo de tabla es muy eficiente, funciona mejor cuando las entradas son secuenciales y en un orden predecible. Por tanto, funcionaría bien para entradas que están en el orden 1, 2, 3, o 106, 107, 108, oaúnpara5, 10, 15, .... Desafortunadamente, pocas aplicaciones proporcionan un arreglo tan ordenado de valores en la tabla. Una sección posterior examina tablas con valores que son secuenciales, pero no en un orden particular.

## BÚSQUEDA EN UNA TABLA

Algunas tablas consisten en números únicos sin patrón aparente. Un ejemplo típico es una tabla de elementos en existencia con números no consecutivos como 134, 138, 141, 239 y 245. Otro tipo de tabla —como una tabla de ingresos gravables— contiene márgenes de valores. Las siguientes secciones examinan ambos tipos de tablas y los requisitos para buscar en ellas.

### Tablas con entradas únicas

En la mayor parte de las compañías, los números de inventario por lo común no están en orden consecutivo. En lugar de eso, tienden a estar agrupadas en categorías, con un número inicial para indicar mueble o accesorio o para señalar que está localizado en cierto departamento. Además, con el tiempo algunos elementos son eliminados del inventario y otros son agregados. Como ejemplo, definamos una tabla con números de inventario y sus descripciones relativas. Éstas podrían ser definidas en tablas separadas, como

```
STOKNOS   DB  '05','10','12',  ...
STOKDESC DB 'Excavators', 'Lifters', 'Presses...' , ...
```

Cada paso en la búsqueda podría incrementar en dos la dirección de la primera tabla (la longitud de cada entrada en STOKNOS) y la dirección de la segunda tabla en 10 (la longitud de cada entrada en STOKDESC). Otro procedimiento podría mantener un conteo del número de ciclos ejecutados y encontrar una coincidencia con cierta llave de número de existencia, multiplicar el contador por 10 y utilizar el producto como un desplazamiento de la dirección de STOKDESC.

Por otra parte, puede ser más claro definir números de inventario y descripciones en la misma tabla, con una línea para cada par:

```
STOKTAB  DB   '05','Excavators'
         DB   '10' , 'Lifters'
         DB   '12','Presses...'
```

El programa en la figura 15-3 define esta tabla con seis pares de números de inventario y descripciones. El ciclo de búsqueda en A20 compara el primer byte del número de inventario de entrada, STOKNIN, con el primer byte de los números de inventario en la tabla. Si la comparación es igual, la rutina compara los segundos bytes. Si estos son *iguales,* el número de inventario ha sido encontrado y en A50 el programa copia la descripción de la tabla a DESCRN, donde es desplegada.

Si la comparación del primero o segundo byte es *menor,* se sabe que el numero de inventario no está en la tabla y, en A40, el programa puede desplegar un mensaje de error (no codificado).

Si la comparación del primero o segundo byte es *alta,* el programa tiene que continuar la búsqueda; para comparar el número de inventario de entrada con el siguiente número de inventario en la tabla, incrementa el SI, que contiene la dirección de la tabla. El ciclo de búsqueda realiza un máximo de seis comparaciones. Si el ciclo excede a seis, se sabe que el número de inventario no está en la tabla.

Verificamos esta lógica comparando de forma sucesiva los números de inventario 01, 06 y 10 con los elementos en la tabla:

- *Número de inventario 01 con elemento 05 de la tabla.* El primer byte es igual, pero el segundo byte es menor, así que el elemento no está en la tabla.
- *Número de inventario 06 con elemento 05 de la tabla.* El primer byte es igual pero el segundo es mayor, así que comparamos la entrada con el siguiente elemento en la tabla: número de inventario 06 con elemento de la tabla 10. El primer byte es menor, así que el elemento no está en la tabla.
- *Número de inventario 10 con elemento 05 de la tabla.* El primer byte es mayor, así que comparamos la entrada con el siguiente elemento de la tabla: número de inventario 10 con elemento 10 de la tabla. El primer byte es igual y el segundo también, así que el elemento se ha encontrado.

La tabla también puede definir precios unitarios. El usuario ingresa un número de inventario y la cantidad vendida. El programa podría localizar el elemento del inventario en la tabla, calcula la cantidad de la venta (cantidad vendida por precio por unidad) y despliega la descripción y la cantidad de la venta.

En la figura 15-3, el número de elemento es de dos caracteres y la descripción es de 10. La programación de detalles varía para diferentes números de entradas y diferentes longitudes de entradas. Por ejemplo, para comparar campos de tres bytes podría utilizar REPE CMPSB, aunque la instrucción implica al registro CX, el cual LOOP ya utiliza.

## Tablas con intervalos

Los ingresos gravables proporcionan un ejemplo característico de una tabla con intervalos de valores. Considere la siguiente tabla hipotética de ingresos gravables, tasas de impuesto y factores de ajuste:

| INGRESO GRAVABLE ($) | TASA | FACTOR DE AJUSTE |
|---|---|---|
| 0-1,000.00 | .10 | 0.00 |
| 1,000.01-2,500.00 | .15 | 050.00 |
| 2,500.01-4,250.00 | .18 | 125.00 |
| 4,250.01-S,000.00 | .20 | 260.00 |
| 6,000.01 y más | .23 | 390.00 |

```
TITLE   P15TABSR (COM)   Tabla de búsqueda que utiliza CMP
        .MODEL  SMALL
        .CODE
        ORG     100H
BEGIN:  JMP     SHORT MAIN

STOKNIN DB      '12'                    Entrada de núm. de inventario
STOKTAB DB      '05','Excavators'       Inicio de la tabla
        DB      '10','Lifters   '
        DB      '12','Presses   '
        DB      '15','Valves    '
        DB      '23','Processors'
        DB      '27','Pumps     '       Fin de la tabla
DESCRN  DB      10 DUP(?), '$'          Área para guardar

MAIN    PROC    NEAR
        MOV     CX, 06                  Inicializa comparación
        LEA     SI,STOKTAB
A20:
        MOV     AL,STOKNIN
        CMP     AL, [SI]                tExistencia(1) : tabla
        JNE     A30                     No es igual, salir
        MOV     AL,STOKNIN+l            Igual:
        CMP     AL, [SI + 1]              #Existencia(2) : tabla
        JE      A50                       igual, encontrada
A30:
        JB      A40                     ;Menor, no se encuentra en la tabla
        ADD     SI,12                   /Mayor, obtener la siguiente entrada
        LOOP    A20
A40:                                    No está en la tabla
;                                      ,-Mostrar mensaje de error
        JMP     A90
A50:
        MOV     CX, 05                  •Longitud de la descripción
        LEA     DI,DESCRN               •Dirección de la descripción
        INC     SI
        INC     SI                      •Extraer la descripción
        REP  MOVSW                      •  de la tabla
        MOV     AH,09H                  •Petición para desplegar
        LEA     DX, DESCRN                descripción de la existencia
        INT     21H

A90:    MOV     AX,4CO0H                Sale al DOS
        INT     21H
MAIN    ENDP
        END     BEGIN
```

Figura 15-3 Tabla de búsqueda que usa CMP

En la tabla de impuestos, las tasas se incrementan conforme lo hacen los ingresos gravables. I factor de ajuste compensa nuestro impuesto calculado en tasas altas, mientras que las tasas baje se aplican a niveles menores de ingresos. Las entradas para los ingresos gravables contienen < ingreso máximo para cada paso:

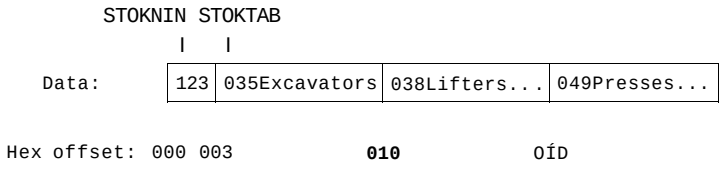
```
TAXTAB  DD  100000, 10, 00000
        DD  250000, 15, 05000
        DD  425000, 18, 12500
        DD  600000, 20, 26000
        DD  999999, 23, 39000
```

Para realizar una búsqueda en la tabla, el programa compara el ingreso gravable del contribuyente con las entradas en la tabla y hace lo siguiente, de acuerdo con los resultados de la comparación:

- Mayor: Incrementa para la entrada siguiente de la tabla.
- Menor o igual: Utiliza la tasa y el factor de ajuste asociados.
- Calcula la deducción de impuesto como (ingreso gravable x tasa de la tabla) - factor de ajuste. Observe que la última entrada en la tabla contiene el valor máximo (999999), que siempre finalizaría de manera correcta la búsqueda.

### Búsqueda en una tabla usando comparaciones de cadenas

REPE CMPS es útil para comparar números que son de dos o más bytes de longitud. El programa de la figura 15-4 define STOKTAB, pero esta vez corregido como un número de inventario de tres bytes. Ya que STOKNIN es el primer campo en el área de datos y STOKTAB es el siguiente, aparecen en el segmento de datos como sigue:




La última entrada de la tabla contiene '999' para que termine la búsqueda, ya que REPE hace que el CX no esté disponible para la instrucción LOOP. La rutina de búsqueda compara STOKNIN (definido de manera arbitraria con 123) con cada entrada de la tabla, como sigue:

| STOKNIN | ENTRADA DE LA TABLA | RESULTADO DE LA COMPARACIÓN |
|---|---|---|
| 123 | 035 | Mayor: examina entrada siguiente |
| 123 | 038 | Mayor: examina entrada siguiente |
| 123 | 049 | Mayor: examina entrada siguiente |
| 123 | 102 | Mayor: examina entrada siguiente |
| 123 | 123 | Igual: entrada encontrada |

El programa inicializa el DI al desplazamiento de la dirección de STOKTAB (003), el CX a la longitud (03) de cada número de inventario y el SI al desplazamiento de STOKNIN (000). La operación CMPSB compara byte por byte, hasta que los bytes sean iguales e incrementa de manera automática los registros DI y SI. Una comparación con la primer entrada de la tabla (123:035) causa la terminación después de un byte; el DI contiene 004, el SI contiene 001 y el CX contiene 02. Para la comparación siguiente, el DI debe contener 010 y el SI debe contener 000. La corrección del SI sólo implica volver a cargar la dirección de STOKNIN. Sin embargo, para la dirección de la entrada de la tabla el incremento depende de si la comparación terminó después de uno, dos o tres bytes. El CX contiene el número de bytes que quedan sin comparar, en este caso 02. Sumando el valor de CX más la longitud de la descripción del inventario da el desplazamiento del elemento siguiente de la tabla, como sigue:

```
                                        page   60,132
                                TITLE   P15STRSR (EXE)   Búsqueda utilizando CMPSB
                                        .MODEL   SMALL
                                        .STACK   64

                                        .DATA
0000 31 32 33                   STOKNIN DB   '123'
0003 30 33 35 45 78 63          STOKTAB DB   '035','Excavators'     ;Inicio de la tabla
     61 76 61 74 6F 72
     73
0010 30 33 38 4C 69 66                  DB   '038','Lifters    '
     74 65 72 73 20 20
     20
001D 30 34 39 50 72 65                  DB   '049', 'Presses   '
     73 73 65 73 20 20
     20
002A 31 30 32 56 61 6C                  DB   '102','Valves     '
     76 65 73 20 20 20
     20
0037 31 32 33 50 72 6F                  DB   '123', 'Processors'
     63 65 73 73 6F 72
     73
0044 31 32 37 50 75 6D                  DB   '127', 'Pumps     '
     70 73 20 20 20 20
     20
0051 39 39 39                           DB   '999', 10 DUP(' ')     ,-Fin de la tabla
     000A[  20 ]
005E 000A[ ?? ]                 DESCRN  DB   10 DUP(?),'$'          ;Área para guardar
     24

                                        -CODE
0000                            BEGIN   PROC   FAR
0000 B8     - R                         MOV    AX,@data             Inicializa
0003 8E D8                              MOV    DS, AX                 registros de
0005 8E CO                              MOV    ES, AX                 segmentos
0007 FC                                 CLD
0008 8D 3E 0003 R                       LEA    DI, STOKTAB          Inicializa dirección
OOOC                            A20:                                  de la tabla
OOOC B9 0003                            MOV    CX,03                Compara 3 bytes
OOOF 8D 36 0000 R                       LEA    SI,STOKNIN           Inic. dirección de #Exist
0013 F3/ A6                             REPE   CMPSB                #Exist : tabla
0015 74 09                              JE     A30                    igual, salir
0017 72 ID                              JB     A40                    menor, no hay entrada
0019 03 F9                              ADD    DI,CX                Sumar CX al desplazamiento
001B 83 C7 OA                           ADD    DI, 10               Siguiente elemento de la tab
001E EB EC                              JMP    A20
0020                            A30:
0020 B9 0005                            MOV    CX, 05               /Establecer para 5 palabras
0023 8B F7                              MOV    SI,DI
0025 8D 3E 005E R                       LEA    DI,DESCRN            ;Direc. de descrip.
0029 F3/ A5                             REP MOVSW                   ;Obtener descripción
                                                                      de la tabla
002B B4 09                              MOV    AH,09H               ;Petición de desplegar
002D 8D 16 005E R                       LEA    DX,DESCRN            ;  descrip. de existencia
0031 CD 21                              INT    21H
0033 EB 01 90                           JMP    A90                  • Ir a salir
0036                            A40:
                                ;           <Despliega mensaje de error>
0036                            A90:
0036 B8 4C00                            MOV    AX,4C00H             ;Sale al DOS
0039 CD 21                              INT    21H
003B CB                                 RET
003C                            BEGIN   ENDP
                                        END    BEGIN
```

Figura 15-**4** Búsqueda en una tabla usando CMPSB.

| | | |
|---|---|---|
| Dirección en DI después de CMPSB: | | 004H |
| Suma la longitud restante en CX: | + | 02H |
| Suma la longitud de la descripción: | + | OAH |
| Dirección del elemento siguiente: | | 010H |

Ya que el CX contiene el número de bytes que quedan por comparar (si existen), la aritmética funciona para todos los casos y termina después de una, dos o tres comparaciones. En una comparación que resulte igual, el CX contiene 00 y el DI ya está incrementado a la dirección de la descripción requerida. Entonces, una operación REP MOVSW copia la descripción en DESCRN, donde es desplegada.

### Tablas con entradas de longitud variable

Es posible definir una tabla con entradas de longitud variable. Un carácter delimitador especial, como 00H, puede seguir a cada entrada, y FFH podría distinguir el final de la tabla. Sin embargo, debe asegurarse de que ningún byte dentro de una entrada contenga la configuración de bits de un delimitador; por ejemplo, una cantidad aritmética binaria puede contener cualquier configuración posible de bits. Utilice la instrucción SCAS para buscar los delimitadores.

## LA INSTRUCCIÓN XLAT (TRADUCIR)

La instrucción XLAT traduce el contenido de un byte a otro valor predefinido. Usted puede utilizar XLAT, por ejemplo, para validar el contenido de los elementos de datos o, si'transfiere datos entre una PC y una macrocomputadora IBM, para traducir entre formatos ASCII y EBCDIC. El formato general para XLAT es

```
[etiqueta:]  XLAT  ;Sin operandos
```

El ejemplo siguiente convierte los números 0-9 ASCII a EBCDIC. Como la representación en ASCII es 30-39 y en EBCDIC es F0-F9, puede utilizar una operación OR para realizar el cambio. Sin embargo, también convertirá todos los otros caracteres en blanco, EBCDIC 40H. Para XLAT, se define una tabla de traducción que toma en cuenta todos los 256 posibles caracteres, con códigos EBCDIC insertados en las posiciones ASCII:

```
XLTBL   DB  48 DUP(40H)                    ;espacios en blanco EBCDIC
        DB  0F0H,0F1H,0F2H,0F3H,    ,0F9H  ;EBCDIC 0-9
        DB  198 DUP(4OH)                   ;espacios en blanco EBCDIC
```

XLAT espera que la dirección de la tabla esté en el registro BX y el byte que se va a traducir (llamémoslo ASCNO) está en el AL. Lo siguiente realiza la inicialización y traducción:

```
LEA   BX,XLTBL      Carga la dirección de la tabla
MOV   AL,ASCNO      Carga el carácter a ser traducido
XLAT                Traduce a EBCDIC
```

```
TITLE   P15XLATE  (COM)   Traduce ASCII a EBCDIC
        .MODEL  SMALL
        .CODE
        ORG     100H
BEGIN:  JMP     MAIN
;-------------------------------------------------------------
ASCNO   DB      '-31.5 '            ;Elemento ASCII a convertir
EBCNO   DB      6 DUP(' ')          ;Elemento EBCDIC convertido
XLTAB   DB      45 DUP(40H)         ;Tabla de traducción
        DB      60H, 4BH
        DB      40H
        DB      OFOH,0F1H,0F2H,0F3H,0F4H
        DB      0F5H,0F6H,0F7H,0F8H,0F9H
        DB      198 DUP(40H)
;-------------------------------------------------------------
MAIN    PROC    NEAR
        LEA     SI,ASCNO            ;Dirección de ASCNO
        LEA     DI,EBCNO            ;Dirección de EBCNO
        MOV     CX, 06              ;-Longitud de los elementos
        LEA     BX,XLTAB            ;-Dirección de la tabla
A20:
        LODSB                       ;Obtener carácter en AL
        XLAT                        ;Traducir el carácter
        STOSB                       ;Almacenar AL en EBCNO
        LOOP    A20                 ;Repetir 6 veces

        MOV     AX,4C00H            ;Salir al DOS
        INT     21H
MAIN    ENDP
        END     BEGIN
```

Figura 15-5 Conversión de ASCII a EBCDIC

XLAT uiliza el contenido del AL como un desplazamiento de dirección; en realidad, el BX contiene la dirección de inicio de la tabla y el AL contiene un desplazamiento dentro de la tabla. Si, por ejemplo, el valor en AL es 00 la dirección de la tabla sería XLTBL+0 (el primer byte de XLTBL con 40H). XLAT reemplazaría el 00 en el AL con 40H desde la tabla.

Observe que el primer DB en XLTBL define 48 bytes, con direcciones desde XLTBL+00 hasta XLTBL+47. El segundo DB en XLTBL define datos empezando en XLTBL+48. Si el número es 32H (50 decimal), la dirección de la tabla es XLTBL+50; esta localidad contiene F2 (2 EBCDIC), que XLAT insertará en el registro AL.

El programa en la figura 15-5 extiende este ejemplo para convertir el signo menos ASCII (2D) y el punto decimal (2E) a EBCDIC (60 y 4B, respectivamente) y repetir el proceso en un campo de seis bytes. En un inicio, ASCNO contiene-31.5 seguido por un blanco, o 2D33312E3520 hexadecimal. Al final del ciclo, EBCNO debe contener 60F3F14BF540.

## DESPLIEGUE DE CARACTERES HEXADECIMALES Y ASCII

El programa de la figura 15-6 despliega los 256 números hexadecimales (00-FF), incluyendo la mayoría de los símbolos ASCII relacionados. Por ejemplo, el programa despliega el símbolo ASCII S junto con su representación hexadecimal, 53. El despliegue completo aparece en la pantalla como una matriz de 16 por 16:

```
          page 60,132
TITLE     P15ASCHX (COM)  Despliega caracteres ASCII y hexadecimales
          .MODEL SMALL
          .CODE
          ORG    100H
BEGIN:    JMP    SHORT MAIN
-------------------------------------------------------------------
DISPROW   DB     16 DUP(5 DUP(' '),13
HEXCTR    DB     00
XLATAB    DB     30H,31H,32H,33H,34H,35H,36H,37H,38H,39H
          DB     41H,42H,43H,44H 45H,46H
-------------------------------------------------------------------
MAIN      PROC   NEAR                 /Procedimiento princip
          CALL   Q10CLR               ,-Limpia la pantalla
          LEA    SI,DISPROW
A20LOOP:
          CALL   C10HEX               /Traduce
          CALL   D10DISP              /  y despliega
          CMP    HEXCTR,0FFH          /¿Es el último número
          JE     A50                  /  sí, termina
          INC    HEXCTR               /  no, obtener el siguí
          JMP    A20LOOP
A50:      MOV    AX,4C00H             /Salir al DOS
          INT    21H
MAIN      ENDP

C10HEX    PROC   NEAR                 /Convierte a hexadecimal
          MOV    AH,00
          MOV    AL,HEXCTR            /Obtiene una pareja hex
          MOV    CL,04                /Fija el valor de corrimiento
          SHR    AX,CL                /Recorre a la derecha un dígito hex
          LEA    BX,XLATAB            /Designa la dirección de la tabla
          XLAT                        /Traduce hex
          MOV    [SI],AL              /Almacena el carácter izquierdo

          MOV    AL,HEXCTR
          AND    AL,0FH               /Limpia el dígito hex de la izquierda
          XLAT                        /Traduce hex
          MOV    [SI]+1,AL            /Almacena el carácter de la derecha
          RET
C10HEX    ENDP
D10DISP   PROC   NEAR                 /Despliega
          MOV    AL,HEXCTR
          MOV    [SI]+3,AL
          CMP    AL,1AH               /¿Es el carácter EOF?
          JE     D20                  /  sí, pasar
          CMP    AL,07H               /¿Es menor que 7?
          JB     D30                  /  sí, ok
          CMP    AL,10H               /¿Es mayor o igual a 16?
          JAE    D30                  /  sí, ok
D20:                                  /En caso contrario forzar un espacio en
          MOV    BYTE PTR [SI]+3 20H
D30:
          ADD    SI,05                /Siguiente lugar en renglón
          LEA    DI,DISPROW+80
          CMP    DI,SI                /¿Está lleno el renglón?
          JNE    D40                  /  no, pasar

          MOV    AH.40H               /Sí, petición de despliegue
          MOV    BX,01                /  manejador de archivo
          MOV    CX,81                /  todo el renglón
          LEA    DX,DISPROW
          INT    21H
          LEA    SI,DISPROW           ,-Redesigna renglón de despliegue
```

Figura 15-6 Despliegue de ASCII y hexadecimal

```
D40:        RET
D10DISP     ENDP

Q10CLR      PROC    NEAR                ;Limpia la pantalla
            MOV     AX,0600H
            MOV     BH,61H              ;atributo
            MOV     CX,0000
            MOV     DX,184FH
            INT     10H
            RET
Q10CLR      ENDP
            END     BEGIN
```

Figura 15-6 (continuación)

```
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F



F0 Fl F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF
```

Como vio en la figura 8-1, desplegar símbolos ASCII no causa problemas graves. Sin embargo, desplegar la representación hexadecimal de un valor ASCII es más complicado. Por ejemplo, para desplegar un hexadecimal como ASCII, tiene que convertir OOH a 3030H, 01H a 3031H, etcétera.

El programa inicializa HEXCTR a 00 y por cada ciclo lo incrementa en 1 de manera sucesiva. El procedimiento C10HEX divide HEXCTR en dos dígitos hexadecimales. Por ejemplo, suponga que HEXCTR contiene 4FH. La rutina extrae el 4 hex y lo utiliza junto con una tabla para una traducción. El número que regresa al AL es 34H. Después, la rutina extrae la F y la traduce a 46H. El resultado, 3446H, se despliega como 4F.

El procedimiento D10DISP convierte caracteres no ASCII en blancos. Ya que la función 40H de la INT 21H del DOS trata a 1AH como un carácter de fin de archivo, el programa también lo cambia a espacio en blanco. Cuando un renglón está lleno con 16 caracteres, el procedimiento lo despliega; el procedimiento termina después de desplegar el renglón decimosexto.

Existen muchas otras maneras de convertir dígitos hexadecimales en caracteres ASCII; por ejemplo, puede experimentar con corrimientos y comparaciones.

## ORDENAMIENTO DE ENTRADAS DE UNA TABLA

Con frecuencia, una aplicación necesita *ordenar* datos de una tabla en secuencia ascendente o descendente. Por ejemplo, un usuario puede necesitar una lista de descripciones de existencias en orden ascendente, o una lista del total de ventas por agente en orden descendente. Existen varias rutinas para ordenar tablas, desde las ineficientes pero claras hasta las eficientes pero poco claras. La rutina presentada en esta sección es bastante eficiente y puede servir para la mayoría de los ordenamientos de tablas.

Un enfoque general para ordenar una tabla es comparar una entrada de la tabla con la entrada que está inmediatamente después de ella. Si resulta ser mayor, entonces se intercambian las entradas. Continúe de esta manera, comparando la entrada 1 con la entrada 2, la entrada 2 con

la entrada 3, y así hasta el final de la tabla, intercambiando las entradas cuando sea necesario. Si realiza algún intercambio de entradas, repita el proceso desde el inicio de la tabla, comparando la entrada uno con la entrada dos, otra vez. Si no se realiza ningún intercambio, la tabla ya está en secuencia y se puede terminar el ordenamiento.

En el seudocódigo siguiente, SWAP es un elemento que indica si se realizó un intercambio (SÍ) o no (NO).

```
G10 :    Inicializa la dirección de la última entrada de la tabla
G20:     Hace SWAP igual a NO
         Inicializa la dirección del inicio de la tabla
G30:     ¿La entrada de la tabla > la siguiente entrada?
             Sí:  Intercambiar entradas
                  Hacer SWAP igual a SÍ
         Incrementar a la siguiente entrada de la tabla
         ¿Es el final de la tabla?
             No:  Pasar a G30
             Sí:  ¿Es SWAP = SÍ?
                  Sí: Pasar a G20 (repetir el ordenamiento)
                                  y
                  No: Terminar el ordenamiento
```

El programa de la figura 15-7 permite al usuario ingresar hasta 30 nombres desde el teclado, que el programa almacena de manera sucesiva en una tabla llamada NAMETAB. Cuando se han ingresado todos los nombres, el usuario sólo presiona la tecla Enter, sin ningún nombre. Entonces, el programa ordena la tabla de nombres en secuencia ascendente y los despliega en la pantalla. Observe que las entradas de la tabla son todas de longitud fija de 20 bytes. Una rutina para ordenar información de longitud variable sería más complicada.

## LISTAS LIGADAS (ENLAZADAS)

Una *lista ligada* contiene datos llamados celdas, al igual que las entradas de una tabla, pero sin secuencia específica. Cada celda contiene un *apuntador (puntero)* a la celda siguiente para facilitar las búsquedas hacia adelante. (Una celda también puede contener un apuntador a la celda precedente de manera que la búsqueda pueda realizarse en cualquier dirección.) El método facilita agregar y eliminar de una lista sin la necesidad de expandirla o contraerla.

Para nuestros propósitos, la lista ligada contiene celdas con un número de parte (un valor ASCII de cuatro bytes), precio por unidad (palabra binaria) y un apuntador (palabra binaria) a la celda siguiente en la lista, que contiene el siguiente número de parte en la secuencia. Por tanto la entrada es de ocho bytes de longitud. El apuntador es un desplazamiento desde el inicio de la

```
          page 60,132
TITLE     P15NMSRT (EXE)  Ordena nombres ingresados desde la terminal
          .MODEL SMALL
          .STACK 64
-------------------------------------------------------------------------
          .DATA
ÑAME PAR  LABEL   BYTE              |Lista de parámetros de nombre:
MAXNLEN   DB      21                |  longitud máxima
NAMELEN   DB      ?                 |  núm. de caracteres ingresados para el
NAMEFLD   DB      21 DUP(' ')       |  nombre

CRLF      DB      13, 10, '$'
ENDADDR   DW      ?
MESSG1    DB      'Ñame? ', '$'
NAMECTR   DB      00
NAMETAB   DB      30 DUP(20 DUP(  ')) /Tabla de nombres
NAMESAV   DB      20 DUP(?), 13, 10, '$'
SWAPPED   DB      00
-------------------------------------------------------------------------
          .CODE
BEGIN     PROC    FAR
          MOV     AX,@data          Inicializa los registros
          MOV     DS,AX               DS y ES
          MOV     ES,AX
          CLD
          CALL    Q10CLR            Limpia la pantalla
          CALL    Q20CURS           ;Coloca el cursor
          LEA     DI,NAMETAB
A20LOOP:
          CALL    B10READ           ,-Acepta un nombre
          CMP     NAMELEN,00        ¿Existen más nombres?
          JZ      A30                 no, ir a ordenar
          CMP     NAMECTR,30        ¿Han ingresado 30 nombres?
          JE      A30                 sí, ir a ordenar/
          CALL    D10STOR           /Almacenar en la tabla el nombre ingresado
          JMP     A20LOOP
A30:                                |Fin de la entrada
          CALL    Q10CLR            |Limpiar la pantalla
          CALL    Q20CURS           |  y colocar el cursor
          CMP     NAMECTR,01        |¿Se ingresó uno o ningún nombre?
          JBE     A40               |  sí, salir
          CALL    G10SORT           |Ordenar los nombres almacenados
          CALL    K10DISP           |Mostrar los nombres ordenados
A40:      MOV     AX,4C00H          |Salir al DOS
          INT     21H
BEGIN     ENDP
          Acepta nombres como entrada:
-------------------------------------------------------------------------
BlOREAD   PROC
          MOV     AH,09H
          LEA     DX,MESSG1         /Despliega indicación
          INT     21H
          MOV     AH, OAH
          LEA     DX,ÑAMEPAR        /Acepta nombre
          INT     21H
          MOV     AH,09H
          LEA     DX,CRLF           /Retorno/avance de línea
          INT     21H

          MOV     BH, 00            /Limpia los caracteres después
          MOV     BL,NAMELEN        /Obtiene el contador de cars.
          MOV     CX, 21
          SUB     CX.BX             /Calcula la longitud restante
```

Figura 15-7 Ordena una tabla de nombres

```
B20:
        MOV     NAMEFLD[BX],20H     ¡Designa el nombre en blanco
        INC     BX
        LOOP    B20
        RET
BIOREAD ENDP
                ¡Almacena nombre en la tabla:

D10STOR PROC
        INC     NAMECTR             ¡Suma al número de nombres
        CLD
        LEA     SI,NAMEFLD
        MOV     CX,10               ¡Diez palabras
        REP MOVSW                   ¡Nombre (SI) a la tabla (DI)
        RET
D10STOR ENDP
                Ordena los nombres de la tabla:
-------------------------------------------------------------------

G10SORT PROC
        SUB     DI,40               ¡Designa la dirección de detención
        MOV     ENDADDR,DI
G20:
        MOV     SWAPPED,00          ¡Designa el inicio
        LEA     SI,NAMETAB          ¡  de la tabla
G30:
        MOV     CX,20               ¡Longitud de comparación
        MOV     DI,SI
        ADD     DI,20               ¡Nombre siguiente por comparar
        MOV     AX,DI
        MOV     BX,SI
        REPE CMPSB                  ¡Compara el nombre con el siguiente
        JBE     G40                   no hay intercambio
        CALL    H10XCHG             ¡  intercambia
G40:
        MOV     SI,AX
        CMP     SI,ENDADDR          ¡¿Es el fin de la tabla?
        JBE     G30                 ;  no, continuar
        CMP     SWAPPED,00          ¡¿Hubo algún intercambio?
        JNZ     G20                 ¡  sí, continuar
        RET                         ¡  no, fin del ordenamiento
G10SORT ENDP

                Intercambia entradas de la tabla:
-------------------------------------------------------------------

H10XCHG PROC
        MOV     CX,10               ¡Número de caracteres
        LEA     DI,NAMESAV
        MOV     SI,BX
        REP MOVSW                   ¡Mueve el elemento menor para guardarlo

        MOV     CX,10               ¡Número de caracteres
        MOV     DI,BX
        REP MOVSW                   ¡Mueve el mayor al menor

        MOV     CX,10
        LEA     SI,NAMESAV
        REP MOVSW                   ¡Mueve el guardado al elemento mayor
        MOV     SWAPPED,01          ¡Señala que se realizó un intercambio
        RET
H10XCHG ENDP
                Despliega los nombres ordenados:

K10DISP PROC
        LEA     SI,NAMETAB
```

Figura 15-7 (continuación)

```
K20:
          LEA     DI,NAMESAV          ;Inicializa el principio de la tabla
          MOV     CX,10               ,-Cuenta los ciclos
          REP  MOVSW
          MOV     AH,09H              /Petición de despliegue
          LEA     DX,NAMESAV
          INT     21H
          DEC     NAMECTR             ¿Es el último?
          JNZ     K20                 ;  no, repetir el ciclo
          RET                         /  sí, salir
K10DISP   ENDP
                Limpia la pantalla:

Q10CLR    PROC
          MOV     AX,0600H
          MOV     BH,61H              /Atributo
          MOV     CX, 00              /Pantalla completa
          MOV     DX,184FH
          INT     10H
          RET
Q10CLR    ENDP
                Coloca el cursor:

Q20CURS   PROC
          MOV     AH,02H               Petición de colocar el cursor
          MOV     BH, 00               Página 0
          MOV     DX, 00               Posición 00:00
          INT     10H
          RET
Q20CURS   ENDP
          END     BEGIN
```

Figura 15-7 (continuación)

lista. La lista ligada inicia en el desplazamiento 0000, el segundo elemento en la serie está en 0024, el tercero en 0032, y así sucesivamente:

| DESPLAZAMIENTO | NO. PARTE | PRECIO | SIG. DIRECCIÓN |
|---|---|---|---|
| 0000 | 0103 | 12.50 | 0024 |
| 0008 | 1720 | 08.95 | 0016 |
| 0016 | 1827 | 03.75 | **0000** |
| 0024 | 0120 | 13.80 | 0032 |
| 0032 | 0205 | 25.00 | 0008 |

El elemento del desplazamiento 0016 contiene cero como la dirección siguiente, para indicar que es el final de la lista o para crear una lista circular.

El programa en la figura 15-8 utiliza el contenido de la lista ligada definida, LINKLST, para localizar un número de parte específico, en este caso 1720. La búsqueda empieza con el primer elemento de la tabla. La lógica para usar CMPSB es similar a la de la figura 15-4. El programa compara el número de parte (1720) con cada elemento de la tabla y hace lo siguiente, de acuerdo con el resultado de la comparación:

- Igual: La búsqueda ha terminado.
- Menor: El elemento no está en la tabla.
- Mayor: El programa obtiene el desplazamiento de la tabla para el elemento siguiente a ser comparado. Si el desplazamiento no es cero, la comparación se repite para el siguiente elemento; si el desplazamiento es cero, la búsqueda termina sin encontrar al elemento.

Un programa más completo podría permitir al usuario introducir cualquier número de parte desde el teclado y desplegar el precio como una cifra ASCII.

## TIPO, LONGITUD Y TAMAÑO DE LOS OPERADORES

El ensamblador provee de varios operadores especiales que usted puede considerar útiles. Por ejemplo, la longitud de una tabla puede cambiar de vez en vez, y tal vez haya que modificar un programa para tomar en cuenta la nueva definición y agregar rutinas que verifiquen el final de la tabla. El uso de los operadores TYPE, LENGTH y SIZE puede ayudar a reducir el número de instrucciones que tienen que ser cambiadas.

Considere esta definición de una tabla con 10 entradas:

```
TABLEX  DW  10 DUP (?)   ;Tabla con diez palabras
```

El programa puede usar el operador TYPE para determinar la definición (en este caso DW), el operador LENGTH para determinar el factor DUP (10) y el operador SIZE para determinar el número de bytes (10 x 2, o 20). Los ejemplos siguientes ilustran los tres operadores:

```
MOV  AX,TYPE TABLEX      ;AX = 0002  (2 bytes)
MOV  BX,LENGTH TABLEX    ;BX = 000A  (10 bytes)
MOV  CX,SIZE TABLEX      ;CX = 0014  (20 bytes)
```

Puede utilizar los valores que LENGTH y SIZE regresan para terminar una búsqueda o clasificación de una tabla. Por ejemplo, si el registro SI contiene la dirección de desplazamiento incrementada de una búsqueda, puede examinar este desplazamiento por medio de

```
CMP  SI,SIZE TABLEX
```

El capítulo 27 describe en detalle los operadores TYPE, LENGTH y SIZE.

## PUNTOS CLAVE

- Para la mayoría de los propósitos, se definen tablas de manera que sus entradas estén relacionadas y tengan la misma longitud y formato de datos.
- Diseñe tablas con base en su formato de datos. Por ejemplo, las entradas de la tabla pueden ser caracteres o numéricas y cada una de uno, dos o más bytes de longitud.
- Recuerde que el número máximo para un DB es 256 y que los DW y DD invierten los bytes. Además, CMP y CMPSW suponen que las palabras contienen bytes en secuencia inversa.
- Si una tabla está sujeta a cambios frecuentes, o si varios programas hacen referencia a la tabla, almacénela en disco. Un programa de actualización puede manejar los cambios en la tabla. Entonces, cualquier programa puede cargar la tabla del disco y los programas no necesitan ser modificados.
- Bajo direccionamiento directo, el programa calcula la dirección de una entrada en la tabla y la accesa de manera directa.

```
TITLE    P15LNKLS (EXE)   Uso de una lista ligada
         .MODEL   SMALL
         .STACK  64                 /Define la pila

         .DATA
PARTNO   DB      '1720               /Número de parte
LINKLST  DB      '0103               Tabla de la lista ligada
         DW      1250, 24
         DB      '1720
         DW      0895, 16
         DB      '1827
         DW      0375, 00
         DB      '0120
         DW      1380,  32
         DB      '0205
         DW      2500, 08

         .CODE                      /Define segmento de código
BEGIN    PROC    FAR
         MOV     AX,©data           |Designa dirección de DATASG
         MOV     DS,AX              |  en el registro DS
         MOV     ES, AX             |  y ES
         CLD
         LEA     DI,LINKLST         Inicializa la dirección de la tabla
A20:
         MOV     CX, 04             |Fija para comparar 4 bytes
         LEA     SI, PARTNO         |Inic. la dirección del # de parte
         REPE CMPSB                 |#Parte : tabla
         JE      A30                |  igual, salir
         JB      A40                |  menor, no está en la tabla
         ADD     DI, CX             |Suma el valor del CX al desplazamiento
         ADD     DI, 02             |Obtiene el desplazamiento del elemento
         MOV     DX, [DI]           |
         LEA     DI,LINKLST
         ADD     DI, DX
         CMP     DX, 00             ¿Es la última entrada de la tabla?
         JNE     A20
         JMP     A40
A30:
;                <Elemento encontrado
         JMP     A90
A40:
/                <Despliega mensaje de error>
A90:
         MOV     AX,4C00H           /Sale al DOS
         INT     21H
BEGIN    ENDP

         END     BEGIN
```

Figura 15-8 Lista ligada

- Cuando busca en una tabla, un programa compara de manera sucesiva un elemento contra cada entrada en la tabla hasta que encuentra la que coincida.
- La instrucción XLAT facilita la traducción de datos de un formato a otro.

## PREGUNTAS

15-1. Dé las diferencias entre el procesamiento de una tabla para direccionamiento directo y para búsqueda.

15-2. Defina una tabla llamada TABLEX con 50 palabras, inicialícela con blancos.

**15**-3. Defina tres diferentes tablas relacionadas que contengan los datos siguientes: (a) los números 06, 10, 14, 21 y 24; (b) las descripciones de una videocinta, receptores, módem, teclados y discos flexibles; (c) los precios 93.95, 82.25, 90.67, 85.80 y 13.85.

**15-4.** Codifique un programa que permita al usuario ingresar, desde el teclado, los números de elementos (ITEMIN) y cantidades (QTYIN). Utilice las tablas definidas en la pregunta 15-3, e incluya una rutina de búsqueda que utilice ITEMIN para localizar el número de elemento en la tabla. Extraiga las descripciones y precios de la tabla. Calcule el valor de cada venta (cantidad x precio) y despliegue la descripción y valor en la pantalla.

**15-5.** Usando la tabla de descripción definida en la pregunta 15-3, codifique lo siguiente: (a) una rutina que mueva el contenido de la tabla a otra tabla (inicialmente vacía); (b) una rutina que ordene el contenido de esta nueva tabla en secuencia ascendente de descripción.

**15-6.** Se necesita un programa que proporcione un cifrado sencillo de información. Defina un área de datos de 80 bytes llamada CRYPTEXT con cualesquiera datos ASCII. Arregle una tabla de traducción para convertir los datos de manera un poco aleatoria, por ejemplo, A a X, B a E, C a R, etc. Proporcione todos los posibles valores de byte. Arregle una segunda tabla de traducción que invierta (descifre) la información. El programa debe realizar las acciones siguientes:

- Desplegar el contenido original de CRYPTEXT en una línea.
- Cifrar CRYPTEXT y desplegar la información cifrada en una segunda línea.
- Descifrar CRYPTEXT y desplegar la información descifrada en una tercer línea. (Debe mostrar la misma información que la primer línea.)

## PARTE E—Entrada/Salida avanzadas

## CAPÍTULO 16

# Organización del almacenamiento en disco

OBJETIVO

**Examinar los formatos básicos de almacenamiento en disco duro y en disco flexible, el registro de arranque, directorio y tabla de asignación de archivos.**

### INTRODUCCIÓN

En algún momento, un programador serio tiene que estar familiarizado con los detalles técnicos de la organización en disco, en particular para el desarrollo de programas de utilerías que examinan el contenido de los discos flexibles o los discos duros. En este libro, en donde se necesite hacer referencia a un disco o a un disco flexible se usará el término general *disco.*

Este capítulo explica los conceptos de pistas, sectores y cilindros y da las capacidades de algunos dispositivos usados.

También se cubre la organización de importante información registrada al inicio de un disco, incluyendo el registro de arranque (que ayuda al sistema a cargar los programas del DOS desde disco y hacia la memoria); el directorio (que contiene el nombre, ubicación y estado de cada archivo en el disco) y la tabla de asignación de archivos (o FAT, que asigna espacio de disco para los archivos).

### CARACTERÍSTICAS DE LOS DISCOS

Para procesar registros en discos, es útil estar familiarizados con los términos y características de su organización. Un disco flexible tiene dos lados (o superficies), mientras que un disco duro contiene varios discos de dos lados.

### Pistas y sectores

Cada lado de un disco flexible o de un disco duro contiene varias *pistas* concéntricas, numeradas iniciando con 00, la pista más externa. Cada pista está formateada en *sectores* de 512 bytes, en donde se almacena la información.

Los dispositivos de discos flexibles y de discos duros se hacen funcionar por un *controlador* que maneja la colocación de las cabezas de lectura/escritura sobre la superficie del disco y la transferencia de información entre el disco y la memoria. Existe una cabeza de lectura/escritura por cada superficie del disco. Para ambos, disco duro y disco flexible, una petición de lectura o una de escritura hace que el controlador del disco mueva las cabezas de lectura/escritura (si es necesario) a la pista requerida. Después, el controlador espera a que el sector que se necesita, sobre la superficie que está girando, llegue a la cabeza. En ese momento la operación de lectura o escritura tiene lugar. La figura 16-1 ilustra este procedimiento.

Existen dos diferencias principales entre una unidad de disco duro y una de disco flexible. En disco duro, las cabezas de lectura/escritura se encuentran sobre la superficie del disco sin tocarla, mientras que en un disco flexible la cabeza de lectura/escritura realmente toca la superficie. Además, un dispositivo de disco duro gira de manera constante, mientras que un dispositivo de disco flexible gira y se detiene con cada operación de lectura/escritura.

### Cilindros

El *cilindro* es el conjunto vertical de todas las pistas con el mismo número en cada superficie de un disco flexible o de un disco duro. Por tanto el cilindro 0 es el conjunto de pistas numeradas con 0, el cilindro 1 es el conjunto de pistas con número 1, y así sucesivamente. Entonces, en un disco flexible el cilindro 0 consiste en la pista 0 en el lado 1 y la pista 0 en el lado 2; el cilindro 1 consiste en la pista 1 en el lado 1 y la pista 1 en el lado 2, etc. Cuando se escribe un archivo, el sistema llena todas las pistas de un cilindro y después avanza las cabezas de lectura/escritura al cilindro siguiente.

Una referencia a las caras (cabezas) de un disco, pistas y sectores es por medio de un número. Los números de lado y pista empiezan con 0, pero los sectores pueden ser numerados de una de dos formas:

1. *Dirección pista-cilindro:* Los números de sector en cada pista inician con 1, así que el primer sector en el disco tiene la dirección cilindro 0, pista 0, sector 1.

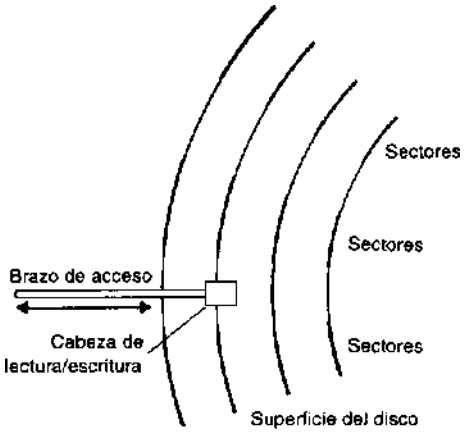



**Figura 16-1** Superficie del disco y cabeza de lectura-escritura

2. *Número de sector relativo:* Los sectores pueden ser numerados de manera relativa al inicio del disco, de modo que el primer sector en el disco que está sobre el cilindro 0, pista 0, tiene la dirección de sector relativo 0.

### Controlador de disco

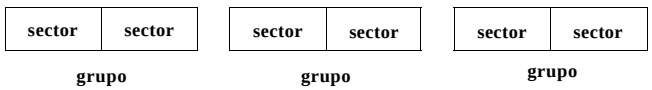
El controlador de disco está localizado entre el procesador y la unidad de disco y maneja toda la comunicación entre ellos. El controlador acepta información del procesador y la convierte una forma que pueda usar el dispositivo. Por ejemplo, el procesador puede enviar una petición de datos de un cilindro, cabeza de disco y sector específicos. El papel del controlador es proporcionar los comandos apropiados para mover el brazo de acceso al cilindro necesario, seleccionar la cabeza de lectura/escritura y aceptar la información del sector cuando ésta llegue a la cabeza de lectura/escritura.

El procesador está liberado para otras tareas mientras el controlador realiza su trabajo. Bajo este enfoque, el controlador maneja sólo un byte a la vez. Sin embargo, el controlador también puede realizar E/S más rápidas evitando al procesador por completo y transfiriendo la información de manera directa a la memoria y desde ella. El método de transferir un bloque grande de datos de esta manera es conocido como *acceso directo a memoria (DMA).* Para este fin, el procesador proporciona al controlador con el comando de lectura o escritura, la dirección del búfer de E/S en memoria, el número de sectores a transferir y el número de cilindro, cabeza y sector inicial. Con este método, el procesador tiene que esperar hasta que el DMA esté completo, ya que sólo un componente a la vez puede utilizar la ruta a la memoria.

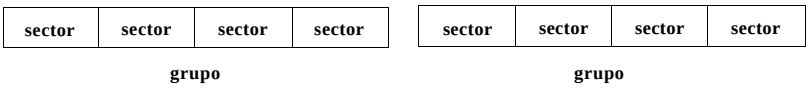
### Grupos

Un *grupo* es un conjunto de sectores que el DOS trata como una unidad de espacio de almacenamiento. El tamaño de grupo siempre es una potencia de 2, como 1, 2, 4 u 8 sectores. Por lo común un disco duro tiene cuatro sectores por grupo. En un dispositivo de disco que utiliza un sector por grupo, sector y grupo son lo mismo. Un archivo empieza en una frontera de grupo y necesita un mínimo de un grupo aunque el archivo sólo ocupe uno de los cuatro sectores.. Un grupo se puede traslapar de una pista a otra.

Un disco con dos sectores por grupo se vería así:




Y un disco con cuatro sectores por grupo se vería así:




Un archivo de 100 bytes (suficientemente pequeño para ocupar un sector) almacenado en un disco con cuatro sectores por grupo utiliza 4 x 512 = 2,048 bytes de almacenamiento, aunque sólo un sector contendría información. El DOS almacena los grupos para archivo en orden ascendente, aunque un archivo puede estar fragmentado de manera que resida, por ejemplo, en los grupos 8, 9, 10, 14, 17 y 18.

### Capacidad de disco

A continuación se presentan capacidades comunes de almacenamiento en disco flexible:

| Capacidad | Pistas por lado (cilindros) | Sectores por pista | Bytes por sector | Total, dos lados | Sectores por grupo |
|---|---|---|---|---|---|
| 5.25" 360KB | 40 | 9 | 512 | 368,640 | 2 |
| 5.25" 1.2MB | 80 | 15 | 512 | 1,228,800 | 1 |
| 3.5" 720KB | 80 | 9 | 512 | 737,280 | 2 |
| 3.5" 1.44MB | 80 | 18 | 512 | 1,474,560 | 1 |
| 3.5" 2.88MB | 80 | 36 | 512 | 2,949,120 | — |

En los discos duros, las capacidades varían considerablemente por dispositivo y por partición. Las operaciones útiles para la determinación del número de cilindros, sectores por pista o cabezas de lectura/escritura incluyen las funciones 1FH y 440DH con código secundario 60H, ambas de la INT 21H, y tratadas en el capítulo 18.

## ÁREA DE SISTEMA Y ÁREA DE DATOS EN DISCO

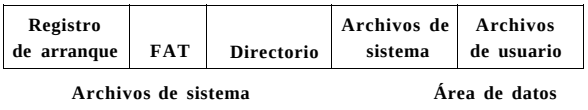
Para dar cuenta de la información almacenada en disco, el DOS reserva ciertos sectores para sus propios objetivos. La organización de discos flexibles y de discos duros varía de acuerdo con su capacidad. Un disco duro y algunos discos flexibles están formateados para autoarranque, esto es, son capaces de procesar el inicio cuando se enciende la computadora o cuando el usuario presiona las teclas Ctrl + Alt + Del. La organización general de un disco es un área de sistema, seguida por un área de datos que comprende el resto del disco.

### Área de sistema

El *área de sistema* es la primer área del disco, en la(s) pista(s) más externa(s) iniciando con el lado 0, pista 0, sector 1. La información que el DOS almacena y mantiene en su área de sistema es utilizada para determinar, por ejemplo, la ubicación de cada archivo que será accesado. Los tres componentes del área de sistema son:

1. Registro de arranque
2. Tabla de asignación de archivos (FAT)
3. Directorio

El área de sistema y el área de datos están acomodados así:




### Sectores asignados para el área de sistema

La lista siguiente da la organización de varios tipos de dispositivos y muestra los números de los sectores de inicio y final para el registro de arranque, la FAT y el directorio (sectores en términos de número de sector relativo, en donde el sector relativo 0 es cilindro 0, pista 0, sector 1, el primer sector del dispositivo):

| Dispositivo | Arranque | FAT | Directorio | Sector/Grupo |
|---|---|---|---|---|
| 5.25" 360KB | 0 | *1-4* | 5-11 | 2 |
| 5.25" 1.2MB | 0 | 1-14 | 15-28 | 1 |
| 3.5" 720KB | 0 | 1-6 | 7-13 | 2 |
| 3.5" 1.44MB | 0 | 1-18 | 19-32 | 1 |

En los discos duros, las ubicaciones del registro de arranque y de la FAT por lo común son las mismas que para un disco flexible; el tamaño de la FAT y la ubicación del directorio varían por dispositivo.

### Área de datos

El *área de datos* en un disco o disco flexible de arranque empieza con los archivos del sistema DOS llamados IOSYS.SYS y MSDOS.COM (para MS-DOS) o IBMBIO.COM e IBMDOS.COM (para IBM PC DOS). Cuando utiliza FORMAT /S para formatear un disco, el DOS copia sus archivos de sistema a los primeros sectores del área de datos. Los archivos del usuario siguen de manera inmediata a los archivos de sistema o, si no existen archivos de sistema, empiezan al inicio del área de datos.

Un disco flexible de dos lados formateado con nueve sectores por pista contiene la información siguiente:

| | |
|---|---|
| Lado O, pista O, sector 1 | Registro de arranque |
| Lado O, pista O, sectores 2-3 | Tabla de asignación de archivos (FAT) |
| Lado O, pista O, sectores 4-7 | Directorio |
| Lado 1, pista O, sectores 1-3 | Directorio |
| Lado 1, pista O, sectores 4 y sig. | Área de datos |

Los registros para los archivos de datos empiezan en el lado 1, pista O, sector 3 al 9. El sistema almacena los registros siguientes en el lado O, pista 1, después en el lado 1, pista 1, después lado O, pista 2, y así sucesivamente. Esta característica de llenado de datos en pistas opuestas (en el mismo cilindro) antes de proceder con el siguiente cilindro reduce el movimiento de la cabeza del disco y es el método usado en ambos tipos de discos, flexibles y duros.

Para otros dispositivos, la FAT y el directorio pueden ser de longitudes diferentes. Las secciones siguientes estudian en detalle el registro de arranque, directorio y la FAT.

## REGISTRO DE ARRANQUE

El registro de arranque contiene las instrucciones que cargan (o "arrancan") los archivos del sistema IOS YS. SYS, MSDOS.COM y COMMAND.COM (si está presente) desde el disco a la memoria. Todos los discos formateados tienen este registro aun cuando no estén almacenados en ellos los archivos de sistema. El registro de arranque contiene la información siguiente, en orden de dirección de desplazamiento:

| | |
|---|---|
| OOH | Salto cercano o lejano a la rutina de arranque en el desplazamiento 1EH o 3EH en el registro de arranque |
| 03H | Nombre del fabricante y número de la versión del DOS cuando fue creado el arranque |
| OBH | Bytes por sector, por lo común 200H (512) |
| ODH | Sector por grupo (1, 2, 4 u 8) |

OEH Sectores reservados
10H Número de copias de la FAT (1 o 2)
11H Número de entradas en el directorio raíz
13H Si el volumen es menor que 32 MJ3, número total de sectores
15H Byte de descripción de medio (igual al primer byte de la FAT, descrita posteriormente)
16H Número de sectores para la FAT
18H Número de sectores por pista
1AH Número de cabezas (lados o superficies) de lectura/escritura
1CH Número de sectores ocultos
1EH Cargador de la rutina de arranque para las versiones del DOS hasta la 3.3
20H Si el volumen es menor que 32 MB, número total de sectores
24H Número de unidad física (para discos flexibles, A = 0; para disco duro, 80H = unidad C, etc.)
25H Reservado por el DOS
26H Sector ampliado de arranque (contiene 29H)
27H Identificación del volumen
2BH Etiqueta del volumen
36H Reservado por el DOS
3EH-1FFH A partir de DOS 4.0, el cargador de arranque inicia aquí.

El DOS 4.0 amplía el registro de arranque con campos adicionales desde 20H hasta 1FFH. Por tanto, el registro original de arranque es de 20H (32) bytes, mientras que la versión ampliada es de 200H (512) bytes.

## DIRECTORIO

Todos los archivos en un disco empiezan en una frontera de grupo, que es el primer sector del grupo. Para cada archivo, el DOS crea una entrada de directorio de 32 (20H) bytes que describe el nombre del archivo, la fecha en que fue creado, su tamaño y la ubicación de su grupo inicial. Las entradas del directorio tienen el formato siguiente:

**BYTE** **PROPÓSITO**

00H-07H Nombre del archivo, como es definido en el programa que crea el archivo. El primer byte del nombre del archivo también puede indicar el estado del archivo:
- 00H El archivo nunca ha sido utilizado
- 05H Actualmente el primer carácter del nombre del archivo es E5H
- 2EH La entrada es para un subdirectorio
- E5H El archivo ha sido borrado

08H-0AH Extensión del nombre del archivo

OBH Atributo del archivo, define el tipo de archivo (observe que un archivo puede tener más de un atributo):
- 00H Archivo normal
- 01H Archivo que sólo puede ser leído (sólo lectura)
- 02H Archivo oculto, en una búsqueda de directorio no se despliega

04H Archivo del sistema DOS, no desplegado por una búsqueda de directorio
08H Etiqueta de volumen (si éste es un registro de etiqueta de volumen, la etiqueta misma está en los campos de nombre y extensión del archivo)
10H Subdirectorio
20H Archivo resguardado, que indica si el archivo fue reescrito en su última actualización

Como ejemplo, el código 07H significaría un archivo del sistema (04H) que es de sólo lectura (01H) y oculto (02H).

0CH-15H Reservado por el DOS.

16H-17H Hora del día en que el archivo fue creado o actualizado por última vez; almacenada con 16 bits en formato binario como |hhhhhmmmmmmssssss|.

18H-19H Fecha de creación o última actualización del archivo, almacenada con 16 bits en formato binario como |yyyyyyym¡mmmddddd|. El año puede ser 0-119 (suponiendo 1980 como el punto de inicio), el mes puede ser 01-12 y el día puede ser 01-31.

1AH-1BH Grupo inicial del archivo. El número es relativo a los dos últimos sectores del directorio. En donde no existen archivos de sistema DOS, el primer archivo de datos inicia en la unidad de asignación relativa 002. El lado, pista y grupo reales dependen de la capacidad del disco. Una entrada cero significa que no hay espacio asignado para él.

1CH-1FH Tamaño del archivo en bytes. Cuando crea un archivo, el DOS calcula y almacena su tamaño en este campo.

Para campos numéricos que excedan un byte en el directorio, los bytes son almacenados en secuencia inversa de bytes.

## TABLA DE ASIGNACIÓN DE ARCHIVOS

El objetivo de la FAT es asignar espacio en disco para los archivos. La FAT contiene una entrada para cada grupo en el disco. Cuando crea un archivo nuevo o revisa un archivo existente, el DOS revisa las entradas asociadas de la FAT de acuerdo con la ubicación del archivo en el disco. La FAT empieza en el sector 2, inmediatamente después del registro de arranque. En un disco en donde un grupo consta de cuatro sectores, el mismo número de entradas de la FAT puede hacer referencia a cuatro veces la información que los discos en donde un grupo consiste en un sector. En consecuencia, el uso de grupo con sectores múltiples reduce el número de entradas en la FAT y permite al DOS direccionar un espacio de almacenamiento mayor en disco.

### Demasiada FAT

Los diseñadores originales proporcionaron dos copias de la FAT (FAT1 y FAT2), presumiblemente porque la FAT2 podría ser utilizada si la FAT1 se dañaba. Sin embargo, aunque la FAT2 aún se mantiene, su uso nunca ha sido implementado. Todo lo tratado en este libro es acerca de la FAT1.

### Primer entrada de la FAT

El primer byte de la FAT, el descriptor de medio, indica el tipo de dispositivo (observe también el byte 15H en el registro de arranque):

| | |
|---|---|
| FOH | 3.5", dos lados, 18 sectores/pista (1.44MB) |
| FOH | 3.5", dos lados, 36 sectores/pista (2.88MB) |
| F8H | Disco duro (incluyendo disco RAM virtual) |
| F9H | 3.5", dos lados, 9 sectores/pista (720 KB) |
| F9H | 5.25", dos lados, 15 sectores/pista (1.2 MB) |
| FCH | 5.25", un lado, 9 sectores/pista (180 MB) |
| FDH | 5.25", dos lados, 9 sectores/pista (360 MB) |
| FFH | 5.25", dos lados, 8 sectores/pista (320 MB) |

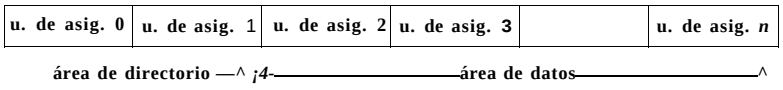
Observe que FOH y F9H identifican, cada uno, a dos diferentes formatos de disco.

### Segunda entrada de la FAT

La segunda entrada de la FAT contiene FFFFH para FAT de discos flexibles que permiten el uso de entradas de la FAT de 12 bits y FFFFFFH para discos duros que permiten el uso de entradas de la FAT de 16 bits. Las primeras dos entradas de la FAT se ven así:

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Disco flexible 1.44MB** | F0 | FF | FF | | | | | | | |
| **Disco duro** | F8 | FF | FF | FF | | | | | | |

Como ya se describió, el primer campo en un disco es el registro de arranque, seguido por la FAT y después por el directorio. El área de datos está después. La ilustración completa es como sigue:




Esperaría que el área de datos estuviera en el punto inicial para los grupos, pero en lugar de eso, los primeros dos números del grupo (0 y 1) apuntan al directorio, de modo que el área de datos para almacenar archivos de datos empieza en el grupo número 2. La razón de este estado de cosas pronto **se** verá clara.

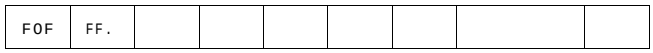
### Apuntadores a entradas de apuntadores en la FAT

Después de las primeras dos entradas de la FAT están *las entradas de apuntadores* que relacionan a cada grupo en el área de datos. El directorio (en 1AH-1BH) contiene la ubicación del primer grupo para un archivo y la FAT contiene una cadena de entradas de apuntadores para cada uno de los grupos que le suceden.

Desde el DOS 3.0, la longitud de la entrada para discos flexibles es de tres dígitos hexadecimales (IVi byte, o 12 bits) pero para disco duro es de cuatro dígitos hexadecimales (2 bytes o 16 bits). Cada entrada de apuntador de la FAT indica el uso de un grupo particular de acuerdo con el formato siguiente:

| 12 bits | 16 bits | Explicación |
|---|---|---|
| 000 | 0000 | Grupo referenciado, actualmente sin usarse |
| nnn | nnnn | Número relativo al grupo que sigue de un archivo |
| FF0-FF6 | FFF0-FFF6 | Grupo reservado |
| FF7 | FFF7 | No se puede usar (pista dañada) |
| FFF | FFFF | Último grupo del archivo |

Las primeras dos entradas en un disco flexible de 1.44MB (12 bits) se ven así:

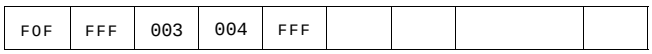



El término "grupo relativo" se refiere al grupo al cual apunta la entrada de la FAT. En cierto sentido, las primeras dos entradas de la FAT (0 y 1) apuntan a los dos últimos grupos en el directorio, que han sido asignados como el inicio de las unidades de asignación; el directorio indica el tamaño y grupo inicial de los archivos.

El directorio contiene el número de inicio del grupo para cada archivo y una cadena de entradas del apuntador del FAT, éste indica la ubicación donde continúa, si existe, el siguiente grupo. Un apuntador de entrada que contiene (F) FFFH indica el último grupo del archivo.

### Entradas muestra de la FAT

Ahora examinemos un ejemplo de entradas FAT que ayudaran a clarificar la estructura de la FAT. Suponga que un disco flexible contiene un solo archivo, de nombre CUSTOMER.FIL, que está almacenado por completo en los grupos 2, 3 y 4. La entrada del directorio para este archivo contiene: el nombre del archivo CUSTOMER, la extensión FIL, OOH para indicar que es un archivo normal, la fecha de creación, 0002H para la ubicación del primer grupo relativo del archivo y una entrada para el tamaño, en bytes, del archivo. La entrada de 12 bits de la FAT aparecerá como sigue, sólo que el par de bytes estarían en orden inverso:




Para las dos primeras entradas de la FAT, F0 indica un disco de dos caras con nueve sectores (1.44MB), seguido por FFFFH. Para leer CUSTOMER.FIL del disco y enviarlo a memoria, el sistema realiza los pasos siguientes:

- En el directorio del disco, busca el nombre de archivo CUSTOMER y la extensión FIL. El DOS extrae del directorio la ubicación del primer grupo relativo (2) del archivo y envía su contenido (datos de los sectores) al programa en la memoria principal.
- Accesa la entrada de apuntador de la FAT que representa el grupo relativo 2. Según el diagrama, esta entrada contiene 003, lo que significa que el archivo continúa en el grupo relativo 3. El DOS envía el contenido de este grupo al programa.
- Accesa la entrada de apuntador de la FAT que representa el grupo relativo 3. Esta entrada contiene 004, y significa que el archivo continúa en el grupo relativo 4. El DOS envía el contenido de este grupo al programa.

La entrada de la FAT para el grupo relativo 4 contiene FFFH para indicar que no hay más grupos asignados para este archivo. Ahora el DOS ya ha enviado todos los datos del archivo, desde los grupos 2, 3 y 4.

Sólo hemos visto el principio de funcionamiento de las entradas de la FAT; ahora veamos cómo funcionan en términos de secuencia inversa de bytes, en donde se requiere un poco más de ingenio.

### Manejo de entradas de 12 bits de la FAT en secuencia inversa de bytes

Continuemos con el mismo ejemplo de las entradas de la FAT para CUSTOMER.FIL, que se acaba de ver, pero ahora con entradas de apuntador en secuencia inversa de byte. Para este archivo la FAT de 12 bits se ve de esta manera:

| Entrada de la FAT: | FOF | FFF | 034 | 000 | FFO | F'xx | |
|---|---|---|---|---|---|---|---|
| U. de asignación relativa: | 0 | 1 | 2 | 3 | 4 | 5 | |

Pero ahora es preciso —para descifrar las entradas— representarlas según el *byte relativo,* y no el grupo:

| Entrada de la FAT: | FO | FF | FF | 03 | 40 | 00 | FF | OF | |
|---|---|---|---|---|---|---|---|---|---|
| Byte relativo: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | |

Para procesar la primer entrada en la FAT:

- Multiplique 2 (el primer grupo del archivo) por 1.5 (la longitud de las entradas de la FAT) para obtener 3. (Para programación, multiplique por 3 y recorra un bit a la derecha.) Accese la palabra de la FAT en los bytes 3 y 4. Estos contienen 03 40, los cuales se convierten, en orden inverso, en 4003. Ya que el grupo 2 fue un número par, utilice los últimos tres dígitos, de modo que 003 sea el segundo grupo para el archivo.
- Para el tercer grupo, multiplique el número de grupo (3) por 1.5 para obtener 4. Accese los bytes 4 y 5 de la FAT. Estos contienen 40 00, que se convierten en 00 40. Como el grupo 3 fue un número impar, utilice los primeros tres dígitos, así que 004 es el tercer grupo para el archivo.
- Para el cuarto grupo, multiplique 4 por 1.5 para obtener 6. Accese los bytes 6 y 7 de la FAT. Estos contienen FF 0F, que se convierten, en orden inverso, en 0FFF. Como el grupo 4 fue un número par utilice los últimos tres dígitos, FFF, lo cual significa que ésta es la última entrada. (¡Vaya!)

### Manejo de entradas de 16 bits de la FAT

Como hemos dicho, después del descriptor de medio para el disco duro está FFFFFFH. Los apuntadores a las entradas en la FAT son de 16 bits de longitud y empiezan con los bytes 3 y 4, que representan el grupo 2. La entrada del directorio proporciona los grupo iniciales para los archivos, y una entrada de apuntador FFFFH indica fin de archivo. La determinación del número de grupo para cada entrada de la FAT es sencilla, aunque los bytes en cada entrada están en orden inverso.

Como ejemplo de entradas de 16 bits de la FAT, suponga que el único archivo en un disco duro en particular ocupa cuatro grupos (en 4 sectores por grupo, o 16 sectores en total). De acuerdo el directorio, el archivo inicia en el grupo 2. Cada entrada de apuntador es una palabra completa, así que invertir los bytes sólo implica la entrada uno. Aquí está la FAT, con entradas de apuntador en secuencia inversa de bytes:

| Entrada de la FAT: | F8FF | FFFF | 0300 | 0400 | 0500 | FFFF |
|---|---|---|---|---|---|---|
| U. de asignación relativa: | | | | | | |

La entrada de la FAT para el grupo relativo 2, 0300, se invierte como 0003 para el grupo siguiente. La entrada de la FAT para el grupo relativo 3, 0400, se invierte como 0004 para el grupo

siguiente. Continúe con la cadena de entradas restantes de esta manera hasta la entrada para el número de grupo 5.

Si su programa ha determinado el tipo de disco que está instalado, puede verificar el descriptor de medio directamente en el sector de arranque o, de preferencia, emplear la función 1BH o 1CH de la INT 21H del DOS.

## EJERCICIO QUE IMPLICA EL USO DE LA FAT

Usemos DEBUG para examinar la FAT de un disco. Para este ejercicio, necesita dos discos flexibles de 3.5 pulgadas en blanco, formateados con capacidades de 720 KB y 1.44MB. Copie dos archivos en cada disco. El primer archivo debe ser mayor de 512 bytes y menor de 1,024 bytes, para que quepa en dos sectores; se sugiere el archivo P04ASM1.ASM. El segundo archivo debe ser mayor de 1,536 bytes y menor de 2,048 bytes, para colocarlo en cuatro sectores; se sugiere el archivo P10DRVID.ASM. Verá que las FAT de los dos discos son similares, pero no idénticas.

Discos de 720K. Primero inserte el disco de 720K en la unidad A (o en la B si es necesario). Cargue DEBUG e introduzca el comando L (cargar) (explicado por completo en el apéndice E):

```
L 100 0 0 20 (para la unidad B, utilice L 100 1 0 20)
```

Las entradas del comando L son:

- 100H es el desplazamiento inicial en el segmento de DEBUG.
- El primer 0 significa utilizar la unidad A (o 1 para la unidad B).
- El segundo 0 significa leer datos empezando con el sector relativo 0.
- 20 significa leer 20H (32) sectores.

Ahora puede examinar el registro de arranque, el directorio y la FAT para este disco. Para desplegar el sector de arranque, ingrese el comando D 100. Observe que algunos campos son:

- El segmento con desplazamiento 103H muestra el nombre del fabricante y la versión del DOS cuando fue creada la FAT.
- 10BH muestra el número de bytes por sector (en donde 0002H en orden inverso de bytes es 0200H, o 512 bytes).
- 115H es el descriptor de medio, F9H para este disco flexible.
- Examine los otros campos.

Encontrará el directorio en F00H:

- F00H muestra el nombre del archivo para el primer archivo, P04ASM1.ASM.
- F1AH da el número del grupo inicial para este archivo (0200 o 0002).
- F1CH-F1FH da el tamaño, en bytes, del archivo.
- F20H empieza la entrada para el segundo archivo, P10DRVID.ASM. Observe que F3AH muestra su grupo inicial como 0300 o 0003.

Encontrará que la FAT en 300H se parece a:

| Entrada de la FAT: | F9 | FF | FF | FF | 4F | 00 | FF | OF | |
|---|---|---|---|---|---|---|---|---|---|
| Byte relativo: | | | | | | | | | |

- F9 es el descriptor de medio.
- FF FF en bytes 1 y 2 es el contenido del segundo campo.

Los apuntadores a entradas que inician en el byte 3 pueden ser calculados como:

- Para el primer archivo, multiplique 2 (su primer grupo) por 1.5 para obtener 3. Accese los bytes con desplazamiento 3 y 4 en la FAT, que contienen FF 4F, e invierta los bytes para obtener 4FFF. Como el grupo 2 fue un número par, utilice los últimos tres dígitos, FFF, que le indican que no existen más grupos para este archivo.
- Para el segundo archivo, multiplique por 3 (su primer grupo) por 1.5 para obtener 4. Accese los bytes con desplazamiento 4 y 5 en la FAT, que contienen 4F 00, e invierta los bytes para obtener 004F. Como el grupo 3 fue un número impar, utilice los primeros tres dígitos, 004, que identifican el grupo siguiente en la serie. Multiplique el número de grupo, 4, por 1.5 para obtener 6. Accese los bytes con desplazamiento 6 y 7 en la FAT, los cuales contienen FF 0F, e invierta los bytes para obtener 0FFF. Como el grupo 4 fue un número par, utilice los primeros tres dígitos, FFF, que indican el final de los datos.

Discos de 1.44MB. Ahora inserte el disco flexible de 1.44MB en la unidad A, e ingrese el comando L 100 0 0 30 de DEBUG. (Carga 30H sectores porque hay más FAT en los discos de 1.44MB.) Despliegue el registro de arranque de este disco y observe que el byte del descriptor de medio en 115H es F0 y el número de sectores por grupo (en 10DH) es 1. Los directorios en 2700H y 2720H deben mostrar que el grupo inicial para el primer archivo es 2 y para el segundo es 4. (El grupo inicial para el segundo archivo en el disco de 720 K fue 3 porque el formato tiene dos sectores por grupo.)

Despliegue la FAT en 300H, que aparece como

| Entrada de la FAT: | F0 | FF | FF | 03 | F0 | FF | 05 | 60 | 00 | 07 | F0 | FF |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Byte relativo: | | | | | | | | | | | 10 | 11 |

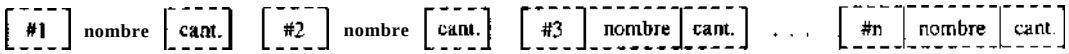
Ya que el primer archivo inicia en el grupo 2, multiplique 2 por 1.5 para obtener el byte relativo 3. Los bytes 3 y 4 contienen 03 F0, que se invierten como F003. Como el grupo 2 fue un número par, utilice los últimos tres dígitos, 003. Grupo 3 x 1.5 es 4; los bytes relativos 4 y 5 contienen F0 FF, que se invierten como FFF0. Puesto que el grupo 3 fue un número impar, utilice los primeros tres dígitos, FFF, que indican que el archivo ya no continúa. Ahora sabemos que el archivo reside en los grupos 2 y 3.

Utilice la misma técnica para seguir la pista de la cadena del segundo archivo, que inicia con el grupo 4, byte relativo 6.

El DOS proporciona algunos servicios de soporte de programas para accesar información acerca del directorio y de la FAT. Las funciones 47H (obtener el directorio actual) y 1BH y 1CH (obtener información de la FAT) son descritas en el capítulo 18.

## PROCESAMIENTO DE ARCHIVOS EN DISCO

Los datos en el disco son almacenados en la forma de un *archivo,* igual como ha almacenado sus programas. Aunque no existe restricción sobre la clase de información que pueda guardar en un archivo, un archivo típico de usuario consistiría en *registros* para clientes, partes de inventario o una lista de nombre y dirección. Cada registro contiene información acerca de un cliente o elemento de inventario en particular. Dentro de un archivo, por lo común, todos los registros son del mismo tamaño y formato. Un registro contiene uno o más campos que proporcionan información acerca del registro. Por ejemplo, los registros para un archivo de clientes podrían contener campos como número de cliente, nombre del cliente y cantidad que debe. Los registros podrían estar en orden ascendente por número de cliente, como sigue:




El procesamiento de archivos en disco duro es similar al de discos flexibles, tiene que proporcionar un nombre de ruta de acceso al archivo en los subdirectorios.

### Servicios de interrupción para Entrada/Salida a disco

Varios servicios de interrupción especiales permiten la entrada/sal ida a disco. Un programa que escribe un archivo primero *crea* el archivo de manera que el DOS pueda generar una entrada para él en el directorio. Cuando todos los registros del archivo han sido escritos, el programa *cierra* el archivo de modo que el DOS puede completar la entrada del directorio para el tamaño del archivo.

Un programa que lee un archivo primero lo *abre* para asegurar que existe. Una vez que todo el archivo ha sido leído, la práctica es cerrarlo, dejándolo disponible para otros programas. A causa del diseño del directorio, usted puede procesar registros en un archivo en disco ya sea en forma secuencial (un registro después de otro, en forma sucesiva) o en forma aleatoria (registros, en el orden que sean necesarios a lo largo del archivo).

El *nivel más alto* de procesamiento de disco es mediante la interrupción 21H del DOS, que permite el acceso a disco por medio de un directorio y "bloque" y "desbloqueo" de registros. El método del DOS lleva a cabo algún procesamiento preliminar antes de enlazar al BIOS. El capítulo 17 trata el uso de las operaciones del DOS para escribir y leer archivos en disco, y el capítulo 18 estudia varias operaciones del DOS que dan soporte a directorios y archivos.

El *nivel más bajo* de procesamiento en disco es por medio de la interrupción 13H del BIOS, que implica direccionamiento directo de números de pista y sector. Esta operación se estudia en el capítulo 19.

## PUNTOS CLAVE

- Cada lado de un disco flexible o de un disco duro contiene varias pistas concéntricas, iniciando con la pista número 00. Cada pista está formateada en sectores de 512 bytes, iniciando en el sector número 1.
- Un cilindro es el conjunto de todas las pistas con el mismo número en cada lado.
- Un grupo es un grupo de sectores que el DOS trata como una unidad de espacio de almacenamiento. El tamaño de un grupo siempre es una potencia de 2, como 1, 2, 4 u 8 sectores. Un archivo inicia en una frontera de grupo y necesita por lo menos un grupo.

- Sin importar el tamaño, todos los archivos inician en una frontera de grupo.
- El registro de arranque contiene las instrucciones que cargan (o "arrancan") los archivos de sistema IOSYS.SYS, MSDOS.COM y COMMAND.COM del disco a la memoria.
- El directorio contiene una entrada para cada archivo en un disco e índica el nombre del archivo, la extensión, atributo de archivo, hora, fecha, sector de inicio y tamaño del archivo.
- El propósito de la tabla de asignación de archivos (FAT) es asignar espacio en disco para los archivos. La FAT inicia en el sector 2 inmediatamente después del registro de arranque y contiene una entrada por cada grupo para cada archivo en el directorio.

## PREGUNTAS

**16-1.** ¿Cuál es la longitud, en bytes, de un sector estándar?

**16-2.** ¿Qué es un cilindro?

**16-3.** ¿Cuál es el objetivo del controlador de disco?

**16-4.** (a) ¿Qué es un grupo? (b) ¿Cuál es su objetivo? (c) Un archivo tiene 48 bytes. ¿Cuál es el espacio en disco para unidades de asignación de 1, 2, 4 y 8?

**16-5.** Muestre cómo calcular la capacidad de un disco flexible, con base en el número de cilindros, sectores por pista y bytes por sector, para (a) un disco flexible de 5.25", 360 KB y (b) un disco flexible de 3.5". 1.44MB.

**16-6.** ¿Qué contiene el área de sistema en disco?

**16-7.** (a) ¿En dónde está ubicado el registro de arranque? (b) ¿Cuál es su propósito?

**16-8.** En el directorio, ¿cuál es la iniciación para un archivo borrado?

**16-9.** En el directorio, ¿cuál es la indicación para (a) un archivo normal; (b) un archivo oculto?

**16-10.** Cuando utiliza FORMAT /S para formatear, ¿cuál es el efecto adicional en un disco flexible y en un disco duro?

**16-11.** Considere un archivo de tamaño de 2,890 (decimal) bytes. (a) El sistema, ¿en dónde almacena el tamaño? (b) En formato hexadecimal, ¿cuál es el tamaño? Muestre el número como lo almacena el sistema.

**16-12.** ¿Dónde y cómo indica la FAT que el dispositivo en el cual reside es (a) un disco duro; (b) un disco flexible de 5.25", 360KB; (c) un disco flexible de 3.5", 1.44MB?

## CAPÍTULO 17

# Procesamiento en disco: I—Escritura y lectura de archivos

OBJETIVO

**Estudiar el uso de manejadores de archivos y las funciones del DOS para la escritura y lectura de archivos en disco, tanto secuenciales como de acceso directo (aleatorio).**

## INTRODUCCIÓN

Los servicios originales del DOS para procesamiento de archivos en disco usaban un método llamado bloques de control de archivo (FCB). Este método, aunque el DOS le da soporte, puede direccionar unidades y nombres de archivo pero no subdirectorios. Las versiones subsecuentes del DOS introdujeron varios servicios ampliados más simples que sus contrapartes originales, y en general son recomendables. Algunas de estas operaciones implican el uso de una cadena ASCIIZ para identificar inicialmente una unidad, una rufa y un nombre de archivo; un manejador de archivo para acceso subsecuente del archivo; y un código de regreso especial para identificar errores. Como un recordatorio, el término *grupo* denota a un conjunto de uno o más sectores de datos, dependiendo del dispositivo.

Aunque no se requiere ninguna instrucción de lenguaje ensamblador, este capítulo introduce varios servicios de la interrupción 21H del DOS para procesamiento de archivos en disco. Aqui están ordenadas por categoría:

| OPERACIONES QUE USAN MANEJADORES DE ARCHIVO | | OPERACIONES QUE USAN FCB | |
|---|---|---|---|
| 3CH | Crea archivo | OFH | Abre archivo |
| 3DH | Abre archivo | 10H | Cierra archivo |
| 3EH | Cierra archivo | 14H | Lee registro |

| | | | |
|---|---|---|---|
| 3FH | Lee registro | 15H | Escribe registro |
| 40H | Escribe registro | 16H | Crea archivo |
| 42H | Mueve apuntador de archivo | 21H | Lectura directa de registro |

| OTROS SERVICIOS DEL DOS | | 22H ESCRITURA DIRECTA DE REGISTRO | |
|---|---|---|---|
| INT 25H | Lectura absoluta | 27H | Lectura directa de bloque |
| INT 26H | Escritura absoluta | 28H | Escritura directa de bloque |

El capítulo trata los servicios del DOS para escribir y leer archivos en disco. El capítulo 18 cubre los diferentes servicios que se requieren para el manejo de unidades de disco, directorios y archivos.

## CADENAS ASCIIZ

Cuando emplee muchos de los servicios ampliados para procesamiento de disco, primero indique al DOS la dirección de un *cadena ASCIIZ* que contenga la ubicación del archivo: unidad de disco, ruta al directorio y nombre de archivo (todas opcionales y entre apóstrofos), seguidos por un byte de ceros hexadecimales; de ahí el nombre de cadena ASCIIZ *(zeros,* en inglés). La longitud máxima de la cadena es de 128 bytes.

El código siguiente define una unidad y un nombre de archivo:

```
PATHNMl DB 'D:\TEST.ASM',00H
```

Este código define una unidad, subdirectorio y nombre de archivo:

```
PATHNM2  DB 'D:\UTILITY\NU.EXE',00H
```

La diagonal inversa también puede ser diagonal derecha, actúa como un separador de ruta. Un byte de ceros termina la cadena. Para interrupciones que necesiten de una cadena ASCIIZ, cargue su dirección de desplazamiento en el registro DX por ejemplo, como

```
LEA DX,PATHNAME.
```

## MANEJADORES DE ARCHIVO

Como se estudió en el capítulo 9, puede usar manejadores de archivos de manera directa para ciertos dispositivos estándar: 00 = entrada, 01 = salida, 02 = error de salida, 03 = dispositivo auxiliar y 04 = impresora. Muchos de los servicios del DOS también implican el uso de un manejador de archivo para operaciones que accesan archivos, y usted debe solicitar un número de manejador de archivo desde el DOS. Un archivo en disco primero debe ser abierto; a diferencia de la transferencia de información desde el teclado o a la pantalla, el DOS tiene que direccionar los archivos en disco por medio de su directorio y entradas de la FAT y debe actualizar estas entradas. Durante la ejecución del programa, cada archivo referenciado debe ser asignado a su propio y único manejador de archivo.

El DOS envía un manejador de archivo cuando usted abre un archivo para entrada o crea un archivo para salida. Las operaciones implican el uso de una cadena ASCIIZ y la función 3CH o 3DH del DOS. El manejador de archivo es un número único de una palabra regresado en el AX que usted guarda en un elemento del área de datos y que utiliza para todas las peticiones subsecuentes de acceso al archivo. En forma común, el primer manejador de archivo que se regresa es 05, el segundo es 06 y así sucesivamente.

El PSP contiene una tabla de manejadores de archivo por omisión que proporciona 20 manejadores (y por tanto el límite nominal para archivos abiertos), pero la función 67H de la INT 21H puede ser usada para aumentar el límite, como se explica en el capítulo 24.

## CÓDIGOS DE ERROR DE REGRESO

Las operaciones de manejador de archivo en disco regresan un *estado de terminación* por medio de la bandera de acarreo y el registro AX. Una operación *exitosa* pone en cero la bandera de acarreo y realiza otras funciones apropiadas. Una operación *no exitosa* pone en uno a la bandera de acarreo y regresa un código de error en el AX, dependiendo de la operación. La figura 17-1 lista los códigos de error 01-36; otros códigos se refieren a redes.

Si estos errores no son suficientes, también puede usar la INT 59H para información adicio nal acerca de errores (véase el capítulo 18).

La sección siguiente cubre los requisitos para crear, escribir y cerrar archivos en disco parí el DOS ampliado.

| | | | |
|---|---|---|---|
| 01 | Número no válido de función | 20 | Unidad desconocida |
| 02 | Archivo no encontrado | 21 | Unidad no preparada |
| 03 | Ruta no encontrada | 22 | Comando desconocido |
| 04 | Demasiados archivos abiertos | 23 | Error de datos CRC |
| 05 | Acceso denegado | 24 | Longitud incorrecta en la estructura |
| 06 | Manejador no válido | 25 | Error de búsqueda |
| 07 | Bloque de control de memoria destruido | 26 | Tipo de medio desconocido |
| 08 | Memoria insuficiente | 27 | Sector no encontrado |
| 09 | Dirección no válida de bloque de memoria | 28 | Impresora sin papel |
| 10 | Ambiente (entorno) no válido | 29 | Falla al escribir |
| 11 | Formato no válido | 30 | Falla al leer |
| 12 | Código de acceso no válido | 31 | Falla generalizada |
| 13 | Datos no válidos | 32 | Violación de compartición |
| 15 | Especificación de unidad no válida | 33 | Violación de bloqueo |
| 16 | Intento de eliminar un directorio | 34 | Cambio no válido de disco |
| 17 | No es el mismo dispositivo | 35 | FCB no disponible |
| 18 | No hay más archivos | 36 | Desbordamiento del búfer compartido |
| 19 | Disco protegido contra escritura | | |

Figura 17-1 Códigos principales de retorno de error en disco.

## APUNTADORES DE ARCHIVO

El DOS mantiene un *apuntador de archivo* separado para cada archivo que un programa est procesando. Las operaciones de crear y abrir fijan el valor del apuntador de archivo en cero, 1 posición inicial del archivo. El apuntador de archivo toma en cuenta de manera subsecuente ( desplazamiento de la ubicación actual dentro del archivo.

Cada operación de lectura/escritura hace que el DOS incremente el apuntador de archivo pe el número de bytes transferidos. Entonces, el apuntador de archivo apunta a la posición del registr siguiente a ser accesado. Los apuntadores de archivo facilitan tanto el procesamiento secuenci; como el directo. Para procesamiento directo, puede usar la función 42H del DOS (estudiada en ] última sección) para colocar el apuntador de archivo en cualquier posición en un archivo.

## USO DE MANEJADORES DE ARCHIVO PARA CREAR ARCHIVOS EN DISCO

El procedimiento para escribir en un archivo es el siguiente:

**1.** Utilice una cadena ASCIIZ para obtener manejador de archivo del DOS.

2. Utilice la función 3CH del DOS para crear el archivo.
3. Utilice la función 40H del DOS para escribir registros en el archivo.
4. Al final, utilice la función 3EH del DOS para cerrar el archivo.

INT 21H, función 3CH: Crea archivo

Para *crear* un archivo nuevo o sobreescribir en uno ya creado con el mismo nombre, use primero la función 3CH del DOS. Cargue el CX con el atributo de archivo requerido (estudiado en el capítulo 16) y el DX con la dirección de la cadena ASCIIZ (donde el DOS envía archivo nuevo). Aquí está un ejemplo que crea un archivo normal en la unidad D con atributo 0:

```
PATHNMl  DB   'D:\ACCOUNTS.FIL' , 00H
HANDLE1  DW   ?

         MOV  AH,3 CH          ;Petición para crear archivo
         MOV  CX, 0 0          ,-Atributo normal
         LEA  DX,PATHNMl       ;Cadena ASCIIZ
         INT  21H              ;Llama al DOS
         JC   error            ;Si hay error, sale
         MOV  HANDLE1,AX       ;Guarda el manejador en una palabra
```

Para operación válida, el DOS crea una entrada de directorio con el atributo dado, limpia la bandera de acarreo y establece el manejador para el archivo en el AX. Utilice este manejador de archivo para todas las operaciones subsecuentes del disco. El archivo nombrado está abierto con su apuntador de archivo fijado en cero y ahora está preparado para escritura. Si un archivo con el nombre dado ya existe en la ruta, la operación configura a cero la longitud para sobreescribir el nuevo archivo en el anterior.

En condición de error, la operación pone en uno la bandera de acarreo y regresa un código en el AX: 03, 04 o 05 (véase la figura 17-1). El código 05 significa que o bien el directorio está lleno o bien el nombre de archivo referenciado tiene el atributo de sólo lectura. Asegúrese de examinar primero la bandera de acarreo. Por ejemplo, la creación de un archivo tal vez envía el manejador 05 al AX, lo cual con facilidad puede ser confundido con el código de error 05, acceso denegado. Los servicios relacionados para crear un archivo son 5AH y 5BH, estudiados en el capítulo 18.

INT 21H, función 40H: Escribe registro

Para *escribir* registros en disco, utilice la función 40H del DOS. Cargue el BX con el manejador de archivo almacenado, el CX con el número de bytes a escribir y el DX con la dirección del área de salida. El ejemplo siguiente utiliza el manejador de archivo de operación de creación precedente para escribir un registro de 256 bytes desde OUTREC:

```
HANDLEl   DW    ?
OUTREC    DB    255 DUP(' ')    ;Área de salida

          MOV   AH,4 0H         Petición para escribir registro
          MOV   BX,HANDLEl      Manejador de archivo
          MOV   CX,256          Longitud del registro
          LEA   DX,OUTREC       Dirección del área de salida
          INT   21H             Llama al DOS
```

```
JC     error2          /Prueba por error
CMP    AX,256          /¿Se han escrito todos los bytes?
JNE    error3
```

Una operación válida escribe los registros en el disco, incrementa el apuntador de archivo y hace el AX igual al número de bytes realmente escritos. Un disco lleno puede causar que el número de escritos difiera del número requerido, aunque el DOS no reporta esta condición como error. Una operación no válida pone en uno la bandera de acarreo y regresa en el AX el código de error 05 (acceso denegado) o 06 (número de manejador no válido).

### INT 21H, función 3EH: Cierra archivo

Cuando ha terminado de escribir en un archivo, tiene que cerrarlo. Cargue el manejador de archivo en el BX y utilice la función 3EH del DOS:

```
MOV   AH,3EH        /Petición para cerrar
MOV   BX,HANDLE1    /Manejador de archivo
INT   21H           /Llama al DOS
```

Una operación correcta de cierre escribe cualquier registro restante en el búfer de la memoria y actualiza la FAT y el directorio con la fecha y tamaño del archivo. Una operación no exitosa pone en uno la bandera de acarreo y regresa el único código de error posible en el AX, 06 (manejador no válido).

### Programa: Uso de manejador de archivo para crear un archivo

El programa de la figura 17-2 crea un archivo de nombres que un usuario ingresa desde un teclado. Sus procedimientos principales son los siguientes.

- C10CREA Utiliza la función 3CH para crear el archivo y guarda el manejador en un elemento de dato llamado HANDLE.
- D10PROC Acepta entrada desde el teclado y limpia las posiciones desde el final del nombre al final del área de entrada.
- F10WRIT Utiliza la función 40H para escribir registros.
- G10CLSE Al final del procedimiento, utiliza la función 3EH para cerrar el archivo a fin de crear una entrada de directorio apropiada.

El área de entrada es de 30 bytes, seguida por 2 bytes para los caracteres Enter (ODH) y Avance de línea (OAH), para un total de 32 bytes. El programa escribe los 32 bytes como un registro de longitud fija. Puede omitir los caracteres Enter/Av. de línea, pero debe incluirlos si quiere ordenar los registros en el archivo, ya que el programa SORT del DOS necesita estos caracteres para indicar el fin de los registros. Para este ejemplo, el comando SORT para ordenar los registros de NAMEFILE.DAT en orden ascendente y dejarlo en NAMEFILE.SRT podría ser

```
SORT D:<NAMEFILE.DAT >NAMEFILE.SRT
```

El programa de la figura 17-3 lee y despliega el contenido de NAMEFILE.SRT. Observe dos puntos: (1) Los caracteres Enter/av. de línea están incluidos después de cada registro sólo para

```
TITLE     P17HANCR (EXE)  Crea en disco un archivo de nombres
          .MODEL  SMALL
          .STACK  64
-------------------------------------------------------------
          .DATA
NAMEPAR   LABEL   BYTE              | Lista de parámetros:
MAXLEN    DB      30                | Longitud máxima
NAMELEN   DB      7                 | Longitud actual
NAMEREC   DB      30 DUPC ") ODH, OAH | Nombre ingresado,
                                    |   CR/LF para escribir
ERRCDE    DB      00                | Indicador de error
HANDLE    DW      7                 | Manejador de archivo
PATHNAM   DB      'D:\NAMEFILE.DAT', 0
PROMPT    DB      'Name?'
ROW       DB      01
OPNMSG    DB      '*** Open error  ***', ODH, OAH
WRTMSG    DB      '*** Write error ***', ODH, OAH
-------------------------------------------------------------
          .CODE
BEGIN     PROC    FAR
          MOV     AX,@data          Inicializa el
          MOV     DS, AX              segmento de datos
          MOV     ES, AX
          MOV     AX,0600H
          CALL    Q10SCR            ;Limpia la pantalla
          CALL    Q2 0CURS          ,- Coloca el cursor
          CALL    C10CREA           ;Crea archivo; designa DTA
          CMP     ERRCDE,00         ;¿Error al crear?
          JZ      A2 0LOOP            sí, continuar
          JMP     A90                 no, salir
A2 0LOOP:
          CALL    DIOPROC
          CMP     NAMELEN, 00       ¿Fin de la entrada?
          JNE     A20LOOP             no, continuar
          CALL    G10CLSE             sí, cerrar,
A90 :     MOV     AX,4C00H          Salir al DOS
          INT     21H
BEGIN     ENDP
          Crea archivo en disco:
-------------------------------------------------------------
ClOCREA   PROC    NEAR
          MOV     AH,3CH            ,-Petición para crear
          MOV     CX,00             ;Normal
          LEA     DX, PATHNAM
          INT     21H
          JC      C20                ¿Hubo error?
          MOV     HANDLE, AX           no, guardar el manejador
          RET
C20 :                                  sí, desplegar
          LEA     DX,OPNMSG            mensaje de error
          CALL    X10ERR
          RET
CÍOCREA   ENDP
          Acepta entrada:
-------------------------------------------------------------
D10PROC   PROC    NEAR
          MOV     AH,40H            Petición de despliegue
          MOV     BX, 01            Manej ador
          MOV     CX, 06            Longitud de la indicación
          LEA     DX, PROMPT        •Despliega la indicación
          INT     21H

          MOV     AH, OAH           Entrada de la petición
          LEA     DX, NAMEPAR       Acepta nombre
```

Figura 17-2 Uso de manejador para crear un archivo

```
          INT     21H
          CMP     NAMELEN,00        ¿Existe un nombre?
          JZ      D90               ;  no, salir
          MOV     AL,20H            Limpiar para almacenar
          SUB     CH,CH
          MOV     CL,NAMELEN        ;Longitud
          LEA     DI,NAMEREC
          ADD     DI,CX             ;Dirección + longitud
          NEG     CX                ,-Calcula la longitud
          ADD     CX,30               restante
          REP STOSB                 ;Asigna blancos
          CALL    F10WRIT           ,•Escribe registro a disco
          CALL    E10SCRL           ,-Verifica para recorrido
D90:
          RET
D10PROC   ENDP
                  Verifica para recorrido:
                  ------------------------

E10SCRL   PROC    NEAR
          CMP     ROW,18            ;¿Se llegó a la parte inf<
          JAE     E10               ;  sí, continuar
          INC     ROW               ;  no, sumar al renglón
          JMP     E90
E10:
          MOV     AX,0601H          ;Recorrer un renglón
          CALL    Q10SCR
E90:      CALL    Q20CURS           ¡Restablecer el cursor
          RET
E10SCRL   ENDP
                  Escribe registro en disco:
                  --------------------------

F10WRIT   PROC    NEAR
          MOV     AH,40H            ;Petición de escritura
          MOV     BX,HANDLE
          MOV     CX,32             ;30 para el nombre + 2 pa
          LEA     DX,NAMEREC
          INT     21H
          JNC     F20               /¿Escritura válida?
          LEA     DX,WRTMSG         ;  no,
          CALL    X10ERR            ;  llama a la rutina de ei
          MOV     NAMELEN,00
F20:
          RET
F10WRIT   ENDP
                  Cierra archivo en disco:

G10CLSE   PROC    NEAR
          MOV     NAMEREC,1AH       ;Coloca la marca EOF
          CALL    F10WRIT
          MOV     AH,3EH            ;Petición para cerrar
          MOV     BX,HANDLE
          INT     21H
          RET
        ENDP
                  Recorre la pantalla:

Q10SCR    PROC    NEAR              ;AX establece entrada
          MOV     BH,1EH            ,-Designa amarillo sobre azul
          MOV     CX,0000
          MOV     DX,184FH
          INT     10H               ¡Recorre
          RET
Q10SCR    ENDP
```

Figura 17-2 (continuación)

```
                    Coloca el cursor:

Q20CURS   PROC    NEAR
          MOV     AH,02H          ;Petición
          MOV     BH,00             para colocar el cursor
          MOV     DH,ROW          /Renglón
          MOV     DL,00           ;Columna
          INT     10H
          RET
Q20CURS   ENDP
                    Despliega mensaje de error en disco:

X10ERR    PROC    NEAR            ,-DX contiene
          MOV     AH,40H          ,-  dirección del mensaje
          MOV     BX,01
          MOV     CX,21           ;Longitud
          INT     21H
          MOV     ERRCDE,01       ;Designa código del error
          RET
X10ERR    ENDP
          END     BEGIN
```

Figura 17-2 (continuación)

facilitar el ordenamiento y de otra forma podrían ser omitidos. (2) Los registros pueden ser de longitud variable; esto implicaría algo de programación extra, como se verá posteriormente.

## USO DE MANEJADORES DE ARCHIVO PARA LEER ARCHIVOS EN DISCO

En esta sección cubriremos los requisitos para abrir y leer archivos en disco por medio de manejadores de archivo. El procedimiento para leer un archivo en disco es el siguiente:

1. Utilice una cadena ASCIIZ para obtener un manejador de archivo del DOS.
2. Utilice la función 3DH del DOS para abrir el archivo.
3. Utilice la función 3FH del DOS para leer registros del archivo.
4. Al final, utilice la función 3EH del DOS para cerrar el archivo.

### INT 21H, función 3DH: Abre archivo

Si su programa es para leer un archivo, primero utilice la función 3DH del DOS para abrirlo. Esta operación verifica que el archivo realmente exista. Cargue el DX con la dirección de la cadena ASCIIZ necesaria y establezca el AL con el código de acceso:

| BITS | PETICIÓN |
|---|---|
| 0-2 | 000 = sólo lectura |
| | 001 = sólo escritura |
| | 010 = lectura/escritura |
| 3 | Reservada |
| 4-6 | Modo compartido |
| 7 | Bandera heredada |

Al escribir en un archivo, asegúrese de utilizar la función 3CH para crear el archivo, no la función 3DH para abrirlo. El ejemplo siguiente abre un archivo para lectura:

```
MOV   AH,3DH      ,-Petición para abrir archivo
MOV   AL,00       ,-Sólo lectura
```

```
LEA   DX,PATHNM1      Cadena ASCIIZ
INT   21H             Llama al DOS
JC    error4          Si hay error, sale
MOV   HANDLE2,AX      Guarda el manejador en una palabra
```

Si un archivo con el nombre dado existe, la operación establece la longitud del registro en uno (el cual puede pasarse por alto), asume el atributo actual del archivo, establece el apuntador de archivo a cero (el inicio del archivo), pone en cero la bandera de acarreo y establece un manejador para el archivo en el AX. Utilice este manejador de archivo para todas las operaciones subsecuentes.

Si el archivo no existe, la operación pone en uno la bandera de acarreo y regresa un código de error en el AX: 02, 03, 04, 05 o 12 (véase la figura 17-1). Asegúrese de examinar primero la bandera de acarreo. Por ejemplo, al crear un archivo tal vez envíe el manejador 05 al AX, lo cual podría con facilidad ser confundido con el código d^ error 05, acceso denegado.

### INT 21H, función 3FH: Lee registro

Para leer registros, utilice la función 3FH del DOS. Cargue el manejador de archivo en el BX, el número de bytes a leer en el CX y la dirección del área de entrada en el DX. El código siguiente lee un registro de 512 bytes:

```
DW    ?
DB    512 DUP(' ')

MOV   AH,3FH          ;Petición de lectura de registro
MOV   BX,HANDLE2      ;Manejador de archivo
MOV   CX,512          ,•Longitud del registro
LEA   DX,INPREC       ;Dirección del área de entrada
INT   21H             :Llama al DOS
JC    error5          ;Prueba por error
CMP   AX, 00          ;¿Cero bytes por leer?
JE    finarchv
```

Una operación válida envía el registro al programa, pone en cero la bandera de acarreo y establece el AX al número de bytes que en realidad se leyó. Cero en el AX significa un intento de leer después del fin de archivo; ésta es un advertencia, no un error. Una lectora no válida pone en uno la bandera de acarreo y regresa al AX el código de error 05 (acceso denegado) o 06 (manejador no válido).

Ya que DOS limita el número de archivos abiertos al mismo tiempo, un programa que de manera sucesiva lee varios archivos debe cerrarlos tan pronto como sea posible.

### Programa: Uso de manejador de archivo para leer un archivo

El programa de la figura 17-3 lee el archivo creado por el programa de la figura 17-2 y ordenad( por el comando SORT del DOS. A continuación están los procedimientos principales:

- E10OPEN Utiliza la función 3DH del DOS para abrir el archivo y guarda el manejador ei un elemento de datos llamado HANDLE.
- F10READ Emite la función 3FH del DOS, que utiliza el manejador para leer los registros

```
TITLE     P17HANRD (EXE)   Lectura secuencial de registros en disco
          .MODEL  SMALL
          .STACK  64
-------------------------------------------------------------------
          .DATA
ENDCDE    DB       00                ;Fin del indicador de proceso
HANDLE    DW
IOAREA    DB       32 DUP(' ')
OPENMSG   DB       '*** Open error ***', ODH, OAH
PATHNAM   DB       'D:\NAMEFILE.SRT',0
READMSG   DB       '*** Read error ***', ODH, OAH
ROW       DB       00
-------------------------------------------------------------------
          .CODE
BEGIN     PROC     FAR
          MOV      AX,@data          ;Inicializa
          MOV      DS,AX             ;  registros de
          MOV      ES,AX             ;  segmento
          MOV      AX,0600H
          CALL     Q10SCR            ;Limpia la pantalla
          CALL     Q20CURS           ,•Coloca el cursor
          CALL     E10OPEN           ,-Abre archivo, designa DTA
          CMP      ENDCDE,00         /¿Apertura válida?
          JNZ      A90               ;  no, salir
A20LOOP:
          CALL     F10READ           Lee registro en disco
          CMP      ENDCDE,00         /¿Lectura normal?
          JNZ      A90               /  no, salir
          CALL     G10DISP           /  sí, desplegar nombre,
          JMP      A20LOOP           ;  continuar
A90:                                 /Fin del procesamiento,
          MOV      AX,4C00H          /  salir al DOS
          INT      21H
BEGIN     ENDP
;                  Abre archivo
          ---------------------------
ÉlOOPEN   PROC     NEAR
          MOV      AH,3DH            /Petición para abrir
          MOV      AL,00             /Archivo normal
          LEA      DX,PATHNAM
          INT      21H
          JC       E20               /¿Error?
          MOV      HANDLE,AX         /  no, guardar manejador
          RET
E20:
          MOV      ENDCDE,01         / sí,
          LEA      DX,OPENMSG        /  desplegar
          CALL     X10ERR            /  mensaje de error
          RET
E10OPEN   ENDP
í                  Lee registro de disco:
          ---------------------------------------------------------
FlOREAD   PROC    NEAR
          MOV      AH,3FH            /Petición de lectura
          MOV      BX,HANDLE
          MOV      CX,32             ,•30 para el nombre + 2 para C
          LEA      DX,IOAREA
          INT      21H
          JC       F20               /¿Error en la lectura?
          CMP      AX,00             /¿Fin del archivo?
          JE       F30
          CMP      IOAREA,1AH        /¿Marcador EOF?
          JE       F30               /  sí, salir
          JMP      F90
```

Figura 17-3 Uso de un manejador para leer un archivo

```
F20:                                        no,
        LEA     DX,READMSG                    lectura no válida
        CALL    X10ERR
F30:
        MOV     ENDCDE,01                   Fuerza la terminación
F90:    RET
F10READ ENDP

                Despliega nombre:

G10DISP PROC    NEAR
        MOV     AH,40H                  |Petición para desplegar
        MOV     BX, 01                  |Establece el manejador
        MOV     CX, 32                  |  y la longitud
        LEA     DX,IOAREA
        INT     21H
        CMP     ROW,20                  |¿Inferior de la pantalla?
        JAE     G80                     |  sí, pasar
        INC     ROW                     |  no, incrementar renglón
        JMP     G90                     |
G80:
        MOV     AX,0601H
        CALL    Q10SCR                  ,-Recorrer
        CALL    Q2 0CURS                ;Colocar cursor
G90:    RET
G10DISP ENDP
,               Recorrido de la pantalla:

Q10SCR  PROC    NEAR                     AX se designó antes
        MOV     BH,1EH                  ;Designa color
        MOV     CX,0000
        MOV     DX,184FH                -Petición para recorrer
        INT     10H
        RET
Q10SCR  ENDP
                Coloca el cursor:

Q2 0CURS PROC   NEAR
        MOV     AH,02H                  |Petición para colocar
        MOV     BH, 00                  |  el cursor
        MOV     DH,ROW                  |  renglón
        MOV     DL, 00                  |  columna
        INT     10H
        RET
Q2 0CURS ENDP
                Despliega mensaje de error en disco:

X10ERR  PROC    NEAR
        MOV     AH,40H                  |DX contiene la dirección
        MOV     BX, 01                  |Manej ador
        MOV     CX,20                   |Longitud
        INT     21H                     |  del mensaje
        RET
X10ERR  ENDP
        END     BEGIN
```

Figura 17-3 (continuación)

- G10DISP Despliega los registros y recorre la pantalla. Como los caracteres Enter y Avance de línea ya siguen a cada registro, el programa no tiene que avanzar el cursor cuando despliega los registros.

## PROCESAMIENTO DE ARCHIVOS ASCII

Los ejemplos anteriores crearon archivos y los leyeron, pero también puede necesitar procesar archivos ASCII creados por DOS o un editor. Todo lo que necesita saber es la organización del directorio y de la FAT y la forma en que el sistema almacena datos en un sector. Por ejemplo, el DOS almacena su información en un archivo .ASM, exactamente en la forma en que usted lo teclea, incluyendo los caracteres par el tabulador (09H), Enter (ODH) y el Avance de línea (OAH). Para conservar espacio en disco, el DOS no almacena los espacios que aparecen en la pantalla inmediatamente antes del carácter de tabulación o los espacios en una línea a la derecha de un carácter Enter. Lo siguiente ilustra una instrucción de lenguaje ensamblador como se ingresaría en un teclado:

```
<Tab>MOV<Tab>AH,09<Enter>
```

La representación hexadecimal para estos datos ASCII sería

```
094D4F560941482C3 03 90D0A
```

en donde 09H es el Tab, ODH Enter y OAH es el Avance de línea. Cuando el comando TYPE o un editor lee el archivo, los caracteres Tab, Enter y Avance de línea de manera automática ajustan el cursor en la pantalla.

Examinemos ahora el programa de la figura 17-4, el cual lee y despliega el archivo P17HANRD.ASM (de la figura 17-3), un sector a la vez. El programa realiza muchas de las funciones que el TYPE del DOS, en donde cada línea despliega todo hasta los caracteres Enter/ Avance de línea. Como las líneas en un archivo ASCII son de longitud variable, tiene que rastrear hasta el final cada línea antes de desplegarla. El recorrido de la pantalla puede ser un problema. Si usted no realiza pruebas especiales para determinar si ya se ha alcanzado la parte inferior de la pantalla, la operación despliega de manera automática nuevas líneas sobre las anteriores, y si una de ellas es de mayor longitud, los caracteres anteriores aparecerán a la derecha. Para un recorrido apropiado de la pantalla, tiene que contar renglones y probar si está en la parte inferior de la pantalla.

El programa lee un sector completo de datos y lo envía a SECTOR. El procedimiento G10XFER transfiere un byte a la vez desde SECTOR a DISAREA, en donde los caracteres son desplegados. Cuando se encuentra un Avance de línea, la rutina despliega el contenido de DISAREA hasta e incluyendo el Avance de línea. (La pantalla de despliegue acepta caracteres de tabulación 09H y coloca de manera automática el cursor en la siguiente posición divisible entre ocho.)

El programa tiene que verificar el final del sector (para leer otro sector) y el final del área de despliegue. Para archivos ASCII convencionales, como archivos .ASM, cada línea es relativamente corta y es seguro que terminan con Enter/Avance de línea. Archivos que no son ASCII, como los archivos .EXE y .OBJ, no tienen líneas, de modo que el programa tiene que verificar por el fin de DISAREA para evitar la caída del sistema. El programa está proyectado para desplegar sólo archivos ASCII, pero la prueba del final debe asegurarse contra archivos no esperados.

Éstos son los pasos en G10XFER:

1. Inicializa la dirección de SECTOR y la dirección de DISAREA.
2. Si se llegó al final de SECTOR, lee el sector siguiente. Si se llegó al final del archivo, salir; de otra forma inicializa la dirección de SECTOR.

```
TITLE     P17ASCRD (EXE)   Lee un archivo ASCII
          .MODEL   SMALL
          .STACK   64

          .DATA
DISAREA   DB       120 DUP('•')      Área de despliegue
ENDCDE    DW       00                Indicador del fin del proceso
HANDLE    DW       0                 Manejador de archivo
OPENMSG   DB       '*** Open error ***'
PATHNAM   DB       'D:\17HANRED.ASM', 0
ROW       DB       00
SECTOR    DB       512 DUP(' ')      ;Área de entrada

          .CODE
BEGIN     PROC     FAR               Procedimiento principal
          MOV      AX,@data          Inicializa
          MOV      DS,AX               registros de
          MOV      ES,AX             segmentos
          MOV      AX,0600H
          CALL     Q10SCR            Limpia la pantalla
          CALL     Q20CURS           Coloca el cursor
          CALL     E10OPEN           Abre archivo
          CMP      ENDCDE,00         ¿Apertura válida?
          JNE      A90                 no, salir
A20LOOP:                             sí, continuar
          CALL     RlOREAD           Lee primer sector del disco
          CMP      ENDCDE,00         ¿Fin del archivo, no hay datos?
          JE       A90                 sí, salir
          CALL     G10XFER           Despliega y lee
A90:
          MOV      AH,3EH            Petición para cerrar archivo
          MOV      BX,HANDLE
          INT      21H
          MOV      AX,4C00H          •Sale al DOS
          INT      21H
BEGIN     ENDP

          Abre archivo en disco:

E10OPEN   PROC     NEAR
          MOV      AH,3DH            ¡Petición para abrir
          MOV      AL,00             ,-Sólo lectura
          LEA      DX, PATHNAM
          INT      21H
          JNC      E20               ;Examina bandera de acarreo,
          CALL     XI0ERR            ;  si está en uno, error
          RET
E20:
          MOV      HANDLE, AX        ;Guardar manejador
          RET
E10OPEN   ENDP

          Transfiere datos a la línea de despliegue:

G10XFER   PROC     NEAR
          CLD                        ,-De izquierda a derecha
          LEA      SI,SECTOR
G20:
          LEA      DI,DISAREA
G30:
          LEA      DX,SECTOR+512
          CMP      SI,DX             ;¿Fin del sector?
          JNE      G40               ;  no, pasar
          CALL     RlOREAD           ,•  sí; lee el siguiente
          CMP      ENDCDE,00         ;¿Fin del archivo?
```

Figura 17-**4** Lectura de un archivo ASCII

```
            JE      G80             ;  sí, salir
            LEA     SI,SECTOR
G40:
            LEA     DX,DISAREA+80
            CMP     DI.DX           ;¿Fin de DISAREA?
            JB      G50             ;  no, pasar
            MOV     [DI],ODOAH        sí, establecer CR/LF
            CALL    HIODISP         ;  desplegar
            LEA     DI,DISAREA
G50:
            LODSB                   ; [SI] a AL; INC SI
            STOSB                   ;AL a [DI], INC DI
            CMP     AL,1AH          ,• ¿Fin del archivo?
            JE      G80             ,• sí, salir
            CMP     AL,OAH          ;¿Avanza línea?
            JNE     G30             ;  no, repetir el ciclo
            CALL    HIODISP         ;  sí, desplegar
            JMP     G20
G80:
            CALL    HIODISP         ;Despliega última línea
G90:        RET
G10XFER     ENDP
            Despliega línea

HIODISP     PROC    NEAR
            MOV     AH,40H          ;Petición de despliegue
            MOV     BX,01           ;Manejador
            LEA     CX,DISAREA      ;Calcula
            NEG     CX              ;  la longitud
            ADD     CX.DI           ;  de la línea
            LEA     DX,DISAREA
            INT     21H
            CMP     ROW,22          /¿Inferior de la pantal
            JAE     H20             ;  no, salir
            INC     ROW
            JMP     H90
H20:
            MOV     AX,0601H            ;Recorre
            CALL    Q10SCR
            CALL    Q20CURS
H90:        RET
HIODISP     ENDP
            Recorre la pantalla:

Q10SCR      PROC    NEAR            ;AX se designa antes
            MOV     BH,1EH          ,-Designa atributo de co
            MOV     CX,0000         ;Recorre
            MOV     DX,184FH
            INT     10H
            RET
Q10SCR      ENDP
            Coloca el cursor:

Q20CURS     PROC    NEAR
            MOV     AH,02H          ;Petición para
            MOV     BH,00           ;  colocar el cursor
            MOV     DH,ROW
            MOV     DL,00
            INT     10H
            RET
Q20CURS     ENDP
            Lee sector del disco:
```

Figura 17-4 (continuación)

```
RlOREAD   PROC    NEAR
          MOV     AH,3FH              Petición para leer
          MOV     BX,HANDLE           Dispositivo
          MOV     CX,512              Longitud
          LEA     DX,SECTOR           Búfer
          INT     21H
          MOV     ENDCDE,AX
          RET
RlOREAD   ENDP
                  Despliega mensaje de error de disco:

X10ERR    PROC   NEAR
          MOV     AH,40H              Petición para desplegar
          MOV     BX, 01              Manej ador
          MOV     CX, 18              Longitud
          LEA     DX,OPENMSG
          INT     21H
          MOV     ENDCDE,01          /Indicador de error
          RET
X10ERR    ENDP
          END     BEGIN
```

Figura 17-4 (continuación)

3. Si se llegó al final de DISAREA, fuerza un Enter/Avance de línea, despliega la línea e inicializa DISAREA.
4. Obtiene un carácter de SECTOR y lo almacena en DISAREA.
5. Si el carácter es fin de archivo (1AH), salir.
6. Si el carácter es un Avance de línea (OAH), despliega la línea y va al paso 2; de otra forma va al paso 3.

Intente correr este programa con DEBUG con un número apropiado de unidad y archivo ASCII. Después de cada entrada de disco, despliegue el contenido del área de entrada y vea cómo el DOS ha formateado sus registros. Una mejora a este programa sería solicitar al usuario introducir el nombre del archivo y la extensión desde el teclado.

## USO DE MANEJADORES DE ARCHIVO PARA PROCESAMIENTO DIRECTO

El estudio anterior acerca de archivos secuenciales en disco es adecuado para crear un archivo, para imprimir su contenido y para realizar cambios a archivos pequeños. Sin embargo, algunas aplicaciones implican el acceso a un registro en particular de un archivo, como información sobre algunos empleados o partes de inventario.

Para actualizar un archivo con información nueva, un programa que está restringido a procesamiento secuencial tiene que leer cada registro en el archivo hasta llegar a aquel que se requiere. Por ejemplo, para accesar el registro 300 de un archivo, el procesamiento secuencial implica la lectura de los 299 registros precedentes antes de enviar el registro 300 (aunque el sistema en un inicio pueda estar en un número de registro específico).

La solución general es usar *procesamiento directo,* en el que un programa puede accesar de manera directa cualquier registro dado en un archivo. Aunque un archivo es *creado* de manera secuencial, puede *accesar* registros de forma secuencial o directa.

Cuando un programa solicita primero un registro aleatorio, la operación utiliza el directorio para localizar el sector en el que el registro se encuentra, lee todo el sector del disco y lo manda al búfer y envía el registro requerido al programa.

En el ejemplo siguiente, los registros son de 128 bytes y cuatro por sector. Una petición para el registro número 21 provoca que los siguientes cuatro registros sean leídos y enviados al búfer:

| record #20 | record #21 | record #22 | record #23 |
|---|---|---|---|

Cuando el programa solicita el siguiente registro aleatorio, digamos el número 23, la operación examina primero el búfer. Como el registro ya se encuentra ahí, lo transfiere de manera directa al programa. Si el programa solicita un número de registro que no se encuentra en el búfer, la operación utiliza el directorio para localizar el registro, lee todo el sector y lo envía al búfer y manda el registro al programa. De acuerdo con esto, por lo común es más eficiente solicitar números de registros que estén cercanos en el archivo.

### INT 21H, función 42H: Mueve apuntador de archivo

El DOS mantiene un apuntador que la operación de abrir inicializa en cero y las lecturas y escrituras subsecuentes incrementan por cada registro procesado. Usted puede usar la función del DOS, 42H (Mover el apuntador del archivo), para colocar el apuntador del archivo en cualquier posición dentro de un archivo y después usar otros servicios para recuperación o actualización directas.

Coloque el manejador de archivo en el BX y el desplazamiento necesario como bytes en el CX:DX. Para un movimiento hasta de 65,535 bytes, establezca cero en el CX y el valor del desplazamiento en el DX. También establezca un código de método en el AL que indique a la operación el punto desde el cual se tomará el desplazamiento:

- 00 Toma el desplazamiento desde el inicio del archivo.
- 01 Toma el desplazamiento desde la posición actual del apuntador del archivo, que puede ser cualquiera dentro del archivo, incluso al inicio.
- 02 Toma el desplazamiento desde el final del archivo. Puede usar este código de método para agregar registros al final del archivo. O puede determinar el tamaño del archivo estableciendo el CX:DX a cero y usando el código de método 02.

El ejemplo siguiente mueve el apuntador 1,024 bytes desde el inicio de un archivo:

```
MOV   AH,42H          /Petición para mover el apuntador
MOV   AL,00           ; al inicio del archivo
LEA   BX,HANDLEl      /Designa el manejador de archivo
MOV   CX,00
MOV   DX,1024         /Desplazamiento de 1,024 bytes
INT   21H                  /Llama al DOS
JC    error
```

Una operación válida pone en cero la bandera de acarreo y envía la nueva posición del apuntador en el DX: AX. Entonces se puede realizar una operación de lectura o escritura para procesamiento directo. Una operación no válida pone en uno la bandera de acarreo y regresa en el AX el código 01 (código no válido de método) o 06 (manejador no válido).

Programa: Lectura directa de un archivo en disco

El programa de la figura 17-5 lee el archivo creado en la figura 17-2. Al teclear un número relativo de registro que esté dentro de los límites del archivo, el usuario puede solicitar cualquier registro en el archivo para que sea desplegado en la pantalla. Si el archivo contiene 24 registros, entonces los números válidos de registro son desde 01 hasta 24. Un número ingresado desde el teclado está en formato ASCII y en este caso sólo debe ser de uno o dos dígitos.

El programa está organizado como sigue:

C10OPEN Abre el archivo y obtiene el manejador de archivo.

D10RECN Acepta un número de registro desde el teclado y verifica su longitud en la lista de parámetros. Existen tres posibles longitudes:

- 00 Fin de la petición de procesamiento
- 01 Petición de un dígito, almacenado en el AL
- 02 Petición de dos dígitos, almacenado en el AL

El procedimiento ha convertido el número ASCII a binario. Ya que el número está en el AL, la instrucción AAD funciona bien para este propósito. El sistema reconoce la posición 0 como el inicio de un archivo. El programa resta uno del número actual (así que, por ejemplo, una solicitud del usuario para el registro 1 se convierte en el registro 0), multiplica el número por 16 (la longitud de los registros en el archivo) y almacena el resultado en un campo llamado RECINDX.

Como ejemplo, si el número ingresado es 12 ASCII, el AX contendría 3132. Una instrucción AND convierte este valor a 0102, ADD después lo convierte a 000C (12) y SHL multiplica de manera eficaz el número por 16 para obtener C0 (192). Una mejora sería validar el número a la entrada.

F10READ Usa la función 42H y la posición relativa del registro desde RECINDX para colocar el apuntador del archivo y emitir la función 3FH para enviar el registro solicitado al programa en IOAREA.

G10DISP Despliega el registro recuperado.

## SERVICIOS DE DISCO QUE USAN BLOQUES DE CONTROL DE ARCHIVO

Ahora estudiamos los servicios FCB del DOS para la creación y procesamiento de archivo tanto secuenciales como de acceso directo. Todos estos servicios fueron introducidos por la primera versión del DOS y están disponibles en todas las versiones.

El procesamiento en disco para los servicios FCB del DOS implican la definición de un bloque de control de archivo (FCB) que define el archivo y un área de transferencia a disco (DTA) que define registros. Usted proporciona al DOS la dirección del DTA para todas las operaciones de entrada/salida de disco. Observe que los FCB no utilizan manejadores de archivo y no usan los códigos de error listados en la figura 17-1; tampoco ponen en cero o uno a la bandera de acarreo para indicar éxito o fracaso. (También los FCB existen en el PSP, que el DOS instala inmediatamente antes de los programas cargados en memoria para su ejecución.)

```
TITLE     P17RANRD (EXE)   Lectura directa de registros en disco
          .MODEL   SMALL
          .STACK   64
-------------------------------------------------------------------------
          .DATA
HANDLE    DW       ?                  | Manejador de archivo
RECINDX   DW       7                  | índice del registro
ERRCDE    DB       00                 | Indicador de error de lectura
PROMPT    DB       'Record number? $'
IOAREA    DB       32 DUP(' ')        | Área de registro de disco
PATHNAM   DB       'D:\NAMEFILE.SRT',0
OPENMSG   DB       '*** Open error ***', ODH, OAH
READMSG   DB       '*** Read error ***', ODH, OAH
ROW       DB       00
COL       DB       00

RECDPAR   LABEL    BYTE               | Lista de parámetros de entrada
MAXLEN    DB       3                  |   longitud máxima
ACTLEN    DB       7                  |   longitud actual
RECDNO    DB       3 DUP(' ')         |   número -de registro
-------------------------------------------------------------------------
          .CODE
.386
BEGIN     PROC     FAR
          MOV      AX.odata           | Inicializa
          MOV      DS, AX             |   registros de
          MOV      ES, AX             |   segmentos
          MOV      AX,0600H
          CALL     Q10SCRN            | Limpia la pantalla
          CALL     Q20CURS            | Coloca el cursor
          CALL     C10OPEN            | Abre archivo
          CMP      ERRCDE,00          | ¿Apertura válida?
          JNZ      A90                |   no, salir
A20LOOP:
          CALL     D10RECN            | Petición de #reg
          CMP      ACTLEN,00          | ¿Existen más peticiones?
          JE       A90                |   no, salir
          CALL     F10READ            | Lee registro en disco
          CMP      ERRCDE,00          | ¿Lectura normal?
          JNZ      A30                |   no, pasar
          CALL     G10DISP            |   sí, desplegar nombre,
A30:
          JMP      A20LOOP            |  continuar
A90:
          MOV      AX,4CO0H           | Salir al DOS
          INT      21H
BEGIN     ENDP
          Abre archivo:
-------------------------------------------------------------------------
C10OPEN   PROC     NEAR
          MOV      AH,3DH             | Petición para abrir
          MOV      AL, 00             |  Archivo normal
          LEA      DX, PATHNAM
          INT      21H
          JC       C20                |  ¿Error?
          MOV      HANDLE, AX         |   no, guardar manejador
          RET
C20:
          MOV      ERRCDE,01          |   sí,
          LEA      DX,OPENMSG         |   desplegar
          CALL     X10ERR             |   mensaje de error
          RET
```

Figura 17-5 Lectura directa de un archivo en disco

```
C10OPEN   ENDP
                Obtiene número de registro:

D10RECN   PROC  NEAR
          MOV   AH,09H            ;Petición para desplegar indicación
          LEA   DX,PROMPT
          INT   21H

          MOV   AH,OAH            /Petición para ingresar
          LEA   DX, RECDPAR       ;  número de registro
          INT   21H
          CMP   ACTLEN,01         ,-Verifica longitud 0, 1, 2
          JB    D40               /Longitud 0, termina
          JA    D20
          XOR   AH, AH            /Longitud 1
          MOV   AL,RECDNO
          JMP   D30
D20:
          MOV   AH, RECDNO        /Longitud 2
          MOV   AL.RECDNO+l
D30:
          AND   AX,0F0FH           Limpia los 3 ASCII
          AAD                      Convierte a binario
          DEC   AX                 Ajusta (primer registro es 0)
          SHL   AX, 05             Multiplica por 1S
          MOV   RECINDX,AX         Guarda el índice
D40:
          MOV   COL,20
          CALL  Q2 0CURS
          RET
          ENDP
                Lectura directa de registro en disco:

FlOREAD   PROC  NEAR
          MOV   AX,4200H           Petición para colocar el apuntador de archivo
          MOV   AL, 00             Inicio del archivo
          MOV   BX,HANDLE
          MOV   CX, 00
          MOV   DX, RECINDX
          INT   21H
          JC    F20                'Condición de error?
                                     si, pasar
          MOV   AH,3FH             Petición de lectura
          MOV   BX,HANDLE
          MOV   CX, 32             ;30 para el nombre, 2 para CR/LF
          LEA   DX,IOAREA
          INT   21H
          JC    F20                /¿Error en la lectura?
          CMP   IOAREA, 1AH        /¿Marcador EOF?
          JE    F30                  sí, salir
          JMP   F90
F20:                                  no,
          LEA   DX,READMSG            lectura no válida
          CALL  X10ERR
F30:
          MOV   ERRCDE,01          /Fuerza la terminación
F90:      RET
Fl OREAD  ENDP
                Despliega nombre:

G10DISP   PROC  NEAR
          MOV   AH,40H             /Petición para desplegar
          MOV   BX, 01             /Designa manejador
          MOV   CX, 32             /  y longitud
```

Figura 17-5 (continuación)

```
            LEA     DX,IOAREA
            INT     21H
            MOV     COL,00              Limpia la columna
            CMP     ROW,20              ¿Inferior de la pantalla?
            JAE     G80                   sí, pasar
            INC     ROW                   no, incrementar renglón
            JMP     G90
380 :
            MOV     AX,0601H
            CALL    Q10SCRN            ;Recorre
            CALL    Q20CURS            ;Coloca el cursor
390 :       RET
310DISP     ENDP
                    Recorre pantalla:

Q10SCRN     PROC    NEAR               ;AX se designa antes
            MOV     BH,1EH             ,-Designa color
            MOV     CX,0000
            MOV     DX,184FH           ;Petición para recorrer
            INT     10H
            RET
Q10SCRN     ENDP
                    Coloca el cursor:

Q20CURS     PROC    NEAR
            MOV     AH,02               Petición para colocar
            MOV     BH, 00                el cursor
            MOV     DH,ROW                renglón
            MOV     DL,COL                columna
            INT     10H
            RET
Q20CURS     ENDP
                    Despliega mensaje de error de disco:

X10ERR      PROC    NEAR
            MOV     AH.40H              DX contiene la dirección
            MOV     BX, 01              Manej ador
            MOV     CX, 20              Longitud
            INT     21H                   del mensaje
            INC     ROW
            RET
X10ERR      ENDP
            END     BEGIN
```

Figura 17-5 (continuación)

**Bloque de control de archivo**

Como el método de FCB no permite el uso de nombres de ruta, lo emplean principalmente para procesamiento de archivos que se encuentran en el directorio actual. El FCB que usted define en el área de datos contiene la información siguiente acerca del archivo y de sus registros (inicializa los bytes 00-15 y 32-36, mientras que el DOS establece los bytes 17-31):

0 *Unidad de disco.* Para la mayoría de las operaciones de FCB, 00 es la unidad por omisión, 01 es la unidad A, 02 es la unidad B y así sucesivamente.

1-8 *Nombre de archivo.* El nombre del archivo, justificado a la izquierda con blancos al final, si existen.

9-11 *Extensión del nombre de archivo.* Una subdivisión del nombre del archivo para identificación posterior, como .DOC o .ASM, justificado a la izquierda si tiene menos de tres caracteres. Cuando crea un archivo, el DOS almacena su nombre de archivo y su extensión en el directorio.

12-13 *Número actual de bloque.* Un bloque consta de 128 registros. Las operaciones de lectura y escritura utilizan el número actual de bloque y el número actual de registro (byte 32) para localizar un registro particular. El número es relativo al inicio del archivo, en donde el primer bloque es 0, el segundo es 1, etc. Una operación de abrir establece esta entrada en cero. El DOS maneja el número actual de bloque de manera automática, aunque usted puede cambiarlo para procesamiento directo.

14-15 *Tamaño lógico del registro.* Una operación de abrir inicializa el tamaño del registro en 128 (80H). Después de abrir y antes de cualquier lectura o escritura, puede cambiar esta entrada al tamaño que usted necesite.

16-19 *Tamaño del archivo.* Cuando un programa crea un archivo, el DOS calcula y almacena su tamaño (número de registros x tamaño del registro) en el directorio. Una operación de abrir de manera subsecuente extrae el tamaño del directorio y lo almacena en este campo. Su programa puede leer el campo, pero no cambiarlo.

20-21 *Fecha.* En el directorio el DOS registra la fecha en que el archivo fue creado o actualizado por última vez. Una operación de abrir extrae la fecha del directorio y la almacena en este campo.

22-31 *Reservado para el DOS.*

32 *Número actual de registro.* Esta entrada es el número de registro actual (0-127) dentro del bloque actual (véase los bytes 12-13). El sistema utiliza el bloque y los registros actuales para localizar registros en el archivo. Aunque al abrir inicializa el número de registro a cero, puede fijar este campo para que empiece un procesamiento secuencial en cualquier número entre 0 y 127.

33-36 *Número relativo de registro.* Para lectura/escritura directa, esta entrada debe contener un número relativo de registro. Por ejemplo, para leer de manera directa el registro 25 (19H), establezca la entrada en 19000000H. Para procesamiento directo, el sistema convierte de manera automática el número relativo de registro al bloque y registro actuales. A causa del límite sobre el tamaño máximo de un archivo (1,073,741,824 bytes) un archivo con un tamaño de registro más pequeño puede contener más registros y puede tener un número relativo de registro máximo mayor que un archivo con un tamaño de registro más grande. Si el tamaño del registro es mayor que 64, el byte 36 siempre contiene 00.

Precediendo al FCB está una extensión opcional de siete bytes, que puede ser usada para procesamiento de archivos con atributos especiales. Para usar la extensión, codifique el primer byte con FFH, el segundo byte con el atributo del archivo (descrito en el capítulo 16) y el resto de los cinco bytes con ceros hexadecimales.

## USO DE FCB PARA CREAR ARCHIVOS EN DISCO

Para cada archivo en disco que es referenciado, un programa que utiliza los servicios de disco originales de DOS define un FCB. Las operaciones de disco requieren la dirección del FCB en el registro DX para accesar los campos dentro del FCB. Las operaciones incluyen la creación de un archivo, designar el área de transferencia de disco (DTA), escribir un registro y cerrar un archivo.

### INT 21H, función 16H: Crea archivo

En la inicialización, un programa utiliza la función 16, de la INT 21H, para crear un archivo nuevo:

```
MOV   AH.16H          ;Petición para crear
LEA   DX,nombredeFCB  ; un archivo en disco
INT   21H             ,•Llama al DOS
```

El DOS busca en el directorio un nombre de archivo que coincida con la entrada del FCB. Si encuentra uno, el DOS reutiliza el espacio en el directorio; si no se encuentra, busca una entrada vacía. Después la operación inicializa el tamaño del archivo en cero y abre el archivo. Al abrir verifica un espacio disponible en disco y establece uno de los siguientes códigos de regreso en el AL: 00H = espacio disponible; FFH = no hay espacio disponible. Al abrir también inicializa el número actual de bloque FCB a cero y establece un valor por omisión en el tamaño del registro del FCB a 128 bytes (80H). Antes de escribir un registro, puede hacer caso omiso de este valor por omisión y usar su propio tamaño de registro.

### El área de transferencia de disco

El área de transferencia de disco (DTA) es el inicio de la definición de su registro de salida. Ya que el FCB contiene el tamaño del registro, el DTA no necesita un delimitador para indicar el final del registro. Antes de una operación de escritura, utilice la función 1AH de FCB para dar al DOS la dirección del DTA. Sólo un DTA puede estar activo en cualquier momento. El código siguiente inicializa la dirección del DTA:

```
MOV  AH, 1AH              ,-Petición para establecer la dirección
LEA  DX, nombredeDTA      ; del DTA
INT  21H                  ;Llama al DOS
```

Si un programa sólo procesa un archivo de disco, necesita inicializar el DTA sólo una vez para toda su ejecución. Si un programa procesa más de un archivo, debe inicializar el DTA apropiado inmediatamente antes de cada lectura o escritura.

### INT 21H, función 15H: Escribe registro

Para escribir un registro de disco de forma secuencial, utilice la función 15H para FCB:

```
MOV  AH,15H             /Petición para escribir un registro
LEA  DX,nombre de FCB   / de forma secuencial
INT  21H                /Llama al DOS
```

La operación de escritura utiliza la información en el FCB y la dirección del DTA actual. Si el registro es del tamaño de un sector, la operación escribe el registro. De otra forma, la operación envía los registros a un área de búfer que es la longitud de un sector y escribe el búfer cuando está lleno. Por ejemplo, si cada registro es de 128 bytes de longitud, la operación llena el búfer con cuatro registros (4 x 128 = 512) y después escribe el búfer en un sector completo de disco.

En una escritura exitosa, el DOS incrementa el campo del tamaño tamaño del FCB (añadiendo el tamaño del registro) e incrementa el número actual de registro en uno. Cuando el número actual de registro excede 127, la operación lo pone en cero e incrementa el número actual de bloque del FCB. (También se puede cambiar el bloque actual y el número actual de registro.) La operación de escritura establece uno de los siguientes códigos de regreso en el AL: OOH = la escritura fue exitosa; 01H = disco lleno; 02H = DTA es muy pequeña para el registro.

### INT 21H, función 10H: Cierra archivo

Cuando ha terminado de escribir registros en el archivo, puede escribir un marcador de fin de archivo (1AH en el primer byte de un último registro especial; no lo confunda con la función 1AH) y después utilice la función 10H para FCB a fin de cerrar el archivo:

```
MOV  AH,10H           ;Petición para cerrar
LEA  DX,nombredeFCB   ; el archivo
INT  21H              ,-Llama al DOS
```

La operación para cerrar escribe en disco cualquier información parcial que aún se encuentre en el búfer del disco del DOS y actualiza el directorio con la fecha y el tamaño del archivo. Uno de los códigos siguientes es regresado al AL: OOH = se cerró de manera exitosa; FFH = el archivo no estaba en la posición correcta en el directorio, tal vez provocado porque el usuario cambió por un disco flexible.

## USO DE FCB PARA LECTURA SECUENCIAL DE ARCHIVOS EN DISCO

Un programa que lee un archivo en disco define un FCB exactamente igual al usado para crear el archivo. Las operaciones de lectura secuencial incluyen abrir el archivo, definir el DTA, leer registros y cerrar el archivo.

### INT 21H, función OFH: Abre archivo

La función OFH abre un archivo FCB para entrada:

```
MOV  AH,OFH           /Petición para abrir
LEA  DX,nombredeFCB   / el archivo
INT  21H              /Llama al DOS
```

La operación para abrir verifica que el directorio tenga una entrada con el nombre y extensión del archivo definidos en el FCB. Si la entrada no está en el directorio, la operación regresa el código FFH en el AL. Si está presente la entrada, la operación regresa el código 00 en el AL y establece el tamaño actual del archivo, la fecha, el número actual de bloque (0) y el tamaño del registro (80H) en el FCB. Después de que se ejecuta la operación de abrir, se puede hacer caso omiso del tamaño por omisión del registro.

### El área de transferencia de disco

El DTA define un área para el registro de entrada, de acuerdo con el formato usado para crear el archivo. Utilice la función 1AH para FCB a fin de fijar la dirección del DTA, igual que cuando crea un archivo en disco.

### INT 21H, función 14H: Lee registro

Para leer secuencialmente un registro en disco, utilice la función 14H para FCB:

```
MOV  AH,14H              ,-Petición para  leer
LEA  DX,nombredeFCB      /  secuencialmente un registro
INT  21H                 /Llama al DOS
```

La operación establece uno de los siguientes códigos de regreso en el AL: 00 = lectura exitosa; 01 = fin de archivo, ningún dato fue leído; 02 = DTA es muy pequeña para el registro; 03 = fin del archivo, el registro se leyó parcialmente y se rellenó con ceros.

Para una lectura exitosa, la operación utiliza la información en el FCB para enviar el registro de disco, iniciando en la dirección del DTA. Un intento de leer después del último registro del archivo provoca que la operación señale una condición de fin de archivo que establece el AL con 01H, que usted debe examinar. Es una práctica recomendada cerrar un archivo de entrada después que se ha leído completamente, ya que el DOS limita el número de archivos que pueden estar abiertos al mismo tiempo.

## USO DE FCB PARA PROCESAMIENTO DIRECTO

Los requerimientos para procesamiento directo simplemente implican insertar el número de registro necesario en el campo de registro relativo del FCB (bytes 33-36) y emitir un comando de lectura o escritura directas. Para localizar un registro de forma directa, el sistema convierte automáticamente el número relativo de registro al bloque actual (bytes 12-13) y al registro actual (byte 32).

### INT 21H, función 21H: Lectura directa de un registro

La operación para abrir y para establecer el DTA son las mismas para procesamiento secuencial y para directo. Considere un programa que sea para leer de manera directa el número relativo de registro 05. Inserte el número 05 en el campo FCB para el número relativo de registro y solicite la función 21H:

```
MOV  AH,21H              /Petición
LEA  DX,nombredeFCB      /  de lectura directa
INT  21H                 /Llama al DOS
```

La operación de lectura regresa uno de los siguientes códigos en el AL: 00 = lectura exitosa; 01 = fin de archivo, no hay más datos disponibles; 02 = DTA muy pequeña para el registro; 03 = el registro ha sido leído de manera parcial y rellenado con ceros.

Una operación exitosa convierte el número relativo de registro a bloque y registro actuales. Utiliza este número para localizar el registro de disco que se necesita y lo envía al DTA. Respuestas erróneas pueden ser causadas por un número relativo de registro no válido o una dirección incorrecta en el DTA o FCB.

### INT 21H, función 22H: Escritura directa de un registro

La operación de crear y establecer el DTA son las mismas para procesamiento directo y para procesamiento secuencial. Con el número relativo de registro inicializado en el FCB, la escritura directa utiliza la función 22H:

```
MOV  AH,22H            /Petición escritura
LEA  DX,nombredeFCB    / directa
INT  21H               /Llama al DOS
```

La operación de escritura regresa uno de los códigos siguientes en el AL: 00 = escritura exitosa; 01 = disco lleno; 02 = DTA muy pequeño para el registro.

## PROCESAMIENTO DIRECTO DE BLOQUES

Si un programa tiene espacio suficiente, una operación directa de bloque puede escribir un archivo completo del DTA al disco y puede leer el archivo completo desde el disco al DTA. Pero aun primero tiene que abrir el archivo e inicializar el DTA. Después puede empezar el procesamiento con cualquier número relativo válido de registro y cualquier número de registros, aunque el bloque debe estar dentro del rango de los registros del archivo.

### INT 21H, función 28H: Escritura directa de bloque

Para la escritura directa de bloque, inicialice el número de registros necesarios en el registro CX, **fije** el número relativo de registro inicial en el FCB y utilice la función 28H:

```
MOV  AH,28H            /Petición de escritura directa de bloque
MOV  CX,registros      /Fija el número de registros
LEA  DX,nombredeFCB    /Dirección del FCB
INT  21H               /Llama al DOS
```

La operación convierte el número relativo de registro en el FCB al bloque y registro actuales. Utiliza este número para determinar la posición de inicio en el disco y establece uno de los siguientes códigos de regreso en el AL: 00 = escritura exitosa de todos los registros; 01 = no se escribió ningún registro a causa de espacio insuficiente en el disco; 02 = DTA muy pequeño para el registro. La operación establece el campo de registro relativo en el FCB y los campos de bloque y registro actuales al número de registro siguiente.

### INT 21H, función 27H: Lectura directa de bloque

Para una lectura directa de bloque, inicialice el número de registros necesarios en el CX, y utilice la función 27H para FCB:

```
MOV  AH,27H            /Petición para lectura directa de bloque
MOV  CX,registros      ,-Inicializa número de registros
LEA  DX,nombredeFCB    /Dirección del FCB
INT  21H               /Llama al DOS
```

La operación de lectura regresa uno de los códigos siguientes en el AL: 00 = lectura exitosa de todos los registros; 01 = ha leído un fin de archivo, el último registro está completo; 02 = DT^ muy pequeño para el registro, lectura no completa; 03 = fin de archivo, ha leído un registre parcialmente.

La operación almacena en el CX el número real de registros a leer y establece el campo de registro relativo en el FCB y los campos de bloque y registro actuales para el registro siguiente.

## E/S ABSOLUTA DE DISCO

Puede utilizar la INT 25H y la 26H del DOS para lecturas y escrituras *absolutas* para procesar un disco de manera directa, por ejemplo, para recuperar un archivo dañado. En este caso, no define manejadores de archivo o FCB y pierde las ventajas de manejo de directorio y bloqueo y desbloqueo de registros que tiene con la INT 21H del DOS. Observe que la función 44H de la INT 21H (estudiada en el capítulo 18) proporciona un servicio similar y, de acuerdo con las revistas de Microsoft, ha sustituido a las INT 25H y 26H.

Como estas operaciones tratan de leer todos los registros como si fueran el tamaño de un sector, accesa de manera directa a un sector completo o un bloque de sectores. El direccionamiento de disco es en términos de número relativo de registro (sector relativo). Para determinar un número relativo de registro en discos flexibles de doble lado con nueve sectores por pista, cuente cada sector desde la pista 0, sector 1, como sigue:

| PISTA | SECTOR | NÚMERO RELATIVO DE REGISTRO |
|---|---|---|
| 0 | 1 | 0 (el primer sector en el disco) |
| 0 | 2 | 1 |
| 1 | 1 | 9 |
| 1 | 9 | 17 |
| 2 | 9 | 26 |

Una fórmula conveniente para determinar un número relativo de registro en discos flexibles con nueve sectores es

Número relativo de sector = (pista 9) + (sector -1)

Por tanto, el número relativo de registro para la pista 2, sector 9 es

(2 x 9) + (9 - 1) = 18 + 8 = 26

A continuación está el código necesario para particiones de disco de menos de 32 MB:

```
MOV  AL,#unidad          ,-0 para A,  1 para B,  etc.
MOV  BX.direcc           /Transfiere dirección
MOV  CX, sector          ,-Número de sectores para leer/escribir
MOV  DX,#sector          ;Inicio número relativo de sector
INT  25H o 26H           ;DOS,  lectura o escritura absoluta
POPF                     ,-Saca las banderas
JC   error
```

Las operaciones absolutas de lectura/escritura en disco destruyen todos los registros excepto los registros de segmento y emplean la bandera de acarreo para indicar una operación exitosa (0) o no exitosa (1). Una operación no exitosa regresa uno de los siguientes códigos diferentes de cero en el AL:

| | | |
|---|---|---|
| | 10000000 | Conexión falló al responder |
| • | 01000000 | Operación de búsqueda falló |
| | 00001000 | Incorrecta lectura de CRC en disco flexible |
| | 00000100 | Sector solicitado no encontrado |
| | 00000011 | Intento de escribir en un disco protegido contra escritura |
| | 00000010 | Otro error |

La operación INT empuja las banderas en la pila. Puesto que las banderas originales aún están en la pila antes de regresar de la operación, debe sacarlas después de examinar la bandera de acarreo.

Desde la versión DOS 4.0 puede usar las INT 25H y 26H para accesar particiones de disco que excedan 32MB. El AL y el CX todavía son usados de la misma manera. El DX no es utilizado, y el BX apunta a un bloque de parámetros de 10 bytes descrito como sigue:

| BYTES | DESCRIPCIÓN |
|---|---|
| 00H-03H | Número de sector de 32 bits ni |
| 04H-05H | Número de sectores de lectura/escritura |
| 06H-07H | Desplazamiento del búfer |
| 08H-09H | Segmento del búfer |

## PUNTOS CLAVE

- Muchos de los servicios de disco del DOS hacen referencia a una cadena ASCIIZ que consiste en una ruta de directorio seguido por un byte de ceros hexadecimales.
- Tras un error, muchas de las funciones de disco del DOS ponen en uno la bandera de acarreo y regresan un código de error en el AX.
- El DOS mantiene un apuntador a archivo por cada archivo que un programa está procesando. Las operaciones de creación y apertura establecen el valor del apuntador de un archivo en cero, la posición de inicio del archivo.
- Las funciones para crear y abrir regresan un manejador de archivo que se utiliza para subsecuente acceso al archivo.
- Cuando se escribe en un archivo se utiliza primero la función para crear 3CH; cuando se lee un archivo se emplea inicialmente la función 3DH.
- Un programa que ha terminado de escribir en un archivo debe cerrarlo, de modo que el DOS pueda actualizar el directorio.
- Un programa que utiliza las funciones originales de la INT 21H del DOS para E/S de disco define un bloque de control de archivo (FCB) para cada archivo que accesa.
- Un bloque de FCB consiste en 128 registros. El número del bloque actual, combinado con el número de registro actual, indica el registro del disco que será procesado. Las entradas en el FCB para el bloque actual, tamaño del registro y número de registro relativo son almacenados en secuencia inversa de bytes.

- El área de transferencia del disco (DTA) es la localidad del registro que será escrito o leído. Se tiene que inicializar cada DTA en un programa antes de ejecutar una operación de lectura o escritura.
- Las INT 25H y 26H del DOS proporcionan operaciones absolutas de lectura y escritura en disco, pero no proveen de manejo automático del directorio, operaciones de fin de archivo o bloqueo y desbloqueo de registros.

## PREGUNTAS

De las preguntas siguientes, las primeras 10 conciernen a operaciones en disco que implican el manejo de archivos y el resto implican operaciones de FCB en disco.

**17-1.** ¿Cuáles son los códigos de error que se regresan para (a) archivo no encontrado; (b) manejador no válido?

**17-2.** Defina una cadena ASCIIZ llamada PATH1 para un archivo con nombre CUST.LST en la unidad C.

**17-3.** Para el archivo de la pregunta 17-2, proporcione las instrucciones para (a) definir un elemento con nombre CUSTHAN para el manejador del archivo; (b) crear el archivo; (c) escribir un registro desde CUSTOUT (128 bytes), y (d) cerrar el archivo. Pruebe por si hay errores.

**17-4.** Para el archivo de la pregunta 17-3, codifique instrucciones para (a) abrir el archivo y (b) leer registros a CUSTIN. Pruebe por si hay errores.

**17-5.** ¿Bajo qué circunstancias debe cerrar un archivo que sólo es usado para entrada?

**17-6.** Corrija el código de la figura 17-4 de modo que el usuario pueda introducir desde el teclado un nombre de archivo, el cual utilice el programa para localizar el archivo y desplegar su contenido. Estipule cualquier número de peticiones y que con sólo presionar la tecla Enter se indique fin de la entrada.

**17-7.** Escriba un programa que permita al usuario ingresar desde una terminal números de parte (tres caracteres), descripciones de las partes (12 caracteres) y precios unitarios (xxx.xx). El programa es para usar manejadores de archivo para crear un archivo en disco que contenga esta información. Recuerde convertir el precio de ASCII a binario. A continuación se ve una muestra de un archivo de entrada:

| PARTE | DESCRIPCIÓN | PRECIO |
|---|---|---|
| 023 | Ensambladores | 00315 |
| 024 | Conexiones | 00430 |
| 027 | Compiladores | 00525 |
| 049 | Compresores | 00920 |
| 114 | Extractores | 11250 |
| 117 | Transporte | 00630 |
| 122 | Elevadores | 10520 |
| 124 | Procesadores | 21335 |
| 127 | Etiquetadores | 00960 |
| 232 | Depositarios | 05635 |
| 999 | | 00000 |

17-8. Escriba un programa que muestre el contenido de los archivos de la pregunta 17-7. Tendrá que convertir el número binario a formato ASCII para el precio.

17-9. Utilice el archivo creado en la pregunta 17-7 para los siguientes requisitos: (a) que el programa lea los registros en una tabla en memoria; (b) que el usuario pueda ingresar desde el teclado el número de parte y la cantidad; (c) que el programa busque en la tabla el número de parte; (c) que si encuentra el número de parte, el programa utilice la tabla de precios para calcular el valor de la parte (cantidad x precio); (e) que el programa despliegue la descripción y el valor calculado.

17-10. Corrija el programa de la pregunta 17-8, de manera que realice procesamiento directo. Defina una tabla con los números válidos de números de parte. Permita al usuario ingresar un número de parte, que el programa localiza en la tabla. Utilice el desplazamiento en la tabla para calcular el desplazamiento en el archivo y emplee la función 42H para mover el apuntador del archivo. Despliegue la descripción y el precio. Permita al usuario ingresar la cantidad vendida; calcule y despliegue el monto de la venta (cantidad x precio).

17-11. Proporcione las operaciones completas de funciones del DOS para las operaciones siguientes con FCB: (a) crear; (b) establecer DTA; (c) escritura secuencial; (d) abrir; (e) lectura secuencial.

17-12. Un programa utiliza el tamaño de registro, que establece por omisión la operación de abrir de FCB. (a) ¿Cuántos registros contendría un sector? (b) Suponiendo tres pistas con nueve sectores por pista, ¿cuántos registros contendría un disco? (c) Si el archivo del inciso (b) se fuera a leer de manera secuencial, ¿cuántos accesos físicos a disco ocurrirían?

# CAPÍTULO 18

# Procesamiento en disco: II—Operaciones del DOS para soporte de discos y archivos

OBJETIVO

Examinar las distintas operaciones implicadas en el uso de unidades de disco y archivos.

## INTRODUCCIÓN

Este capítulo introduce varias operaciones útiles implicadas en el manejo de unidades de disco, el directorio, la FAT y los archivos en disco.

**OPERACIONES PARA MANEJO DE UNIDADES DE DISCO**

| | |
|---|---|
| ODH | Restablecer unidad de disco |
| OEH | Seleccionar unidad por omisión |
| 19H | Obtener unidad por omisión |
| 1BH, 1CH | Obtener información de la unidad |
| 1FH | Obtener DPB por omisión |
| 2EH | Establecer/restablecer verificación de disco |
| 32H | Obtener DPB |
| 36H | Obtener espacio libre en disco |
| 4400H | Obtener información del dispositivo |
| 4401H | Establecer información del dispositivo |

| | |
|---|---|
| 4404H | Leer datos de control desde la unidad |
| 4405H | Escribir datos de control a la unidad |
| 4406H | Verificar estado de la entrada |
| 4407H | Verificar estado de la salida |
| 4408H | Determinar si es medio removible para dispositivo |
| 440DH, Código secundario 41H | Escribir sector del disco |
| 440DH, Código secundario 61H | Leer sector del disco |
| 440DH, Código secundario 42H | Formatear pista |
| 440DH, Código secundario 46H | Establecer identificación de medios |
| 440DH, Código secundario 60H | Obtener parámetros de dispositivo |
| 440DH, Código secundario 66H | Obtener identificación de medios |
| 440DH, Código secundario 68H | Percibido a tipo de medio |
| 54H | Obtener estado de verificación |
| 59H | Obtener error ampliado |

| OPERACIONES PARA MANEJAR ARCHIVOS EN DISCO | | OPERACIONES PARA MANEJAR EL DIRECTORIO Y LA FAT | |
|---|---|---|---|
| 29H | Análisis gramatical del nombre de archivo | | |
| 41H | Borrar archivo | 39H | Crear subdirectorio |
| 43H | Obtener/establecer atributo de archivo | 3AH | Eliminar subdirectorio |
| 45H, 46H | Duplicar manejador de archivo | 3BH | Cambiar directorio actual |
| 4EH, 4FH | Encontrar coincidencia de archivo | 47H | Obtener directorio actual |
| 56H | Renombrar archivo | | |
| 57H | Obtener/poner fecha/hora | | |
| 5AH, 5BH | Crear archivo temporal/nuevo | | |

Los códigos de error mencionados en este capítulo se refieren a la lista de la figura 17-1.

## OPERACIONES PARA MANEJO DE UNIDADES DE DISCO

### INT 21H, función ODH: Restablecer unidad de disco

Por lo común, al cerrar un archivo de manera adecuada esta **función** escribe todos los registros restantes y actualiza el directorio. En circunstancias especiales, como pasos entre programas c una condición de error, el programa puede necesitar restablecer un disco. La función ODH del DOS salta todos los búfers de archivo (la operación no cierra de manera automática los archivos ni regresa valores):

```
MOV AH,ODH      ;Petición para restablecer disco
INT 21H         ;Llama al DOS
```

### ENT 21H, función OEH: Seleccionar unidad por omisión

El objetivo principal de la función OEH del DOS es seleccionar una unidad como la actual poi omisión. Coloque el número de unidad en el DL, donde 0 = unidad A, 1 = B, y así sucesiva mente:

```
MOV  AH,OEH       ;Petición para designar por omisión
MOV  DL,02        ;  la unidad C
INT  21H          ;Llama al DOS
```

La operación regresa el número de unidades (todos los tipos, incluyendo los discos RAM virtuales) al AL. Puesto que el DOS requiere al menos dos unidades lógicas A y B, regresa el número 02 para un sistema con una sola unidad. (Utilice la INT 11H para determinar el número real de unidades.)

### INT 21H, función 19H: Obtener la unidad de disco por omisión

La función 19H del DOS determina la unidad de disco por omisión:

```
MOV  AH, 19H       ,-Obtiene  la unidad por omisión
INT  21H           ,Llama al DOS
```

La operación regresa un número de unidad en el AL, donde 0 = A, 1 = B, y así sucesivamente. Puede mover este número de forma directa a su programa accesando un archivo desde la unidad por omisión, aunque algunas operaciones suponen que 1 = unidad A y 2 = unidad B.

### INT 21H, función 1BH: Obtiene información de la unidad por omisión

Esta función regresa información acerca de la unidad por omisión:

```
MOV   AH,1BH      ;Petición de información
INT   21H         ;Llama al DOS
```

Ya que la operación cambia el DS, debe guardarlo (PUSH) en la pila antes llamar a la interrupción y sacarlo (POP) después de la interrupción. La operación ahora ha sido reemplazada por la función 36H. Una operación 1BH exitosa regresa la información siguiente:

| | |
|---|---|
| AL | Número de sectores por grupo |
| BX | Apuntador (DS:BX) al primer byte (descriptor de medios) en la FAT |
| CX | Tamaño del sector físico, por lo común 512 |
| DX | Número de grupo en el disco |

El producto del AL, CX y DX da la capacidad del disco. Una operación 1BH no exitosa regresa FFH en el AL.

### INT 21H, función 1CH: Obtener información de una unidad específica

Esta función regresa información acerca de una unidad específica. Inserte el número de unidad requerida en el DL, donde 0 = por omisión, 1 = A, y así sucesivamente:

```
MOV  AH,1CH       ;Petición de información
MOV  DL,drive     ,-Número de dispositivo
INT  21H          ;Llama al DOS
```

La operación es idéntica a la función 1BH y también está reemplazada por la función 36H.

### INT 21H, función 1FH: Obtener bloque de parámetros de la unidad por omisión (DPB)

El bloque de parámetros de la unidad es un área de datos que contiene la siguiente información de bajo nivel acerca de la estructura de la unidad:

| DESPLAZAMIENTO | TAMAÑO | CONTENIDO |
|---|---|---|
| OOH | Byte | Número de unidad (0 = A, etc.) |
| 01H | Byte | Unidad lógica del controlador |
| 02H | Palabra | Tamaño del sector, en bytes |
| 04H | Byte | Sectores por grupo menos 1 |
| 05H | Byte | Sectores por grupo (potencia de 2) |
| 06H | Palabra | Primer sector relativo de la FAT |
| 08H | Byte | Copias de la FAT |
| 09H | Palabra | Número de entradas en el directorio raíz |
| OBH | Palabra | Primer sector relativo del primer grupo |
| ODH | Palabra | Número más alto de grupo más 1 |
| OFH | Palabra | Sectores ocupados por cada FAT |
| 11H | Palabra | Primer sector relativo del directorio |
| 13H | Palabra doble | Dirección del controlador de dispositivo |
| 17H | Byte | Descriptor de medios |
| 18H | Byte | Bandera de acceso (0 si el disco fue accesado) |
| 19H | Palabra doble | Apuntador al siguiente bloque de parámetros |
| 1DH | Palabra | Último grupo asignado |
| 1FH | Palabra | Número de grupo libre |

Guarde (PUSH) en la pila el DS antes de emitir esta función y sáquelo (POP) de la pila al regresar de la función. La operación no tiene parámetros. Una operación válida limpia el Al y regresa una dirección en el DX:BX que apunta al DBP de la unidad por omisión. Si hay un error, el AL tiene FFH. Véase también la función 32H.

### INT 21H, función 2EH: Establecer/restablecer la verificación de escritura en disco

Esta función le permite verificar las operaciones de escritura en disco, es decir si la información se escribió de manera apropiada. La operación activa un interruptor que le indica al sistema que verifique la redundancia cíclica (CRC) del controlador del disco, una forma sofisticada de verificación de paridad. Al cargar el AL con 00, desactiva la verificación y con 01 la activa. El interruptor permanece con su estado hasta que otra operación lo cambia. A continuación está un ejemplo:

```
MOV  AH,2EH      ,-Petición para verificar  (o MOV AX,2E01H)
MOV  AL,01       ;Activa la verificación
INT  21H         ;Llama al DOS
```

La operación no regresa valor alguno, ya que sólo activa un interruptor. En consecuencia, el sistema responde a operaciones no válidas de escritura. Puesto que es raro que una unidad de disco registre información de manera incorrecta y que la verificación provoque un retardo, la operación es muy usada cuando la información registrada es especialmente crítica. Una función relacionada, 54H, envía el estado actual del interruptor de verificación.

### INT 21H, función 32H: Obtener un bloque de parámetros de la unidad (DPB)

Para obtener el DPB, cargue el número de unidad en el DX (donde 0 = por omisión, 1 = A, etc.). (Véase la función 1FH; esta función es idéntica a la 32H, con excepción de la petición de una unidad específica.)

### INT 21H, función 36H: Obtener un espacio libre en el disco

Esta función envía la información acerca del espacio en un dispositivo de disco. Cargue el número de unidad en el DL (0 = por omisión, 1 = A, 2 = B, etc.):

```
MOV  AH,3SH      ;Petición del espacio en disco
MOV  DL,0        ; para la unidad por omisión
INT  21H         ,-Llama al DOS
```

Una operación exitosa regresa lo siguiente:

AX = Número de sectores por grupo
BX = Número de grupos disponibles
CX = Número de bytes por sector
DX = Número total de grupo en el dispositivo

El producto de AX, CX y DX da la capacidad del disco. Para un número no válido de dispositivo, la operación regresa FFFFH en el AX. La operación no pone en uno o cero la bandera de acarreo.

### INT 21H, función 44H: Control de E/S para dispositivos

Este servicio elaborado, IOCTL, comunica información entre un programa y un dispositivo abierto. El servicio también incluye varias operaciones que no se estudian aquí. Cargue un número de subfunción en el AL para solicitar una de diferentes acciones. Una operación válida pone en cero la bandera de acarreo. Un error, como un manejador no válido de archivo, la pone en uno y regresa un código de error estándar al AX. A continuación se presentan las subfunciones de IOCTL.

### INT 21H, función 4400H: Obtener información del dispositivo

Esta operación regresa información acerca de un archivo o un dispositivo:

```
MOV  AX,4400H       ,-Petición de información del dispositivo
MOV  BX,handle      ,-Manejador de archivo o dispositivo
INT  21H            ;Llama al DOS
```

Una operación válida pone en cero la bandera de acarreo y regresa un número en el DX, donde el bit 7 = 0 significa que el manejador es de un archivo y bit 7 = 1 significa que es de un dispositivo, Los otros bits tienen este significado:

**ARCHIVO (BIT 7 = 0):**

| | |
|---|---|
| 0-5 | Número de unidad (0 = A, 1 = B, etc.) |
| 6 | 1 = archivo no se escribió |

**DISPOSITIVO (BIT 7 = 1):**

| | |
|---|---|
| 0 | Entrada estándar de la consola |
| 1 | Salida estándar a la consola |
| 2 | Dispositivo nulo |
| 3 | Dispositivo de reloj |
| 4 | Dispositivo especial |
| 5 | 0 = modo ASCII, 1 = modo binario |
| 6 | Para entrada, 0 = fin del archivo regresado, si el dispositivo es leído |

Un error pone en uno la bandera de acarreo y regresa en el AX el código 01, 05 o 06.

**INT 21H, función 4401H: Establece información del dispositivo**

Esta función carga el manejador de archivo en el BX y el bit de configuración en el DL para los bits 0-7, como se mostró para la subfunción OOH. La operación establece la información del dispositivo de acuerdo con esto. Un error pone en uno la bandera de acarreo y regresa el código 01, 05, 06 o ODH en el AX.

**INT 21H, función 4404H: Leer datos de control de la unidad**

Esta operación lee los datos de control de un controlador de dispositivo de bloque (unidad de disco). En el BL, carga la unidad (0 = por omisión, 1 = A, etc.), el número de bytes a leer en el CX y la dirección del área de datos en el DX. Una operación exitosa regresa al AX el número de bytes transferidos. Un error pone en uno la bandera de acarreo y regresa el código 01, 05 o ODH en el AX.

**INT 21H, función 4405H: Escribir datos de control en la unidad**

Esta operación escribe datos de control en un controlador de dispositivo de bloque. La configuración **es** la misma que para la función 4404H.

**INT 21H, función 4406H: Verificar estado de la entrada**

Este servicio verifica si un archivo o dispositivo está listo para entrada. Cargue el manejador en el BX. Una operación válida regresa uno de los siguientes códigos en el AL:

- Dispositivo: OOH = no está preparado, FFH = preparado
- Archivo: OOH = EOF alcanzado, FFH = EOF no alcanzado

Un error pone en uno la bandera de acarreo y regresa el código 01, 05 o 06 en el AX.

**INT 21H, función 4407H: Verifica estado de la salida**

Este servicio verifica si un archivo o dispositivo está preparado para enviar salida de resultados, Una operación válida regresa uno de los siguientes en el AL:

- Dispositivo: 00H = no está preparado, FFH = preparado
- Archivo: 00H = preparado, FFH = preparado

Un error pone en uno la bandera de acarreo y regresa el código 01, 05 o 06 en el AX.

### INT 21H, función 4408H: Determina si hay medio removible para el dispositivo

Este servicio determina si el dispositivo contiene un medio removible, como un disco flexible. Cargue el BL con el número de unidad (0 = por omisión, 1 = A, etc.). Una operación válida pone en cero la bandera de acarreo y regresa uno de los códigos siguientes en el AX:

- 00H = dispositivo removible o 01FH = dispositivo fijo

Un error pone en uno la bandera de acarreo y en el AX regresa el código 01 o OFH (número no válido de unidad).

### INT 21H, función 440DH, código secundario 41H: Escribir sector de disco

Esta operación escribe datos del búfer a uno o más sectores en disco. Cargue estos registros:

```
MOV  AX,440DH          ;IOCTL para dispositivo de bloque
MOV  BX,dríve          ;Unidad (0 = por omisión, 1 = A, etc.)
MOV  CH,08H            ,-Categoria del dispositivo = 08H
MOV  CL,41H            ;Código secundario = escribir pista
LEA  DX,devblock       /Dirección de dispositivo de bloque
INT  21H               ,-Llama al DOS
```

El DX apunta a un bloque de dispositivo con el formato siguiente:

```
devblock   LABEL  BYTE
specfune   DB     0          /Función especial (cero)
rwhead     DW     cabeza     /Cabeza lectura/escritura
rwcyl      DW     cilindro   /Cilindro
rwsectl    DW     sector     /Sector inicial
rwsects    DW     número     /Número de sectores
rwbuffr    DW     búfer      /Desplazamiento del búfer
           DW     SEG DATA   /Dirección del segmento de datos
```

La entrada rwbuffr proporciona la dirección del búfer en el formato segmento desplazamiento (DS:DX), aunque codificado en secuencia inversa de palabra. El operador SEG indica la definición de un segmento, en este caso del segmento de datos, _DATA. El búfer identifica el área de datos que será escrita y debe ser de la longitud del número de sectores x 512, como

```
WRBUFFER DB 1024 DUP (?)  /Búfer de salida
```

Una operación exitosa pone en cero la bandera de acarreo y escribe los datos. De otra manera, 1: operación pone en uno la bandera de acarreo y regresa el código de error 01, 02 o 05 en el AX

**INT 21H, función 440DH, código secundario 42H: Fomatear pista**

Para usar esta función a fin de formatear pistas, designe estos registros:

```
MOV   AX,44ODH        ;Petición de servicio de disco
MOV   BX,drive        ,-Unidad (0 = por omisión, 1 = A, etc
MOV   CH, 08          ,•Categoría del dispositivo (08)
MOV   CL,42H          ;Código secundario = formatear pista
LEA   DX,block        ;Dirección del bloque (DS:DX)
INT   21H             ;Llama al DOS
```

El DX apunta a un bloque con el formato siguiente:

```
blkname   LABEL  BYTE
specfun   DB     0        ;Función especial, código 0
diskhd    DW     ?        ;Cabeza de disco
cylindr   DW     ?        ;Cilindro
tracks    DW     ?        ;Número de pistas
```

Una operación exitosa pone en cero la bandera de acarreo y formatea las pistas. De otn forma, la operación pone en uno la bandera de acarreo y regresa el código de error 01, 02 o 05 er el AX.

**INT 21H, función 440DH, código secundario 46H: Establecer identificación de medio**

Para que esta función designe la identificación de medio, designe estos registros:

```
MOV  AX,440DH        ;Petición de servicio de disco
MOV  BX,drive        ,-Unidad (0 = por omisión, 1 = A, etc.)
MOV  CH, 08          ;Categoría del dispositivo (08)
MOV  CL,46H          ;Código secundario = establecer ID del medio
LEA  DX,block        ,-Dirección del bloque (DS:DX)
INT  21H             ,•Llama al DOS
```

El DX apunta a un bloque de medio con el formato siguiente:

```
blkname   LABEL   BYTE
infolev   DW      0              ;Nivel de información = 0
serialn   DD      ??             ,-Número de serie
```

```
volabel   DB      11 DUP (?)       Etiqueta de volumen
filetyp   DB      8 DUP (?)       ;Tipo de FAT
```

El campo filetyp contiene el valor ASCII FAT12 o FAT16, con blancos al final. Una operación exitosa pone en cero la bandera de acarreo y establece la identificación. De otra forma, la operación pone en uno la bandera de acarreo y regresa el código de error 01, 02, 05 en el AX. (Véase también la función 440DH, código secundario 66H.)

**INT 21H, función 440DH, código secundario 60H:**
**Obtener parámetros del dispositivo**

Para que esta función obtenga los parámetros de dispositivo, establezca estos registros:

```
MOV  AX,440DH        ;Petición de servicio de disco
MOV  BX,device       /Unidad (0 = por omisión, 1 = A, etc.)
MOV  CH,08           /Categoría del dispositivo (08)
MOV  CL,SOH          ,•Código secundario = obtener parámetros
LEA  DX,block        /Dirección del bloque (DS:DX)
INT  21H             ;Llama al DOS
```

apunta a un bloque de parámetro de dispositivo con el formato siguiente:

```
specfun  DB      7       ,•Función especial (0 o 1)
devtype  DB              ;Tipo de dispositivo
devattr  DW      7       /Atributo del dispositivo
cylindr  DW      7       ;Número de cilindros
medityp  DB      7       /Tipo de medio
bytesec  DW      7       ;Bytes por sector
secclus  DB      7       ;Sectores por grupo
ressect  DW      O       ,-Número de sectores reservados
fats     DB      7       ;Número de FAT
rootent  DW      7       ;Número de entradas en el directorio raíz
sectors  DW      7       ;Número total de sectores
mediads  DB      7       /Descriptor de medios
fatsecs  DW      7       ,-Número de sectores por FAT
sectrak  DW      7       /Sectores por pista
heads    DW      7       /Número de cabezas
```

```
hidsect  DD              Número de sectores ocultos
exsects  DD             ;Número de sectores si campo de sectores = 0
```

Si el campo specfun es 0, la información es acerca del medio por omisión en la unidad; si es 1, la información es acerca del medio actual. Una operación exitosa pone en cero la bandera de acarreo y envía la información. De otra forma, la operación pone en uno la bandera de acarreo y regresa el código de error 01, 02 o 05 en el AX.

**INT 21H, función 440DH, código secundario 61H: Leer sector de disco**

Esta operación lee información de uno o más sectores en disco y la envía a un búfer. Ponga el CL con **el** código secundario 61H; los detalles técnicos para la operación son idénticos a aquellos para el código secundario 41H, el cual escribe sectores. La figura 18-1 ilustra la función.

**INT 21H, función 440DH, código secundario 66H: Obtener identificación de medios**

Para que esta función obtenga la identificación de medios, establezca estos registros:

```
MOV  AX,440DH       ;Petición de servicio de disco ,
MOV  BX,device      ;Unidad (0 = por omisión, 1 = A, etc.)
MOV  CH, 08         ,-Categoría del dispositivo (08)
MOV  CL,66H         ;Código secundario = obtener identificación de medios
LEA  DX,block       /Dirección del bloque (DS:DX)
INT  21H            /Llama al DOS
```

El DX apunta a un bloque de medio con el formato siguiente:

```
blkname  LABEL  BYTE
infolev  DW     0           Nivel de información = 0
serialn  DD     ?           Número de serie
volabel  DB     1 DUP (?)   Etiqueta de volumen
filetyp  DB     8 DUP (?)   Tipo de FAT
```

Una operación exitosa pone en cero la bandera de acarreo y establece la identificación. El campo filetyp contiene el valor ASCII FAT12 o FAT16, con espacios en blanco al final. De otra manera, la operación pone en uno la bandera de acarreo y regresa el código de error 01, 02 o 05 eh el AX. (Véase también la función 440DH, código secundario 46H.)

**INT 21H, función 440DH, código secundario 68H: tipo de medio percibido**

Para utilizar esta función a fin de obtener el tipo de medio, establezca estos registros:

```
MOV  AX,440DH           /Petición de servicio de disco
MOV  BX,drive           /Unidad (0 = por omisión, 1 = A, etc.)
```

```
MOV CH, 0 8          /Categoría del dispositivo (08)
MOV  CL.68H          ;Código secundario = obtener tipo de medio
LEA  DX,block        ,-Dirección del bloque (DS:DX)
INT  21H             ;Llama al DOS
```

El DX apunta a un bloque de medio de dos bytes para recibir información en el formato siguiente:

```
default  DB  ?  ;01 para valor por omisión,  02 para otros
medatyp  DB  ?  ;Disco-02 = 720K,  07 = 1.44MB,  09 = 2.88MB
```

Una operación exitosa pone en cero la bandera de acarreo y establece el tipo. De otra forma, la operación pone en uno la bandera de acarreo y regresa el código de error 01 o 05 en el AX.

Otras operaciones de la función 44H IOCTL concernientes a la compartición de archivo están fuera del alcance de esta obra.

### INT 21H, **función** 54H: **Obtener estado de verificación**

Este servicio determina el estado de la bandera de verificación de escritura en disco. (Véase la función 2EH para activar este interruptor.) La operación regresa 00H al AL para verificación apagada o 01H para verificación activada. No existe condición de error.

### INT 21H, **función** 59H: **Obtener error ampliado**

Esta operación proporciona información adicional acerca de los errores después de la ejecución de los servicios de la INT 21H que ponen en uno la bandera de acarreo, los servicios de FCB que regresan FFH y de error en los manejadores con la INT 24H. La operación regresa lo siguiente:

- AX = Código de error ampliado
- BH = Clase de error
- BL = Acción sugerida
- CH = Posición

También, la operación pone en cero la bandera de acarreo y destruye el contenido de los registros CL, DI, DS, DX, ES y SI. Guarde en la pila (PUSH) todos los registros necesarios antes de esta interrupción y sáquelos de ella (POP) después.

**Código de error ampliado (AX).** Regresa alguno de los 90 o más códigos de error; 00 significa que la operación anterior INT 21H no tuvo error.

**Clase de error** (BH). Proporciona la información siguiente:

01H No hay recurso, como canal de almacenamiento
02H Situación temporal (no un error), como condición de archivo bloqueado que debe desaparecer
03H Falta de autorización apropiada
04H Error en el sistema de software, no de este programa
05H Fallo en el hardware

06H Grave error del DOS, no de este programa
07H Error en este programa, como petición inconsistente
08H Petición de elemento no encontrado
09H Formato inadecuado de archivo o de disco
OAH Archivo o elemento está bloqueado
OBH Error en disco, como error de CRC o disco incorrecto
OCH Archivo o elemento ya existe
ODH Clase de error desconocida

Acción (BL). Proporciona información sobre la acción a tomar:
01 Intente de nuevo unas cuantas veces; puede preguntar al usuario si se termina
02 Haga una pausa y reintente unas cuantas veces
03 Pregunte al usuario para que vuelva a ingresar una petición apropiada
04 Cierre archivos y termine el programa
05 Termine el programa inmediatamente; no cierre archivos
06 Ignore el error
07 Solicite al usuario que realice una acción (como cambiar de disco) e intente de nuevo la operación

Posición (CH). Proporciona información adicional sobre la localización del error:
01 Situación desconocida, no puede ayudar
02 Problema de almacenamiento en disco
03 Problema con la red
04 Problema de dispositivo en serie
05 Problema con la memoria

## PROGRAMA: LECTURA DE INFORMACIÓN DESDE LOS SECTORES

El programa de la figura 18-1 ilustra el uso de IOCTL función 44H, subfunción ODH, código secundario 61H. El programa lee información desde un sector y la envía a un búfer en memoria y despliega cada byte de entrada como una pareja de bytes hex. RDBLOCK en el segmento de datos de manera arbitraria especifica una cabeza, cilindro y sector inicial, que usted puede cambiar para sus propios propósitos. RDBUFFR define dos direcciones:

1. IOBUFFR es el desplazamiento del búfer de entrada, que proporciona un sector de datos.
2. SEGDATA utiliza el operador SEG para identificar la dirección del segmento de dato; para la operación IOCTL.

Los procedimientos principales en el segmento de código son:

B10READ Utiliza la operación IOCTL para leer el sector. La prueba por una lectun válida es hecha al regresar del procedimiento.
C10CONV Convierte cada byte en IOBUFFR en dos caracteres hexadecimales para des plegado. Dos instrucciones XLAT manejan la conversión de cada medio byte La rutina despliega 16 renglones de 32 parejas de caracteres.

```
TITLE     P18RDSCT (EXE)  Lee sector de disco
          .MODEL   SMALL
          .STACK   64

          .DATA
ROW       DB       00
COL       DB       00
XLATAB    DB       30H,31H,32H,33H,34H,35H,36H,37H,38H,3 9H
          DB       41H,42H,43H,44H,45H,46H
READMSG   DB       '*** Read error ***', ODH, OAH

RDBLOCK   DB       0                    Estructura
RDHEAD    DW       0                      del bloque
RDCYLR    DW       0
RDSECT    DW       8
RDNOSEC   DW       1
RDBUFFR   DW       IOBUFFR
          DW       SEG   DATA
IOBUFFR   DB       512 DUP(' ')         Área del sector del disco

.386
          .CODE
MAIN      PROC     FAR
          MOV      AX,©data             Inicializa
          MOV      DS,AX                  registros
          MOV      ES, AX                 de segmento
          CALL     Q10SCR               Limpia la pantalla
          CALL     Q2 0CURS             Coloca el cursor
          CALL     B10READ              Obtiene datos del sector
          JNC      A8 0                 Si la lectura es válida, pasar
          LEA      DX, READMSG            lectura no válida
          CALL     X10ERR
          JMP      A90
A80 :
          CALL     C10CONV              •Convertir y desplegar
A90 :
          MOV      AX,4C00H             ,-Salir al DOS
          INT      21H
MAIN      ENDP

                   Lee datos del sector:

BlOREAD   PROC     NEAR
          MOV      AX,440DH             IOCTL para dispositivo de bloque
          MOV      BX, 01               Unidad A
          MOV      CH, 08               Categoría del dispositivo
          MOV      CL,61H               Lee sector
          LEA      DX,RDBLOCK           Dirección de la estructura de bloque
          INT      21H
          RET
B10READ   ENDP
                   Desplegar datos del sector:

C10CONV   PROC     NEAR
          LEA      SI,IOBUFFR
C20 :
          MOV      AL, [SI]
          SHR      AL, 04                 Correr a la derecha un dígito hex
          LEA      BX,XLATAB              Designar dirección de la tabla
          XLAT                            Traducir el hex
          CALL     Q30DISPL
          INC      COL
          MOV      AL,[SI]
```

Figura 18-1 Lectura de sectores del disco

```
            AND     AL, OFH             ,-Borrar el dígito hex de la izquierda
            XLAT                        ,•Traducir el hex
            CALL    Q30DISPL
            INC     SI
            INC     COL
            CMP     COL,64
            JBE     C20
            INC     ROW
            MOV     COL,00
            CALL    Q20CURS
            CMP     ROW,16
            JBE     C20
            RET
C10CONV     ENDP
                    Recorrido de la pantalla:

Q10SCR      PROC    NEAR
            MOV     AX,0600H            ;Petición para recorrer
            MOV     BH,1EH              ,•Establece atributo
            MOV     CX,0000
            MOV     DX,184FH
            INT     10H
            RET
Q10SCR      ENDP
                    Establece cursor:

Q20CURS     PROC    NEAR
            MOV     AH,02H               Petición para colocar
            MOV     BH, 00                 el cursor
            MOV     DH, ROW                renglón      -
            MOV     DL,COL                 columna
            INT     10H
            RET
Q20CURS     ENDP
Q30DISPL    PROC    NEAR
            MOV     AH,02H              ;Petición para
            MOV '   DL, AL              ;   imprimir carácter
            INT     21H
            RET
Q30DISPL    ENDP

                    Desplegar mensaje de error de disco:

X10ERR      PROC    NEAR
            MOV     AH,40H               DX contiene la dirección
            MOV     BX, 01               Manejador
            MOV     CX, 20               Longitud
            INT     21H                    del mensaje
            INC     ROW
            RET
XiOERR      ENDP
            END     MAIN
```

**Figura 18-1** (continuación)

Puede mejorar este programa permitiendo al usuario solicitar sectores vía el teclado.

## OPERACIONES PARA MANEJAR EL DIRECTORIO Y LA FAT

### INT 21H, función 39H: Crear subdirectorio

Este servicio crea un subdirectorio, tal como lo hace el comando MKDIR del DOS. Cargue el DX con la dirección de una cadena ASCIIZ con la unidad y la ruta al directorio; es así de sencillo:

```
ASCstrg DB   'd:\pathname'  ,00H;Cadena ASCIIZ

        MOV AH,39H          /Petición para crear subdirectorio
        LEA DX,ASCstrg      /Dirección de la cadena ASCIIZ (DS:DX)
        INT 21H
```

Una operación exitosa pone en cero la bandera de acarreo; un error la pone en uno y regresa el código 03 o 05 en el AX.

**INT 21H, función 3AH: Eliminar subdirectorio**

Este servicio elimina un subdirectorio, tal como lo hace el comando RMDIR del DOS. Cargue el DX con la dirección de una cadena ASCIIZ con la unidad y la ruta al directorio (tome en cuenta que no puede eliminar el directorio actual o un subdirectorio con archivos):

```
ASCstrg DB   'd:\pathname'  ,00H/Cadena ASCIIZ

        MOV AH.3AH          /Petición para eliminar subdirectorio
        LEA DX,ASCstrg      /Dirección de la cadena ASCIIZ (DS:DX)
        INT 21H
```

Una operación exitosa pone en cero la bandera de acarreo; un error la pone en uno y regresa el código 03, 05 o 10H en el AX.

**INT 21H, función 3BH: Cambiar de directorio actual**

Este servicio cambia el directorio actual a uno que usted especifique, tal como lo hace el comando CHDIR del DOS. Cargue el DX con la dirección de una cadena ASCIIZ con la nueva unidad y la ruta del directorio:

```
ASCstrg DB   'd:\pathname'  ,00H /Cadena ASCIIZ

        MOV AH,3BH          /Petición para cambiar de directorio
        LEA DX,ASCstrg      /Dirección de la cadena ASCIIZ (DS:DX)
        INT 21H
```

Una operación exitosa pone en cero la bandera de acarreo; un error la pone en uno y regresa el código 03 en el AX.

**INT 21H, función 47H: Obtener el directorio actual**

La función 47H del DOS determina el directorio actual en cualquier unidad. Defina un espacio para el búfer suficientemente grande para contener el nombre de ruta más largo (64 bytes) y cargue su dirección en el SI. Identifique la unidad en el DL por 0 = por omisión, 1 = A, 2 = B, y así sucesivamente:

```
buffer  DB   64 DUP (20H)   ;Espacio de 64 bytes para el búfer

        MOV  AH,47H         ,-Petición para obtener el subdirectorio
        MOV  DL,drive       /Unidad
        LEA  SI,buffer      /Dirección del búfer (DS:DI)
        INT  21H
```

Una operación válida pone en cero la bandera de acarreo y envía el nombre del directorio actual (pero no la unidad) al búfer como una cadena ASCIIZ, como

```
ASSEMBLE\EXAMPLESO
```

Un byte con OOH identifica el final del nombre de la ruta. Si el directorio solicitado es el principal (raíz), el valor regresado es sólo un byte de OOH. De esta manera, puede obtener el nombre de la ruta actual a fin de accesar cualquier archivo en un subdirectorio. Un número no válido de unidad pone en uno la bandera de acarreo y regresa el código de error OFH en el AX.

### INT 21H, función 56H: Renombrar archivo o directorio

Véase esta función en la sección siguiente.

## PROGRAMA: DESPLIEGUE DEL DIRECTORIO

El programa de la figura 18-2 ilustra el uso de dos de las funciones descritas en la sección precedente. Los procedimientos realizan lo siguiente:

- **B10DRIV** Utiliza la función 19H para obtener la unidad por omisión en el registro AL. La unidad es regresada como 0 (para A), 1 (para B) y así sucesivamente. Para adaptar el número a su equivalente alfabético, sólo sume 41H, de modo que 00 se convierta en 41H (A), 01 se convierta en 42H (B), y así sucesivamente. Después el procedimiento despliega la letra de la unidad seguida por dos puntos y diagonal inversa (n:\).
- **C10PATH** Utiliza la función 47H para obtener el nombre de la ruta al directorio actual. El procedimiento prueba de forma inmediata por el delimitador OOH ASCIIZ, ya que un valor por omisión al directorio raíz enviaría sólo ese carácter. En otro caso, la rutina despliega cada carácter hasta el OOH.

El programa de forma intencional sólo contiene las características necesarias para que funcione; un programa completo incluiría, por ejemplo, el borrado de la pantalla y la utilización de colores.

## OPERACIONES PARA MANEJAR ARCHIVOS EN DISCO

Esta sección describe las operaciones del DOS que procesan archivos en disco.

```
TITLE   P18GETDR (COM)  Obtiene directorio actual
        .MODEL SMALL
        .CODE
        ORG     100H
BEGIN:  JMP     SHORT MAIN

PATHNAM DB      64 DUP(' ')         ;Nombre actual de la ruta

MAIN    PROC    NEAR
        CALL    B10DRIV             |Obtiene/despliega la unidad por omisión
        CALL    C10PATH             |Obtiene/despliega la ruta
        MOV     AH,10H              |Hace una pausa hasta que el usuario
        INT     16H                 |  presiona una tecla
        MOV     AX,4C00H            |Sale al DOS
        INT     21H
MAIN    ENDP

B10DRIV PROC    NEAR;
        MOV     AH,19H               Petición de la unidad por omisión
        INT     21H
        ADD     AL,41H              |Cambia el número hex a letra
        MOV     DL, AL              |  0=A, 1=B, etc.
        CALL    Q10DISP             |Despliega el número de unidad,
        MOV     DL,':'              |
        CALL    Q10DISP             • dos puntos,
        MOV     DL,'\'
        CALL    Q10DISP             ; diagonal inversa
        RET
B10DRIV ENDP

C10PATH PROC    NEAR;
        MOV     AH,47H               Petición de nombre de la ruta
        MOV     DL, 00
        LEA     SI,PATHNAM
        INT     21H
C20 :
        CMP     BYTE PTR [SI] , 00H |¿Fin del nombre de la ruta?
        JE      C90                 |  sí, salir
        MOV     AL, [SI]            |Despliega el nombre de la ruta
        MOV     DL, AL              |  un byte
        CALL    Q10DISP             |  a la vez
        INC     SI
        JMP     C2 0                ,- Repite
C90 :   RET
C10PATH ENDP

Q10DISP PROC    NEAR                ;DL se designa al inicio
        MOV     AH,02H              ;Petición para desplegar
        INT     21H
        RET
Q10DISP ENDP
        END     BEGIN
```

Figura 18-2 Obtiene directorio actual

### INT 21H, función 29H: Análisis gramatical del nombre de archivo

Este servicio convierte a una línea de comando, que contiene una especificación de archivo (filespec), de la forma del: nombre de archivo ext en el formato FCB. La función puede aceptar una especificación de archivo del usuario para copiar y suprimir archivos.

Cargue el registro SI (asociado con el DS) con la dirección de la especificación del archivo que será analizado, el DI ( asociado con el ES) con la dirección de un área en donde la operación genera el formato FCB y el AL con el valor en bit que controla el método del análisis gramatical:

```
MOV   AH,29H          ;Petición para análisis gramatical del nombre de archivo
MOV   AL,code         /Método de análisis gramatical
LEA   DI,FCBname      ;Dirección del FCB  (ES:DI)
LEA   SI,filespec     ,-Dirección de la especificación de archivo  (DS:SI)
INT   21H             ,-Llama al DOS
```

Los códigos para el método de análisis gramatical son:

| BIT | VALOR | ACCIÓN |
|---|---|---|
| 0 | 0 | Filespec empieza en la posición del primer byte. |
| 0 | 1 | Salta separadores (como blancos) para encontrar la filespec. |
| 1 | 0 | Coloca el byte de identificación en el FCB generado: sin unidad = 00, A = 01, B = 02, y así sucesivamente. |
| 1 | 1 | Cambia el byte de identificación de la unidad en el FCB generado sólo si la filespec analizada especifica una unidad. De esta manera, un FCB puede tener su propia unidad por omisión. |
| 2 | 0 | Cambia el nombre del archivo en el FCB como es requerido. |
| 2 | 1 | Cambia el nombre del archivo en el FCB, sólo si la filespec contiene un nombre válido de archivo. |
| 3 | 0 | Cambia la extensión del nombre del archivo como es requerido. |
| 3 | 1 | Cambia la extensión sólo si la filespec contiene una extensión válida. |
| 4-7 | 0 | Debe ser cero. |

Para datos válidos, la función 29H crea un formato FCB estándar para el nombre y extensión del archivo, con un nombre de archivo de ocho caracteres, rellenados con blancos si es necesario, una extensión de tres caracteres, rellenada con blancos si es necesario, y sin punto entre ellos.

La operación reconoce la puntuación estándar y convierte los comodines * y ? en una cadena de uno o más caracteres. Por ejemplo, PROG12.* se convierte en PROG12bb???. El AL regresa uno de los códigos siguientes:

OOH No se encontraron comodines
01H Comodines convertidos
FFH Unidad especificada no válida

Después de la operación, el DS:SI contiene la dirección del primer byte después de la filespec analizada gramaticalmente y el ES:DI contiene la dirección del primer byte del FCB. Para una operación fallida, el byte en DI+1 es un blanco, aunque la operación intenta convertir casi cualquier cosa que usted le envíe.

Para que esta operación funcione con manejadores de archivo, tiene que editar después el FCB para eliminar los blancos e introducir el punto entre el nombre y la extensión del archivo.

### INT 21H, función 41H: Borrar archivo

Esta función borra un archivo (que no sea de sólo lectura) desde un programa. Cargue en el DX la dirección de una cadena ASCIIZ con la ruta al dispositivo y el nombre del archivo, sin comodines:

```
ASCstrg DB   'd:\pathname' , 00H  /Cadena ASCIIZ


         MOV  AH,41H           ,-Petición para borrar
         LEA  DX,ASCstrg       /Dirección de la cadena ASCIIZ (DS:DX)
         INT  21H              /Llama al DOS
```

Una operación válida pone en cero la bandera de acarreo, marca el nombre del archivo como borrado en el directorio y libera en la FAT el espacio en disco asignado al archivo. Un error pone en uno la bandera de acarreo y regresa el código 02, 03 o 05 en el AX.

### INT 21H, función 43H: Obtener o establecer atributo de archivo

Puede utilizar esta operación tanto para obtener como para establecer un atributo de archivo en el directorio. La operación necesita la dirección de una cadena ASCIIZ con la unidad, ruta y nombre del archivo para el archivo requerido. (O utilice el directorio por omisión si no se da ninguna ruta.)

Para obtener el atributo del archivo, cargue el AL con el código 00. El ejemplo siguiente obtiene un atributo de archivo:

```
ASCstrgDB   'd:\pathname'  , 00H  /Cadena ASCIIZ


         MOV  AH,43H           /Petición
         MOV  AL, 0 0          ,• para obtener atributo
         LEA  DX,ASCstrg       /Dirección de la cadena ASCIIZ (DS:DX)
         INT  21H              /Llama al DOS
```

Una operación válida pone en cero la bandera de acarreo y regresa el atributo actual al CX (CH = 00 y CL = atributo):

| BIT | ATRIBUTO | BIT | ATRIBUTO |
|---|---|---|---|
| 0 | Sólo lectura | 3 | Etiqueta de volumen |
| 1 | Archivo oculto | 4 | Subdirectorio |
| 2 | Archivo de sistema | 5 | Archivo archivado |

Un error pone en uno la bandera de acarreo y regresa el código 02 o 03 en el AX.

Para establecer atributo de archivo, cargue el AL con el código 01 y coloque el (los) bit(s) de atributo en el CX. Puede cambiar los bits de archivo de sólo lectura, oculto, de sistema y archivado, pero no la etiqueta del volumen o de subdirectorio. El ejemplo siguiente establece los atributos de oculto y de archivado para un archivo:

```
MOV  AH.43H            ;Petición
MOV  AL,01             ; para establecer atributos-
MOV  CX,22H            ; de oculto y de archivado
LEA  DX.ASCstrg        /Cadena ASCIIZ  (DS:DX)
INT  21H               /Llama al DOS
```

Una operación válida pone en cero la bandera de acarreo y designa la entrada del directorio con los atributos en el CX. Una operación no válida pone en uno la bandera de acarreo y regresa el código 02, 03 o 05 al AX.

**INT 21H, función 45H: Duplicar un manejador de archivo**

Puede usar este servicio para dar a un archivo más de un manejador. Los usos de manejadores anteriores comparados con los nuevos son idénticos: los manejadores hacen referencia al mismo archivo, apuntador de archivo y área del búfer. Un uso es para solicitar un manejador de archivo y utilizar ese manejador para cerrar el archivo. Esta acción provoca que el DOS limpie el búfer y actualice el directorio. Entonces puede utilizar el manejador original del archivo para continuar el procesamiento del archivo. Un ejemplo del uso de la función 45H es el siguiente:

```
MOV AH,45H          /Petición para duplicar manejador
MOV BX.handle       /Manejador actual que será duplicado
INT  21H
```

Una -operación exitosa pone en cero la bandera de acarreo y regresa el nuevo manejador (el siguiente disponible) en el AX. Un error pone en uno la bandera de acarreo y regresa el código de error 04 o 06 al AX. (Véase también la función 46H.)

**INT 21H, función 46H: Forzar duplicación de un manejador de archivo**

Este servicio es similar a la función 45H, salvo que puede asignar un manejador de archivo específico. Usted podría utilizar este servicio, por ejemplo, para redireccionar la salida. Cargue el BX con el manejador original y el CX con el segundo manejador.

Una operación exitosa pone en cero la bandera de acarreo; un error la pone en uno y regresa el código de error 04 o 06 al AX. Algunas combinaciones pueden no funcionar; por ejemplo, el manejador 00 siempre es la entrada desde el teclado, 04 es la salida a una impresora y 03 (auxiliar) no puede ser redireccionada. (Véase también la función 45H.)

**INT 21H, función 4EH: Encontrar primer archivo que coincida**

Esta operación es similar (y se prefiere) a la función original 11H. Utilice la función 4EH para empezar a buscar en un directorio y 4FH para continuar la búsqueda. Tiene que definir un búfer

de 43 bytes para la operación, a fin de regresar la entrada localizada en el directorio y emitir la función 1AH (establecer DTA) antes de utilizar este servicio. Para iniciar la búsqueda, establezca el CX con el atributo del archivo de los nombres de archivo que serán regresados cualquier combinación de sólo lectura (bit 0), oculto (bit 1), de sistema (bit 2), de etiqueta de volumen (bit 3), directorio (bit 4) o de archivado (bit 5). Cargue el DX con la dirección de una cadena ASCIIZ con el nombre de la ruta; la cadena puede contener caracteres comodines ? y *:

```
DB    43 DUP (?)
DB    'ASCIIZ string', OOH

MOV   AH, 1AH           ;Petición para establecer DTA
LEA   DX,DTAname        ;Área para DTA (DS:DX)
INT   21H               ;Llama al DOS
MOV   AH,4EH            ;Petición primera coincidencia
MOV   CX,OOH            ;Atributo normal
LEA   DX,ASCstrg        ;Cadena ASCIIZ (DS:DX)
INT   21H               ;Llama al DOS
```

Una operación que localiza una coincidencia entre los bits de atributos pone en cero la bandera de acarreo y llena el DTA de 43 bytes (2BH) con lo siguiente:

| | |
|---|---|
| 00H-14H | Reservado por DOS para búsquedas subsecuentes |
| 15H | Atributo del archivo |
| 16H-17H | Hora del archivo |
| 18H-19H | Fecha del archivo |
| 1AH-1DH | Tamaño del archivo: palabra baja y después palabra alta |
| 1EH-2AH | Nombre y extensión como una cadena ASCIIZ, seguida por hex 00 |

Un error pone en uno la bandera de acarreo y regresa el código 02, 03 o 12H.

Un uso único para la función 4EH es determinar si una referencia es a un archivo o a un directorio. Por ejemplo, si el atributo regresado es 10H, la referencia es a un subdirectorio. También la operación regresa el tamaño del archivo. Por tanto, puede usar la función 4EH para determinar el tamaño de un archivo y la función 36H para verificar el espacio disponible para escribirlo.

### INT 21H, función 4FH: Encontrar el siguiente archivo que coincida

Esta operación es similar a la función original 12H. Primero utilice la función 4EH para empezar la búsqueda en un directorio y después la función 4FH para continuar la búsqueda. Si planea utilizar 4FH, no cambie el contenido del DTA (véase la función 4EH para el valor llenado en el DTA):

```
MOV   AH,4FH        ;Petición de siguiente coincidencia
INT   21H           ;Llama al DOS
```

Una operación exitosa pone en cero la bandera de acarreo y regresa al AX los códigos 00 (nombre de archivo encontrado) o 18 (no hay más archivos). Un error pone en uno la bandera de acarreo y regresa el código 02, 03 o 12H en el AX.

La figura 18-3 ilustra las funciones 4EH y 4FH.

### INT 21H, función 56H: Renombrar archivo o directorio

Este servicio puede renombrar un archivo o directorio desde un programa. Cargue el DX con la dirección de una cadena ASCIIZ con la unidad, ruta y nombre anteriores del archivo o directorio que será renombrado. Cargue el DI (realmente ES:DI) con la dirección de una cadena ASCIIZ con la unidad, ruta y nombre nuevos, sin comodines. Si usó números de unidad deben ser los mismos en ambas cadenas. Ya que las rutas no necesitan ser las mismas, la operación puede renombrar y mover el archivo a otro directorio en la misma unidad:

```
oldstrg  DB    'd:\oldpath\oldname', 00H
newstrg  DB    'd:\newpath\newname', 00H

         MOV   AH,5GH          Petición para renombrar archivo/directorio
         LEA   DX.oldstring    DS:DX
         LEA   DI,newstring    ES:DI
         INT   21H             Llama al DOS
```

Una operación exitosa pone en cero la bandera de acarreo; un error pone en uno la bandera de acarreo y regresa en el AX el código 02, 03, 05 u 11H.

### INT 21H, función 57H: Obtener/poner la fecha y hora del archivo

Este servicio permite a un programa obtener o poner la fecha y hora de un archivo abierto. Los formatos para la hora y fecha son los mismos que los del directorio:

| BITS PARA LA HORA | | BITS PARA LA FECHA | |
|---|---|---|---|
| 0BH-0FH | Horas | 09H-0FH | Año (relativo a 1980) |
| 05H-0AH | Minutos | 05H-08H | Mes |
| 00H-04H | Segundos | 00H-04H | Día del mes |

(Los segundos están en la forma del número de incrementos cada dos segundos, 0-29.) Cargue la petición (0 = obtener, 1 = poner) en el AL y el manejador de archivo en el BX. Para una petición de poner la hora y fecha, cargue la hora en el CX y la fecha en el DX. A continuación está un ejemplo:

```
         MOV   AH,57H          Petición de fecha/hora
         MOV   AL, 01          Poner fecha y hora
         MOV   BX,handle       Manejador de archivo
         MOV   CX,time  -      Hora nueva
         MOV   DX.date         Fecha nueva
         INT   21H
```

Una operación válida pone en cero la bandera de acarreo; la operación de obtener regresa la hora en el CX y la fecha en el DX, mientras que la operación para poner la hora y fecha cambia las entradas de fecha y de hora para el archivo. Una operación no válida pone en uno la bandera de acarreo y regresa en el AX el código de error 01 o 06.

### INT 21H, función 5AH: Crear un archivo temporal

Este servicio es útil para un programa que cree archivos temporales, en especial en redes, en el que los nombres de otros archivos pueden ser desconocidos y el programa sirve para evitar sobreescribir en ellos de manera accidental. La operación crea un archivo con un nombre único en la ruta.

Cargue el CX con el atributo necesario del archivo: cualquier combinación de sólo lectura (bit 0), oculto (bit 1), de sistema (bit 2), de etiqueta de volumen (bit 3), directorio (bit 4) o de archivado (bit 5). Cargue el DX con la dirección de una ruta ASCIIZ: la unidad (si es necesario), el subdirectorio (si hay), una diagonal inversa y OOH, seguida por 13 bytes para el nombre nuevo del archivo:

```
ASCpath  DB    'd:\pathname\',  OOH 13 DUP (20H)

         MOV   AH, 5AH          ,-Petición para crear archivo
         MOV   CX,atributte     /Atributo del archivo
         LEA   DX,ASCpath       ;Ruta ASCIIZ
         INT   21H
```

Una operación exitosa pone en cero la bandera de acarreo, envía el manejador de archivo al AX y añade el nombre nuevo del archivo a la cadena ASCIIZ, iniciando en el byte OOH. Un error pone en uno la bandera de acarreo y regresa el código 03, 04 o 05 en el AX.

### INT 21H, función 5BH: Crear un archivo nuevo

Este servicio crea un archivo sólo si el archivo nombrado no existe; por lo demás es idéntica a la función 3CH (crear archivo). Usted debe utilizar la función 5BH siempre que no quiera sobreescribir en un archivo. Una operación válida pone en cero la bandera de acarreo y regresa el manejador del archivo en el AX. Una operación no válida (incluyendo encontrar un nombre de archivo idéntico) pone en uno la bandera de acarreo y regresa el código 03, 04, 05 o 50H en el AX.

## PROGRAMA: BORRAR ARCHIVOS DE FORMA SELECTIVA

El programa de la figura 18-3 ilustra el uso de las funciones 4EH y 4FH del DOS para encontrar todos los nombres de archivo en el directorio por omisión y la función 41H borra los archivos seleccionados. El programa consiste en los procedimientos siguientes:

| | |
|---|---|
| MAIN | Llama a los procedimientos B10FIRST, C10NEXT, DIODISPLy E10DELET. |
| B10FIRST | Establece el DTA para la función 4EH y encuentra la primera entrada que coincida en el directorio. |
| C10NEXT | Encuentra las entradas subsecuentes en el directorio que coinciden. |
| D10DISPL | Muestra los nombres de las entradas que coinciden y pregunta si serán borradas. |
| E10DELET | Acepta una contestación Y (sí) para borrar el archivo, N (no) para conservarlo, o Enter para terminar el proceso y borrar los archivos requeridos. |

```
TITLE     P18SELDL (COM)   Selecciona y borra los archivos
CODESG    SEGMENT PARA 'Code'
          .MODEL SMALL
          .CODE
          ORG    100H
BEGIN:    JMP    MAIN
-------------------------------------------------------------------------
TAB       EQU    09
LF        EQU    10
CR        EQU    13
CRLF      DB     CR, LF, '$'
PATHNAM   DB     'F:\*.*', OOH
DELMSG    DB     TAB, 'Erase ','$'
ENDMSG    DB     CR, LF, 'No more directory entries', CR, LF, '$'
ERRMSG1   DB     'Invalid path/file', '$'
ERRMSG2   DB     'Write-protected disk','$'
PROMPT    DB     'Y = Erase, N = Keep, Ent = Exit', CR, LF, '$'
DISKAREA  DB     43 DUP(20H)
-------------------------------------------------------------------------
MAIN      PROC   NEAR              ;Procedimiento principal
          CALL   Q10SCRN           ;Limpia la pantalla
          CALL   Q20CURS           ,-Coloca el cursor
          CALL   B10FIRST          ,• en la entrada del directorio
          CMP    AX,OOH            ,-Si no hay entradas,
          JNE    A90               ,• salir
          LEA    DX,PROMPT         ,•Indicación inicial
          CALL   Q30LINE
A20:
          CALL   D10DISPL          /Despliega nombre de archivo
          CALL   E10DELET          ;Si se solicita, lo borra
          CMP    AL,OFFH           /¿Solicitud para terminar?
          JE     A90               /  sí, salir
          LEA    DX,CRLF           /Colocar el cursor en
          CALL   Q30LINE           ;  la línea siguiente
          CALL   C10NEXT           /Obtiene la siguiente entrada del
          CMP    AX,OOH            /¿Existen más entradas?
          JE     A20               /  si, repetir
A90:
          MOV    AX,4C00H          /Salir al DOS
          INT    21H
MAIN      ENDP

B10FIRST  PROC   NEAR
          MOV    AH,1AH            /Obtener el DTA para
          LEA    DX,DISKAREA       /  llamadas de la función
          INT    21H
          MOV    AH,4EH            /Localizar primer entrada
          MOV    CX,00             /  del directorio
          LEA    DX, PATHNAM       /Dirección de la cadena ASCIIZ
          INT    21H
          JNC    B90               /¿Operación válida?
          PUSH   AX                ," no,
          LEA    DX,ERRMSG1        /  mostrar mensaje
          CALL   Q3QLINE           /  final
          POP    AX
B90:      RET
B10FIRST  ENDP

C10NEXT   PROC   NEAR              /Lee entrada del directorio
          MOV    AH,4FH            /Obtiene la siguiente
          INT    21H
          CMP    AX,OOH            /¿Hay más entradas?
          JE     C90               :  sí, pasar
```

Figura 18-3 Selecciona y borra los archivos

```
            PUSH    AX                      no,
            LEA     DX,ENDMSG               mostrar mensaje
            CALL    Q30LINE                 final
            POP     AX
C90:        RET
C10NEXT     ENDP

D10DISPL    PROC    NEAR
            LEA     DX,DELMSG           /Mostrar mensaje para borrar
            CALL    Q30LINE
            LEA     SI,DISKAREA+1EH     ¡Inicio del nombre de archivo
D30:
            MOV     DL,[SI]             ;Obtener carácter
            CALL    Q40CHAR
            INC     SI                  Carácter siguiente
            CMP     BYTE PTR [SI],00    ¿Cero hex para detener?
            JNE     D30                   no, obtener carácter siguiente
            MOV     DL,'?'                sí, salir
            CALL    Q4OCHAR
            RET
D10DISPL    ENDP

E10DELET    PROC    NEAR
            MOV     AH,10H              Aceptar un carácter
            INT     16H                   respuesta (y/n)
            CMP     AL,ODH              ¿Carácter Enter?
            JE      E50                   sí, salir
            OR      AL,00100000B        Fuerza letra minúscula
            CMP     AL,'y'              ¿Petición de borrado?
            JNE     E90                   no, pasar
            MOV     AH,41H                sí
            LEA     DX,DISKAREA+1EH       dirección de la entrada borrada
            INT     21H                   del nombre de archivo
            JNC     E90                 ¿Borrado válido?
            LEA     DX,ERRMSG2            no, mostrar
            CALL    Q30LINE
E50:                                      mensaje de advertencia
            MOV     AL,OFFH
E90:        RET                         :Indicador del fin del proceso
            ENDP
E10DELET
            PROC    NEAR
Q10SCRN     MOV     AX,0600H            ;Petición para limpiar la pantalla
            MOV     BH,1EH              /Establece atributo
            MOV     CX,00
            MOV     DX,184FH
            INT     10H
            RET
Q10SCRN     ENDP

Q20CURS     PROC    NEAR
            MOV     AH,02H              Petición para
            MOV     BH,00                 colocar el cursor
            MOV     DH,00               Renglón 0
            MOV     DL,10               Columna 10
            INT     10H
            RET
Q20CURS     ENDP

Q30LINE     PROC    NEAR
            MOV     AH,09H              /Petición para desplegar una línea
            INT     21H                 /DX designa al inicio
            RET
Q30LINE     ENDP
```

Figura 18-3 (continuación)

```
Q4 OCHAR    PROC    NEAR
            MOV     AH,02H              /Petición para desplegar
            INT     21H                 ,-DL designa al inicio
            RET
Q4OCHAR     ENDP
            END     BEGIN
```

Figura 18-3 (continuación)

Como precaución, durante la prueba utilice archivos temporales copiados.

## PUNTOS CLAVE

- Las operaciones implicadas en el manejo de unidades de disco incluyen restablecer, seleccionar por omisión, obtener información de la unidad, obtener el espacio libre en el disco y el control extenso de la operación de E/S para dispositivos.
- Las operaciones implicadas en el manejo del directorio y de la FAT incluyen crear subdirectorio, eliminar subdirectorio, cambiar del directorio actual y obtener el directorio actual.
- Las operaciones implicadas en el manejo de archivos (diferentes de crear, abrir, leer y escribir) incluyen renombrar archivo, obtener/designar atributo, encontrar coincidencia de archivo y obtener/poner fecha/hora.

## PREGUNTAS

Utilice DEBUG para las primeras tres preguntas. Teclee el comando A 100 y las instrucciones necesarias. Examine los valores regresados en los registros.

18-1. Operaciones que requieren unidades de disco:

(a) Función 19H para determinar la unidad de disco por omisión.
**(b)** Función 1BH para información acerca de la actual unidad de disco por omisión.
(c) Función 1FH para información acerca de DPB por omisión.
**(d)** Función 36H para determinar la cantidad de espacio libre en el disco.
**(e)** Función 4400H para obtener información sobre el dispositivo en uso.
**(f)** Función 4408H para determinar si algunos medios son removibles.
(g) Función 440DH, código secundario 60H, para obtener los parámetros del dispositivo.
(h) Función 440DH, código secundario 66H, para obtener la identificación del medio.

18-2. Operaciones que implican directorios:

(a) Función 39H para crear un subdirectorio. Por seguridad, usted debe crearlo en un disco RAM (disco virtual) o en un disco flexible. Utilice cualquier nombre.
**(b)** Función 56H para renombrar el subdirectorio.
(c) Función 3AH para eliminar el subdirectorio.

18-3. Operaciones que implican archivos en disco:

(a) Función 43H para obtener el atributo de un archivo en disco. (Para este ejercicio, utilice una copia de un archivo.)
**(b)** Función 56H para renombrar el archivo.
(c) Función 43H para colocar el atributo de oculto.
**(d)** Función 57H para obtener la fecha y hora del archivo.
**(e)** Función 41H para borrar el archivo.

18-**4.** Escriba un pequeño programa desde el DEBUG, que simplemente ejecute la función 29H del DOS, analizar gramaticalmente el nombre del archivo. Proporcione la especificación del archivo en 81H y el FCB en 5CH; ambos están en el PSP, inmediatamente antes del programa. Introduzca varias especificaciones de archivo, como D:PROGA.DOC, PROGB, PROGC* y C:*.ASM. Después de cada ejecución de la función, verifique los resultados en el desplazamiento 5CH.

# CAPÍTULO 19

# Procesamiento en disco: III—Operaciones del BIOS para disco

OBJETIVO

Examinar los requisitos básicos de programación para utilizar las funciones del BIOS para leer, escribir, formatear y verificar discos.

## INTRODUCCIÓN

En los capítulos 17 y 18 examinamos el uso de los servicios del DOS para procesamiento de disco. También puede codificar de forma directa a nivel del BIOS para procesamiento de disco, aunque el BIOS no facilita un uso automático del directorio o bloqueo y desbloqueo de registros. La operación INT 13H del BIOS para disco, trata información como el tamaño de un sector y maneja el direccionamiento en disco en términos de los números de pista y sectores reales. Las operaciones del BIOS para disco implican restablecer la lectura, escritura, verificación y formateo de la unidad.

La mayoría de las operaciones del BIOS son para expertos desarrolladores de software que están conscientes del peligro potencial por un mal uso. También las versiones del BIOS pueden variar de acuerdo con el procesador utilizado o aun por el modelo de computadora.

Este capítulo introduce las siguientes funciones de la INT 13H del BIOS:

| FUNCIONES PARA DISCO FLEXIBLE | | FUNCIONES PARA DISCO DURO | |
|---|---|---|---|
| 00H | Restablecer sistema de disco flexible | 00H | Restablecer sistema de disco |
| 01H | Leer estado del disco flexible | 01H | Leer estado del disco |

| | |
|---|---|
| 02H | Leer sectores |
| 03H | Escribir sectores |
| 04H | Verificar sectores |
| 05H | Formatear pistas |
| 08H | Obtener parámetros de la unidad |
| 15H | Obtener tipo de disco |
| 16H | Cambiar estado del disco |
| 17H | Establecer tipo de disco |
| 18H | Establecer tipo de medio para formatear |

| | |
|---|---|
| 02H | Leer sectores |
| 03H | Escribir sectores |
| 04H | Verificar sectores |
| 05H | Formatear pistas |
| 08H | Obtener parámetros de la unidad |
| 09H | Inicializar unidad |
| OAH | Leer sector ampliado del búfer |
| OBH | Escribir sector ampliado del búfer |
| OCH | Buscar cilindro |
| ODH | Restauración alterna de disco |
| OEH | Leer búfer del sector |
| OFH | Escribir búfer del sector |
| 15H | Obtener tipo de disco |
| 19H | Estacionar las cabezas del disco |

## BYTE DEL ESTADO DEL BIOS

La mayoría de las funciones de la INT 13H del BIOS ponen en uno o en cero la bandera de acarreo, si hubo un éxito o fracaso y regresan un código de error en el registro AH. El BIOS, en su área de datos, mantiene información acerca de cada dispositivo y su estado. El *byte de estado* mostrado en la figura 19-1 refleja los bits indicadores que se encuentran en el área de datos del BIOS en 40:41H para el área de datos de unidades de discos flexibles y en 40:74H para el área de datos de disco duro. (Véase el capítulo 25 para más detalles.)

Si una operación regresa un error, una acción común del programa es restablecer el disco (función OOH) y reintentar la operación tres veces. Si aún persiste un error, muestra un mensaje y da al usuario una oportunidad para cambiar el disco flexible, si ésa es la solución del problema.

| Código | Estado |
|---|---|
| OOH | No hubo error |
| 01H | Comando incorrecto, no reconocido por el controlador |
| 02H | Marca de dirección en disco no se encontró |
| 03H | Intento de escribir en un disco protegido |
| 04H | Pista/sector no válido |
| 05H | Fallo en la operación de restablecer |
| 06H | Se retiró el disco flexible desde el último acceso |
| 07H | Parámetros de la unidad erróneos |
| 08H | Acceso directo a memoria (DMA) rebasado (información accesada demasiado rápido para ingresar) |
| 09H | Intento de DMA de cruzar una frontera de 64K al leer/escribir |
| 10H | Se encontró una incorrecta CRC en una lectura (verificador de errores indica datos dañados) |
| 20H | Fallo en el controlador (fallo en el hardware) |
| 40H | Fallo en una operación de búsqueda (fallo en el hardware) |
| 80H | Fallo en el dispositivo al responder (disco flexible: compuerta de la unidad abierta o no hay disco; disco duro: se acabó el tiempo) |
| AAH | Unidad no está preparada |
| BBH | Error no definido |
| CCH | Fallo al escribir |

Figura 19-1 Códigos de estado de la INT 13H

## OPERACIONES BÁSICAS DEL BIOS PARA DISCO

Esta sección cubre las funciones básicas para disco de la INT 13H. Cada una necesita un código de función en el registro AH.

### INT 13H, función OOH: Restablecer el sistema de disco flexible

Utilice esta operación después de que la operación anterior ha reportado un error grave. La operación realiza una reinicialización del controlador de disco flexible o del disco duro. Esto es, la siguiente vez que la unidad es accesada, primero se coloca en el cilindro 0. En un disco flexible, establezca el DL al número de la unidad (0 = unidad A, etc.). En disco duro, establezca el DL a un número de 80H o superior (80H = la primera unidad, 81H = la segunda, etc.). Un ejemplo del uso de la función OOH es como sigue:

```
MOV AH.OOH      ,-Petición para restablecer el disco
MOV DL.80H      /Disco duro
INT 13H         /Llama al BIOS
```

Una operación válida pone en cero la bandera de acarreo; un error la pone en uno y regresa un código de estado en el AH. La función ODH es una operación relacionada.

### INT 13H, función 01H: Leer estado del disco

Esta operación le da otra elección para examinar el estado de la mayoría de las más recientes operaciones en disco. (Véase byte de estado en la figura 19-1.) Establezca el DL al código usual (0 = unidad A, etc.) para disco flexible y un número de 80H o más (80H = primer unidad, etc.) para disco duro. Esta operación regresa al AL el código de estado que la última operación en el disco habría regresado al AH. La operación siempre debería ser válida, y pone en cero la bandera de acarreo y regresa su propio código de estado, OOH, en el AH.

### INT 13H, función 02H: Leer sectores

Esta operación lee un número especificado de sectores en la misma pista y de manera directa los envía a la memoria. Inicialice los registros siguientes:

| | |
|---|---|
| **AL** | Número de sectores, hasta el máximo por pista |
| **CH** | Número de pista (los números inician con cero) |
| **CL** | Bits 7-6 número de pista (bits superiores) |
| | Bits 5-0 número de sector inicial (los números inician con uno) |
| **DH** | Número de cabeza (lado) (0 o 1 para disco flexible) |
| **DL** | Número de unidad para disco flexible (0 = **A)** o unidad de disco duro (80H o mayor) |
| ES:BX | Dirección de un búfer de E/S en el área de datos, debe ser suficientemente grande para todos los sectores que sean leídos. (En este caso BX está sujeto al ES.) |

El ejemplo siguiente lee un sector y lo envía a un área llamada INSECT:

```
INSECT   DB      512 DUP(?)     /Área para entrada

         MOV     AH,02H         /Petición de lectura
```

```
MOV     AL, 01          ;Un sector
LEA     BX,INSECT       ,• Búfer de entrada (ES:BX)
MOV     CH, 05          ;Pista 05
MOV     CL, 03          ;Sector 03
MOV     DH, 00          ;Cabeza 00
MOV     DL, 03          .•Unidad 03 (D)
INT     13H             ;Llama al BIOS
```

Al regresar de una operación válida, la bandera de acarreo está en cero y el AL contiene el número de sectores que la operación ha leído realmente. El contenido de los registros DS, BX, CX y DX se preservan. Un error pone en uno la bandera de acarreo y regresa el código del estado en el AH; restablece la unidad (función OOH) y reintenta la operación.

Para la mayoría de las situaciones, usted sólo especifica un sector o todos los sectores para una pista. Inicialice el CH y CL e increméntelos para leer los sectores de forma secuencial. Una vez que el número del sector exceda el máximo para una pista, tiene que restablecerlo a 01 y ya sea incrementar el número de pista en el mismo lado del disco o bien incrementar el número de cabeza para el lado siguiente.

### Prueba si un disco flexible está preparado

Un programa puede emitir una petición para accesar un disco flexible que aún no haya sido insertado. Una práctica común es intentar la operación tres veces antes de mostrar un mensaje al usuario. El ejemplo que sigue utiliza la función 02H de la INT 13H en un intento para leer un sector de datos. Pruebe utilizando DEBUG para ingresar las instrucciones (pero no los comentarios) y pruebe el código con y sin un disco flexible en la unidad A. Para un disco flexible en la unidad, la operación debe leer el contenido del registro de arranque del disco, 512 (200H) bytes leídos, iniciando en la posición DS.200H. El código es:

```
0100    MOV CX, 03          ;Contador para el ciclo
0103    PUSH CX             ,-Guarda el contador
0104    MOV AX,0201         ,• Código de la función y sectores
0107    MOV BX,0200         /Dirección de entrada
010A    MOV CX,0001         ,-Números de pista y sector
010D    MOV DX,0000         /Números de cabeza y unidad
0110    INT 13              /Llama al BIOS
0112    POP CX              /Restaura el contador
0113    JNC 118             /Si no hay error, salir
0115    CLC                 ,-Si hay error,
0116    LOOP 103            / intentar 3 veces
0118    NOP
```

### INT 13H, función 03H: Escribir sectores

Esta operación, la opuesta de la función 02H, escribe un área especificada desde la memoria (512 bytes o un múltiplo de 512) sobre sectores designados formateados. Carga los registros y maneja el procesamiento igual que la función 02H. Una operación válida pone en cero la bandera de acarreo y envía al AL el número de sectores que fueron escritos. El contenido de los registros DS, BX, CX y DX son preservados. Un error pone en uno la bandera de acarreo y regresa un código **de** estado en el AH; restablece la unidad y reintenta la operación.

## USO DEL BIOS PARA LEER SECTORES

Ahora examinemos el programa de la figura 19-2, que usa la INT 13H del BIOS para leer sectores desde el disco y enviarlos a la memoria. Observe que no existe operación para abrir o para un manejador de archivo. Las secciones principales son:

CURADR Contiene la pista y sector iniciales (que el programa incrementa)

END ADR Contiene la pista y sector finales. Una manera de mejorar el programa sería solicitar al usuario las pistas y sectores iniciales y finales.

C10ADDR Calcula cada dirección en el disco en términos de lado, pista y sector. Cuando el número de sector llega a 10, la rutina restablece el sector a 01. Si el lado es 1, el programa incrementa el número de pista; el número de lado es entonces cambiado, de 0 a 1 o de 1 a 0. Este proceso funciona sólo para discos flexibles (porque son de dos lados) que contienen nueve sectores por pista.

FlOREAD Lee un sector e incrementa el número de sector para una operación de lectura válida.

G10DISP Muestra el sector actualmente leído.

Pruebe ejecutar este programa bajo DEBUG. Rastree las instrucciones que inicializan los registros de segmento. Para la operación de entrada, ajuste los sectores de inicio y final a la ubicación de la FAT del disco. (Véase el capítulo 16.) Utilice G (go) para ejecutar el programa y examine la FAT y las entradas del directorio en el área de entrada.

Como una alternativa a DEBUG, su programa convertiría los caracteres ASCII en el área de entrada a sus equivalentes hexadecimales y despliega los números hexadecimales igual que lo hace DEBUG. (Véase también el programa de la figura 15-6.) De esta manera, podría examinar el contenido de cualquier sector —aun los ocultos— y podría permitir al usuario ingresar cambios y escribir los sectores cambiados de regreso en el disco.

Observe que cuando el DOS crea un archivo, inserta registros en grupos disponibles, los cuales pueden no ser contiguos en el disco. Así, no puede esperar que la INT 13H del BIOS lea un archivo secuencialmente aunque pueda accesar las entradas de la FAT para la ubicación de] siguiente grupo.

## OTRAS OPERACIONES DEL BIOS PARA DISCO

A continuación se describen servicios adicionales de la INT 13H del BIOS para disco flexible j disco duro.

```
TITLE     P19BIORD (COM)   Lee sectores del disco vía el BIOS
          .MODEL  SMALL
          .STACK  64
-------------------------------------------------------------
          .DATA
CURADR    DW       0304H              Pista/sector iniciales
ENDADR    DW       0S01H             ;Pista/sector finales
ENDCDE    DB       00                ;Indicador del fin del proc
READMSG   DB        '* Error al leer *$'
RECDIN    DB       512 DÜP(' ')      ;Área de entrada
SIDE      DB       00
-------------------------------------------------------------
          .CODE
BEGIN     PROC     FAR
          MOV      AX,@data           Inicializa
          MOV      DS,AX                registros
          MOV      ES,AX                de segmentos
          MOV      AX,0600H           Petición de recorrido
A20LOOP:
          CALL     Q10SCRN            Limpia la pantalla
          CALL     Q20CURS            Coloca el cursor
          CALL     C10ADDR            Calcula la dirección en disco
          MOV      CX,CURADR
          MOV      DX,ENDADR
          CMP      CX,DX             ,-¿Es el sector final?
          JE       A90               ;  sí, salir
          CALL     F10READ           ,-Lee el registro del disco
          CMP      ENDCDE,00         ;¿Lectura normal?
          JNZ      A90                 no, salir
          CALL     G10DISP           ,-Muestra sector
          JMP      A20LOOP           ;Repite
A90:      MOV      AX.4C00H
          INT      21H               ;Salir al DOS
BEGIN     ENDP
                   Calcula la siguiente dirección del disco:

C10ADDR   PROC     NEAR
          MOV      CX,CURADR         ,-Obtiene pista/sector
          CMP      CL, 10            ;¿Pasó el último sector?
          JNE      C90               ;  no, salir
          MOV      CL, 01            /Establece el sector en 1
          CMP      SIDE,00           ;Si es el lado 0, pasar
          JE       C20
          INC      CH                ;Incrementa la pista
C20:
          XOR      SIDE,01           ,-Cambia de lado
          MOV      CURADR,CX
C90 :     RET
C10ADDR   ENDP
                   Lee sector de disco:

FlOREAD   PROC     NEAR
          MOV      AH,02H            | Petición de lectura
          MOV      AL, 01            | Número de sectores
          LEA      BX,RECDIN         | Dirección del búfer
          MOV      CX, CURADR        | Pista/sector
          MOV      DH,SIDE           | Lado
          MOV      DL,01             | Unidad B
          INT      13H
          CMP      AH, 00            | ¿Lectura normal?
          JZ       F90               |   sí, salir
          MOV      ENDCDE,01         |   no:
          CALL     X10ERR            |   lectura no válida
```

Figura **19-2** Uso de la INT 13H para leer sectores del disco

```
F90:
          INC     CURADR              ;Incrementa sector
          RET
FlOREAD  ENDP
                  Muestra sector:

G10DISP   PROC    NEAR
          MOV     AH,40H              Petición para desplegar
          MOV     BX, 01              Manejador
          MOV     CX,512              Longitud
          LEA     DX,RECDIN
          INT     21H
          RET
G10DISP   ENDP
                  Limpia la pantalla:

Q10SCRN   PROC    NEAR
          MOV     AX,0600H            Petición para recorrer
          MOV     BH,1EH              Designa atributo
          MOV     CX,0000             Pantalla completa
          MOV     DX,184FH
          INT     10H
          RET
Q10SCRN   ENDP
                  Coloca el cursor:

Q2 0CURS  PROC    NEAR
          MOV     AH,02H              ;Petición para
          MOV     BH, 00              ;  colocar el cursor
          MOV     DX,0000
          INT     10H
          RET
Q20CURS   ENDP
                  Muestra el mensaje de error de disco:

XIOERR    PROC    NEAR
          MOV     AH,40H              ;Petición para desplegar
          MOV     BX, 01              ;Manejador
          MOV     CX, 18              ;Longitud del mensaje
          LEA     DX,READMSG
          INT     21H
          RET
X10ERR    ENDP
          END     BEGIN
```

Figura 19-2 (continuación)

INT 13H, función 04H: Verificar sectores

Esta operación sólo verifica que los sectores especificados puedan ser leídos y realiza una verifi cación de redundancia cíclica (CRC). Cuando una operación escribe a un sector, el controlado) del disco calcula y escribe una suma de verificación CRC inmediatamente después del sector, coi base en los bits que están en uno. La función 04H lee el sector, recalcula la suma de verificacióí y la compara con el valor almacenado. Observe que la verificación consiste en recalcular la sum; de verificación en lugar de verificar que los valores de byte en el sector coincidan con los datos di salida en memoria. Puede utilizar esta función después de escribir (función 03H) para asegura mayor confianza en la salida, aunque a un costo de más tiempo de E/S.

Cargue los registros igual que para la función 02H, pero ya que la operación no realiza un verificación real de los datos escritos, no existe necesidad de establecer la dirección en el ES:BX

Al regresar de cargar, la bandera de acarreo se pone en cero y el AL contiene el número de sectores realmente verificados. El contenido de los registros DS, BX, CX y DX se preservan. Un error pone en uno la bandera de acarreo y regresa un código de estado en el AH; restablece la unidad y reintenta la operación.

**INT 13H, función 05H: Formatea pistas**

Las operaciones de lectura/escritura necesitan información sobre el formateo y procesan un sector requerido. Esta operación formatea pistas de acuerdo con uno de cuatro tamaños diferentes. Antes de la ejecución de la operación, utilice la función 17H para establecer el tipo de disco flexible y la función 18H para establecer el tipo de medio. Para formateo de discos flexibles, inicialice estos registros:

| | |
|---|---|
| AL | Número de sectores a formatear |
| CH | Número de pista (los números inician con 0) |
| DH | Número de cabeza (lado) (0 o 1 para disco flexible) |
| DL | Número de unidad, para disco flexible (0 = A) o para disco duro (80H o mayor) |
| ES:BX | Dirección segmento:desplazamiento que apunta a un grupo de campos de dirección para una pista<br>Para cada sector del disco flexible en una pista, debe estar una entrada de cuatro bytes de la forma T/H/S/B, donde<br>Byte 0 T = número de pista (cilindro)<br>1 H = número de cabeza (superficie)<br>2 S = número de sector<br>3 B = bytes por sector (OOH = 128, 01H = 256, 02H = 512H, 03H = 1024) |

Por ejemplo, si usted formatea la pista 03, cabeza 00 y 512 bytes por sector, la primera entrada para la pista es 03000102 hex, seguido por una entrada para cada sector restante.

La operación pone en uno o cero la bandera de acarreo y regresa el código de estado en el AH.

**INT 13H, función 08H: Obtener parámetros de la unidad**

Esta útil función regresa la información acerca de la unidad de disco. Cargue el número de unidad en **el** DL (0 = A, 1 = B para disco flexible y 80H o mayor para disco duro). Una operación exitosa regresa lo siguiente:

| | |
|---|---|
| BL | Tipo de disco flexible (01H = 360K, 02H = 1.2M, 03H = 720K, 04H = 1.44M) |
| CH | Número superior de cilindro/pista |
| CL | Bits 0-5 = número superior de sector<br>Bits 6-7 = dos bits de orden alto del número de cilindro |
| DH | Número superior de cabeza |
| DL | Número de unidades conectadas al controlador |
| ES:DI | Para discos flexibles, segmento desplazamiento de una tabla de 11 bytes de parámetros de unidad de disco flexible.<br>Dos campos relevantes son:<br>Desplazamiento 3—bytes por sector (OOH = 128, 01H = 256, 02H = 512H, 03H = 1024)<br>Desplazamiento 4—sectores por pista |

Puede utilizar el comando D ES desplazamiento de DEBUG (el desplazamiento en el DI) para desplegar los números. La operación pone en uno o cero la bandera de acarreo y regresa el código de estado en el AH.

INT **13H, función** 09H: **Inicializar la unidad**

El BIOS realiza esta función cuando arranca su computadora, de acuerdo con una tabla de disco duro en BIOS. El DL contiene el número de unidad (80H o mayor). La operación pone en uno o cero la bandera de acarreo y regresa el estado en el AH. Las INT 41H e INT 46H del BIOS son operaciones relacionadas.

**INT 13H, función OAH: Leer búfer ampliado del sector**

El búfer del sector en discos duros incluye los 512 bytes de datos más 4 bytes para un código de corrección de error (ECC) utilizado para verificación de error y corrección de información. Esta función puede leer todo el búfer del sector al igual que sólo una parte de la información. Para leer un búfer ampliado, cargue estos registros:

| | |
|---|---|
| AL | Número de sectores (hasta el máximo para la unidad) |
| BX | Segmento desplazamiento del búfer de entrada (como ES:BX) |
| CH | Número de cilindro/pista |
| CL | Bits 0-5 = número superior de sector |
| | Bits 6-7 = dos bits de orden alto del número del cilindro |
| DH | Número de cabeza (lado) |
| DL | Número de unidad (80H o mayor) |

Una operación exitosa regresa al AL el número de sectores transferidos. La operación pone en uno o cero la bandera de acarreo y regresa un código de estado en el AH.

**INT 13H, función OBH: Escribe búfer ampliado del sector**

Esta función es similar a la función OAH, excepto que, en lugar de leer el búfer del sector, lo escribe (incluyendo el código ECC) en el disco.

**INT 13H, función OCH: Busca cilindro**

Esta función coloca la cabeza de lectura/escritura de un disco duro en un cilindro específico (pista), pero no transfiere información. Para buscar un cilindro, cargue estos registros:

| | |
|---|---|
| CH | Número de cilindro/pista |
| CL | Bits 0-5 = número de sector |
| | Bits 6-7 = dos bits de orden alto del número de cilindro |
| DH | Número de cabeza (lado) |
| DL | Unidad (80H o mayor) |

La operación pone en uno o cero la bandera de acarreo y regresa un código de estado en el AH.

**INT ljH, función ODH: Restauración alterna de disco**

Esta operación es similar a la función OOH, excepto que está restringida a discos duros. Cargue la unidad (80H o mayor) en el DL. El brazo de acceso de lectura/escritura se reposiciona en el

cilindro 0. La operación pone en uno o cero la bandera de acarreo y regresa un código de estado en el AH.

**INT 13H, función OEH: Leer búfer del sector**

Esta operación es similar a la función OAH, salvo que lee parte de los 512 bytes del sector y no los bytes del ECC.

**INT 13H, función OFH: Escribir búfer del sector**

Esta operación es similar a la función OBH, salvo que escribe parte de los 512 bytes del sector ECC.

**INT 13H, función 10H: Probar si está preparada la unidad; 11H: Recalibrar la unidad de disco duro; 12H: Diagnóstico de ROM; 13H: Diagnóstico de la unidad, y 14H: Diagnóstico del controlador**

Estas funciones realizan un diagnóstico interno y reportan información específica para el BIOS y para programas avanzados de utilerías. Estas operaciones ponen en uno o cero la bandera de acarreo y regresan un código de estado en el AH.

**INT 13H, función 15H: Obtiene el tipo de disco**

Esta función regresa la información acerca de la unidad de disco. Cargue el DL con la unidad (0 = A, etc. para disco flexible o bien 80H o mayor para disco duro). Una operación válida regresa uno de los códigos siguientes en el AH:

| | |
|---|---|
| OOH | No está presente unidad/disco |
| 01H | Unidad de disco flexible no es sensible a cambio de disco |
| 02H | Unidad de disco flexible es sensible a cambio de disco |
| 03H | Unidad de disco duro |

Por el código de regreso 03 en el AH, la pareja CX:DX contiene el número total de sectores de disco en la unidad. La operación pone en uno o cero la bandera de acarreo y regresa un código de estado en el AH.

**INT 13H, función 16H: Cambia el estado del disco flexible**

Esta función verifica si hay un cambio de disco flexible para sistemas que pueden ser sensibles a cambios. Cargue el DL con el número de unidad (0 = A, etc.). La operación regresa uno de los códigos siguientes en el AH:

| | |
|---|---|
| OOH | No se ha cambiado de disco flexible (bandera de acarreo = 0) |
| 01H | Parámetro no válido de disco flexible (bandera de acarreo = 1) |
| 06H | Disco flexible cambiado (bandera de acarreo = 1) |
| 80H | Unidad de disco flexible no está preparada (bandera de acarreo = 1) |

Los códigos de estado 01H y 80H son errores que ponen en uno la bandera de acarreo, mientras que el 06H es un estado válido que también pone en uno la bandera de acarreo. Ésta es una fuente potencial de confusión.

**INT 13H, función 17H: Establece el tipo de medio**

Esta operación configura la combinación de unidad y disco flexible. Utilice la función 17H junto con la función 05H para formateo de disco. Cargue el número de unidad (0 = A, etc.) en el DL y el tipo de disco flexible en el AL. Los tipos de disco flexible son:

01H Disco flexible de 360K en unidad de 360K
02H Disco flexible de 360K en unidad de 1.2M
03H Disco flexible de 1.2M en unidad de 1.2M
04H Disco flexible de 720K en unidad de 720K

La operación pone en uno o cero la bandera de acarreo y regresa un código de estado en el AH.

**INT 13H, función 18H: Establece tipo de medio para formatear**

Utilice esta operación inmediatamente antes de ejecutar la función 05H. Para establecer el tipo de medio, cargue estos registros:

CH Número de pistas (los ocho bits de orden bajo)
CL Número de pistas (dos bits de orden alto 7-6), sectores por pista (bits 5-0)
DL Unidad (0 = A, etc.)

Una operación válida regresa en el ES:DI un apuntador a una tabla, de 11 bytes, de parámetros de disco flexible. (Véase la función 08H.) La operación pone en uno o cero la bandera de acarreo y regresa un código de estado en el AH.

**INT 13H, función 19H: Estacionar las cabezas del disco**

Esta operación necesita el número de la unidad en el DL (80H y mayor para disco duro). La operación pone en uno o cero la bandera de acarreo y regresa un código de estado en el AH.

## PUNTOS CLAVE

- La INT 13H del BIOS proporciona acceso directo a las pistas y sectores.
- La INT 13H del BIOS no proporciona un manejo automático de directorio, operaciones con fin de archivo ni bloqueo y desbloqueo de registros.
- La operación para verificar sector realiza una comprobación elemental de datos escritos con un costo en el tiempo de procesamiento.
- El programa debe examinar el byte de estado después de cada operación para disco del BIOS.

## PREGUNTAS

**19-1.** ¿Cuáles son las dos principales desventajas de utilizar la INT 13H del BIOS? Esto es, ¿por qué por lo común es preferible el uso de las interrupciones del DOS?

**19-2.** ¿Bajo qué circunstancias un programador usaría la INT 13H del BIOS?

**19-3.** La mayoría de las operaciones de la INT 13H regresan un código de estado, (a) ¿En dónde es regresado este código? (b) ¿Qué significa el código OOH? (c) ¿Qué significa el código 03H?

**19-4.** ¿Cuál es el procedimiento común para un error regresado por la INT 13H?

**19-5.** Codifique las instrucciones para restablecer el controlador del disco flexible.

**19-6.** Codifique las instrucciones para leer el estado del disco flexible.

**19-7.** Usando la dirección de memoria INDSK, unidad A, cabeza 0, pista 6 y sector 3, codifique las instrucciones para la INT 13H del BIOS a fin de leer un sector.

**19-8.** Usando la dirección de memoria OUTDSK, unidad B, cabeza 0, pista 8 y sector 1, codifique instrucciones para la INT 13H del BIOS a fin de escribir tres sectores.

**19-9.** Después de la escritura de la pregunta 19-8, ¿cómo verificaría un intento de escribir en un disco protegido?

**19-10.** Con base en la pregunta 19-8, codifique las instrucciones para verificar la operación de escritura.

## CAPÍTULO 20

# Impresión

OBJETIVO

**Describir los requisitos para imprimir por medio de las interrupciones del DOS y del BIOS.**

## INTRODUCCIÓN

Comparada con el manejo de la pantalla y de disco, la impresión parece ser un proceso relativamente sencillo. Sólo se emplean unas cuantas operaciones, todas ellas por medio de la INT 21H del DOS o por medio de la INT 17H del BIOS. Los comandos especiales para la impresora incluyen Avance de página (FF), Avance de línea (LF) y Retorno de carro (CR).

La impresora debe entender una señal enviada desde el procesador, por ejemplo, para saltar a una página nueva, avanzar hacia abajo una línea o tabular. El procesador también debe entender una señal enviada desde una impresora que indica que está ocupada o sin papel. Desafortunadamente, muchos tipos de impresoras responden de manera diferente a señales enviadas desde un procesador y una de las tareas más difíciles para los especialistas en software es realizar la interfaz entre sus programas y tales impresoras.

Este capítulo introduce las siguientes operaciones de interrupción:

| FUNCIONES DE LA INT 21H DEL DOS | | FUNCIONES DE LA INT 17H DEL BIOS | |
|---|---|---|---|
| 40H | Imprime caracteres | 00H | Imprime un carácter |
| 05H | Imprime un carácter | 01H | Inicializa puerto |
| | | 02H | Obtiene estado del puerto de la impresora |

## CARACTERES COMUNES DE CONTROL DE LA IMPRESORA

Los caracteres estándar que controlan la impresión en todas las impresoras comunes a la PC incluyen los siguientes:

| Decimal | Hex | Función |
|---|---|---|
| 09 | 09H | Tabulación (tab) horizontal |
| 10 | OAH | Avanza una línea |
| 12 | OCH | Avanza una página (avanza a la página siguiente) |
| 13 | ODH | Retorno de carro (regresa al margen izquierdo) |

**Tabulación horizontal.** El carácter de control Tabulación horizontal (09H) provoca que la impresora coloque el carácter actual en la siguiente marca de tabulación (por lo común, si todas las marcas están puestas, cada ocho posiciones). El comando funciona sólo en impresoras que tienen la característica y sólo cuando las tabulaciones de la impresora están configuradas. Puede imprimir espacios en blanco para evitar una incapacidad de impresora en las tabulaciones.

**Avance de línea(LF).** El carácter de control de Avance de línea (OAH) avanza una sola línea, y dos LF sucesivos dan un espacio doble.

**Avance de página (FF).** Cuando enciende su impresora, la posición en donde se encuentre el papel determina la posición inicial para la parte superior de una página. La longitud de una hoja, por omisión, es de 11 pulgadas, que permiten usar 66 líneas a 6 líneas por pulgada. Ni el procesador ni la impresora verifican de forma automática si se llegó a la parte inferior de la página. En formas continuas, si su programa continúa imprimiendo hacia abajo de una página, en algún momento imprime sobre la perforación de la parte inferior y sobre la parte superior de la siguiente página. Para controlar la paginación, cuente las líneas conforme se imprimen y al llegar al máximo por página (como 60), emita un comando Avance de página (OCH) y después reinicialice el conteo de líneas a 0 o a 1.

Al final de la impresión, envíe un comando LF o FF para forzar a que la impresora imprima la última línea que aún está en su búfer. Emitir un Avance de página al final de la impresión, también facilita el corte de la última página.

**Retorno de carro (CR).** El carácter de Retorno de carro (ODH) vuelve a colocar la impresora en su margen izquierdo y, por lo común, los programas lo acompañan con un LF. En el teclado, este carácter es conocido como Enter, Return o Intro.

## DOS 21H, FUNCIÓN 40H: IMPRIMIR CARACTERES

Ya hemos utilizado manejadores de archivo en los capítulos acerca de manejo de pantalla y procesamiento en disco. Para imprimir con la función 40H de la INT 21H del DOS, cargue estos registros:

AH Función 40H
BX Manejador de archivo 04
CX Número de caracteres que se van a imprimir
DX Dirección del texto

El ejemplo siguiente imprime 25 caracteres desde un elemento de datos llamado HEADING, empezando en el margen de la izquierda. Los caracteres de Retorno de carro (ODH) y Avance de línea (OAH) provocan inmediatamente después del texto en HEADING que la impresora reubique el carro y avance una línea:

```
DB    'Industrial Bicycle MFRS', ODH, OAH


MOV   AH,40H        ;Petición de salida
MOV   BX, 04        ,-Manejador 04 para la impresora
MOV   CX, 25        ,-Envía 25 caracteres
LEA   DX,HEADING    ;Dirección del área de impresión
INT   21H           ;Llama al DOS
```

Una operación exitosa imprime el texto, pone en cero la bandera de acarreo y regresa el número de caracteres impresos en el AX. Una operación no válida pone en uno la bandera de acarreo y regresa en el AX el código de error 05 (acceso denegado) o 06 (manejador no válido). Un marcador de fin de archivo (Ctrl-Z o OAH) en los datos también provoca que la operación termine.

## IMPRESIÓN CON ENCABEZADOS DE PÁGINA

El programa de la figura 20-1 se parece al de la figura 9-2 en que acepta de un usuario nombres por medio del teclado y los despliega hacia abajo en la pantalla. Sin embargo, el primero dirige los nombres a la impresora en lugar de almacenarlos en disco. Cada página impresa contiene un encabezado seguido por un espacio doble y los nombres ingresados en el formato siguiente:

```
List of Employee Ñames     Page 01

Clancy  Alderson

Janet  Brown

David  Christie
```

El programa cuenta cada línea impresa y, cerca de la parte inferior de la página, salta la forma al inicio de la página siguiente. Los procedimientos principales son los siguientes:

| | |
|---|---|
| D10INPT | Solicita y acepta un nombre desde el teclado. |
| E10PRNT | Si está al final de una página (60 líneas), llama a M10PAGE; imprime el nombre (su longitud está basada en la longitud real en la lista de parámetros de entrada del teclado). |
| M10PAGE | Avanza a una página nueva; imprime el encabezado; restablece el contador de líneas y agrega uno al contador de páginas. |
| P10OUT | Rutina común, maneja la petición real para imprimir. |

```
TITLE     P20PRTNM (EXE)  Acepta nombres y los imprime
          .MODEL  SMALL
          .STACK  64

          .DATA
NAMEPAR   LABEL   BYTE              |Lista de parámetros del teclado
MAXNLEN   DB      20                |  Longitud máxima del nombre
NAMELEN   DB      ?                 |  longitud ingresada realmente
NAMEFLD   DB      20 DUP(' ')       |  nombre ingresado
                                    |Encabezado de línea:
HEADG     DB      'List of Employee Ñames    Page '
PAGECTR   DB      •OÍ", OAH, OAH

FFEED     DB      OCH               Avanza página
LFEED     DB      OAH               /Avanza línea
LINECTR   DB      01
PROMPT    DB      'Ñame? '

          .CODE
BEGIN     PROC    FAR
          MOV     AX,@data          /Inicializa
          MOV     DS,AX               registros de
          MOV     ES,AX               segmentos
          CALL    Q10CLR            /Limpia la pantalla
          CALL    MÍOPAGE           /Encabezado de página
A20LOOP:
          MOV     DX,0000           /Coloca el cursor en 00,00
          CALL    Q20CURS
          CALL    DIOINPT           /Proporciona entrada de nombre
          CALL    Q10CLR
          CMP     NAMELEN,00        ¿No se ingresó un nombre?
          JE      A30                 no, salir
          CALL    E10PRNT             sí, preparar la impresión
          JMP     A20LOOP
A30:
          MOV     CX,01             |Fin del procesamiento:
          LEA     DX,FFEED          |  un carácter
          CALL    P10OUT            |  para avance de página.
          MOV     AX,4C00H          |  salir al DOS
          INT     21H
BEGIN     ENDP
          Acepta entrada de nombre:

D30INPT   PROC    NEAR
          MOV     AH,40H            Petición para desplegar
          MOV     BX,01
          MOV     CX,05               5 caracteres
          LEA     DX,PROMPT           petición de mensaje
          INT     21H
          MOV     AH,0AH            Petición de entrada
          LEA     DX,NAMEPAR          desde el teclado
          INT     21H
          RET
DIOINPT   ENDP
          Prepara para imprimir:

E10PRNT   PROC    NEAR
          CMP     LINECTR,60        ¿Fin de página?
          JB      E20                 no, pasar
          CALL    MÍOPAGE             sí, imprimir encabezado
E20:
          MOV     CH,00
          MOV     CL,NAMELEN        Asignar número de caracteres
          LEA     DX,NAMEFLD        Asignar dirección del nombre
```

Figura 20-1 Impresión con encabezados de página

```
            CALL    P10OUT              Imprime nombre
            MOV     CX, 01              Avanza
            LEA     DX,LFEED              una línea
            CALL    P10OUT
            INC     LINECTR           ,-Incrementa en uno el conteo de línea
            RET
E10PRNT     ENDP
                    Rutina de encabezado de página:

MI OPAGE    PROC    NEAR
            CMP     WORD PTR PAGECTR,3130H   /¿Primer página?
            JE      M30                       si, pasar
            MOV     CX, 01
            LEA     DX,FFEED                  no,
            CALL    P10OUT                    avanzar página,
            MOV     LINECTR,03                reinicializar conteo de líneas
M30:
            MOV     CX,3G               /Longitud del encabezado
            LEA     DX,HEADG            /Dirección del encabezado

            CALL    P10OUT
            INC     PAGECTR+1           /Incrementa en uno el conteo de páginas
            CMP     PAGECTR+1, 3AH      /¿Núm. de página = 3AH?
            JNE     M50                 /  no, pasar
            MOV     PAGECTR+1,3 OH      /  sí, cambiar a ASCII
            INC     PAGECTR
M50:        RET
MIOPAGE     ENDP
                    Rutina de impresión:

P10OUT      PROC    NEAR                CX y DX designan entrada
            MOV     AH,40H              Petición de impresión
            MOV     BX, 04              Manej ador
            INT     21H
            RET
P10OUT      ENDP
                    Limpia pantalla:

Q10CLR      PROC    NEAR
            MOV     AX,0600H            Petición para recorrer
            MOV     BH,6OH              Atributo
            MOV     CX,0000             Desde 00,00
            MOV     DX,184FH              hasta 24,79
            INT     10H
            RET
Q10CLR      ENDP
                    Coloca el cursor en renglón/columna

Q2 0CURS    PROC    NEAR                DX designa entrada
            MOV     AH, 0:2H            Petición de colocar el cursor
            MOV     BH, 00              Página número 0
            INT     10H
            RET
Q2 0CURS    ENDP
            END ,   BEGIN
```

Figura 20-1 (continuación)

Al inicio de la ejecución es necesario imprimir un encabezado, pero no saltar a una página! nueva. Para este fin, si PAGECTR contiene 01, su valor inicial, entonces M10PAGE pasa por¡ alto el Avance de página. PAGECTR está definido como i

```
PAGECTR DB '01'
```

que genera un número ASCII, 3031H. La rutina en M10PAGE incrementa en uno PAGECTR de manera que se convierte en forma progresiva en 3032, 3033, etc. El número es válido hasta 3039 y después se convierte en 303A, que se imprimiría como un cero y dos puntos. Si el byte de más a la derecha de PAGECTR contiene 3AH, la rutina lo cambia a 30H y le suma uno al byte de más a la izquierda, de modo que 303AH se convierte en 3130H, o 10 decimal.

Colocar una prueba para el final de la página antes (en lugar de después) de imprimir un nombre, asegura que la última página tiene al menos un nombre debajo del título.

## IMPRESIÓN DE ARCHIVOS ASCII Y MANEJO DE TABULADORES

Un procedimiento común realizado, por ejemplo, para el adaptador de video, es reemplazar un carácter de tabulador (09H) con blancos hasta la siguiente posición divisible entre 8. Por tanto los altos de tabulador podrían estar en las posiciones 8, 16, 24, etc., de manera que todas las posiciones 0 y 7 se tabulen a 8, aquellas entre 8 y 15 se tabulen a la posición 16, y así sucesivamente. Sin embargo, algunas impresoras ignoran los caracteres del tabulador. Por ejemplo, el PRINT del DOS imprime archivos ASCII (tal como programas fuente de ensamblador), tiene que verificar cada carácter que envía a la impresora. Si el carácter es un Tab, el programa inserta espacios en blanco hasta la siguiente posición de tabulador.

El programa de la figura 20-2 solicita al usuario introducir un nombre de un archivo e imprime el contenido del archivo. El programa es similar al de la figura 17-3 que despliega registros, pero va un paso más allá al reemplazar las marcas de tabulación con espacios en blanco para la impresora. Encontrará la lógica en G10XFER después de la etiqueta G60. A continuación están tres ejemplos de altos de tabulador para imprimir las posiciones 1, 9 y 21 y la lógica para la determinación de la siguiente posición del tabulador:

| Posición actual de impresión: | 1 | 9 | 21 |
|---|---|---|---|
| Número binario: | 00000001 | 00001001 | 00010101 |
| Borrar los 3 bits de la derecha: | 00000000 | 00001000 | 00010000 |
| Sumar 8: | 00001000 | 00010000 | 00011000 |
| Nueva posición del tabulador: | 8 | 16 | 24 |

El programa está organizado como sigue:

| | |
|---|---|
| C10PRMP | Solicita al usuario que introduzca un nombre de archivo. Para indicar que el usuario ha terminado, se presiona sólo Enter. |
| E10OPEN | Abre para entrada el archivo en disco solicitado. |
| G10XFER | Verifica que la entrada de datos par a fin de sector, fin de archivo, fin de área de despliegue, LF y tabulador. Básicamente, envía caracteres regulares al área de despliegue. |
| P10PRNT | Imprime la línea de despliegue y la limpia con blancos. |
| R10READ | Lee un sector del archivo. |

Los caracteres Retorno de carro, Avance de línea y Avance de página deben funcionar en todas las impresoras. Usted podría modificar el programa anterior para contar las líneas impresas y forzar a un avance de página cuando se está cerca de la parte inferior de la página, en la línea 60 o algo así. (Algunos usuarios prefieren utilizar un programa editor para incrustar caracteres de Avance de página en sus archivos ASCII, en la posición exacta en donde quieren el cambio de página,

```
TITLE     P20PRTAS (EXE)  Lee e imprime registros de disco
          .MODEL  SMALL
          .STACK  64
1
          .DATA
PATHPAR   LABEL   BYTE              Lista de parámetros para
MAXLEN    DB      32                  entrada de
NAMELEN   DB      ?                   nombre de archivo
FILENAM   DB      32 DUP(' ')

COUNT     DW      00
DISAREA   DB      120 DUP(' ')      Área de despliegue
ENDCDE    DW      00                Indicador de fin del proceso
FFEED     DB      OCH
HANDLE    DW      0
OPENMSG   DB      '*** open error ***'
PROMPT    DB      'Ñame of file? '
SECTOR    DB      512 DUP(' ')      ,-Área de entrada para el archivo

          .CODE
BEGIN     PROC    FAR               Procedimiento principal
          MOV     AX.Odata          Inicializa
          MOV     DS, AX              registros de
          MOV     ES, AX              segmentos
          CALL    Q10SCR            Limpia la pantalla
          CALL    Q2 0CURS          Coloca el cursor
AI0LOOP:
          MOV     ENDCDE,00         Inicializa
          CALL    C10PRMP           Petición de nombre de archivo
          CMP     NAMELEN,00        ¿Alguna petición?
          JE      A90                 no, salir
          CALL    E10OPEN           Abre archivo, obtiene manejador
          CMP     ENDCDE,00         ¿Apertura válida?
          JNE     A80                 no, pedir otra vez
          CALL    R10READ           Lee primer sector del disco
          CMP     ENDCDE,00         ¿Fin del archivo, no hay datos?
          JE      A80                 sí, pedir el siguiente
          CALL    G10XFER           Imprime/lee
A80:
          JMP     A10LOOP           Repite
A90:      MOV     AX,4C00H          /Salir al DOS
          INT     21H
BEGIN     ENDP
!                 Petición del nombre de archivo:

C10PRMP   PROC    NEAR
          MOV     AH,40H            ,-Pide el nombre del archivo
          MOV     BX, 01
          MOV     CX, 13
          LEA     DX, PROMPT
          INT     21H
          MOV     AH, OAH           ;Acepta nombre del archivo
          LEA     DX,PATHPAR
          INT     21H
          MOV     BL,NAMELEN        ,•Inserta
          MOV     BH, 00            ;  cero al final del
          MOV     FILENAM[BX] ,0    ;  nombre del archivo
C90:      RET
C10PRMP   ENDP
;                 Abre archivo en disco:

ÉlOOPEN   PROC    NEAR
          MOV     AH,3DH            ,-Petición para abrir
          MOV     AL, 00            ,-Sólo lectura
```

Figura **20-2** Impresión de un archivo ASCII

```
            LEA     DX, FILENAM
            INT     21H
            JNC     E20             /Prueba bandera de acarreo,
            CALL    X10ERR          /  si está en uno, error
            JMP     E90
E20:
            MOV     HANDLE, AX      /Guarda el manejador
            MOV     AX,2020H
            MOV     CX,256          /Limpia el área
            LEA     DI,SECTOR       /  del sector
            REP  STOSW
E90:        RET
E10OPEN     ENDP
                    Transfiere datos a la línea de impresión:

G10XFER     PROC    NEAR
            CLD                     /Designa de izquierda a derecha
            LEA     SI,SECTOR       /Inicializa
G20:
            LEA     DI,DISAREA
            MOV     COUNT,00
G30:
            LEA     DX.SECTOR+512
            CMP     SI,DX           ¿Fin del sector?
            JNE     G40               no, pasa
            CALL    RlOREAD           sí, lee el siguiente
            CMP     ENDCDE,00        ¿Fin del archivo?
            JE      G80               sí, salir
            LEA     SI,SECTOR
G40:
            MOV     BX,COUNT
            CMP     BX, 80          ¿Está al final del área de despliegue?
            JB      G50             /  no, pasar
            MOV     [DI+BX],0D0AH   /  sí, colocar CR/LF
            CALL    P10PRNT
            LEA     DI, DISAREA     /Reinicializar
            MOV     COUNT,00
G50:
            LODSB                   / [SI] a AL, INC SI
            MOV     BX,COUNT
            MOV     [DI+BX],AL      /Carácter a línea de impresión
            INC     BX
            CMP     AL, 1AH         ¿Fin del archivo?
            JE      G80             /  sí, salir
            CMP     AL, OAH         /¿Avanza línea?
            JNE     G60
            CALL    P10PRNT         /Llamada para imprimir
            JMP     G20
GSO:
            CMP     AL,09H          /¿Carácter tabulador?
            JNE     G70
            DEC     BX              /  sí, reinicializa BX
            MOV     BYTE PTR [DI+BX] 20H  /Limpia tabulador a blanco
            AND     BX,0FFF8H       /Limpia los 3 bits de la derecha
            ADD     BX, 08          /  y suma 8
G70:
            MOV     COUNT,BX
            JMP     G30
G80:
            MOV     BX,COUNT        /Fin del archivo
            MOV     BYTE PTR [DI+BX],OCH   /Avanza página
            CALL    P10PRNT         /Imprime última línea
G90:        RET
G10XFER     ENDP
```

Figura 20-2 (continuación)

```
                    Imprime linea:

P10PRNT     PROC    NEAR
            MOV     AH,40H          ;Petición para imprimir
            MOV     BX,04
            MOV     CX,COUNT        ;Longitud
            INC     CX
            LEA     DX,DISAREA
            INT     21H
            MOV     AX,2020H        ;Limpia línea de despliegue
            MOV     CX,60
            LEA     DI,DISAREA
            REP  STOSW
            RET
P10PRNT     ENDP
                    Lee sector de disco:

RlOREAD     PROC    NEAR
            MOV     AH,3FH           Petición para leer
            MOV     BX,HANDLE        Dispositivo
            MOV     CX,512           Longitud
            LEA     DX,SECTOR        Búfer
            INT     21H
            MOV     ENDCDE, AX
            RET
RlOREAD     ENDP
                    Recorre la pantalla:

Q10SCR      PROC    NEAR
            MOV     AX,0S00H        ;Petición para recorrido
            MOV     BH,1EH          ;Designa atributo
            MOV     CX,0000
            MOV     DX,184FH
            INT     10H
            RET
Q10SCR      ENDP
                    Coloca cursor:

Q2 0CURS    PROC    NEAR
            MOV     AH,02H          ;Petición para
            MOV     BH, 00          ;  colocar el cursor
            MOV     DX, 00
            INT     10H
            RET
Q20CURS     ENDP
                    Despliega mensaje de error de disco:

X10ERR      PROC    NEAR
            MOV     AH,40H           Petición para despliegue
            MOV     BX, 01           Manej ador
            MOV     CX, 18           Longitud
            LEA     DX,OPENMSG
    ?10t>6  INT     21H              Mensaje de error
            MOV     ENDCDE, 01
            RET                     ;Indicador de error
X10ERR      ENDP
            END     BEGIN
```

Figura 20-2 (continuación)

como al final de un procedimiento. El método usual es mantener oprimida la tecla Alt y presionar los números del teclado numérico; por ejemplo, 012 para FF.)

Puede corregir el programa para la función 05H del DOS para enviar-cada carácter directamente a la impresora, eliminando la definición y uso del área de despliegue.

## DOS 21H, FUNCIÓN 05H: IMPRIMIR UN CARÁCTER

La función original 05H del DOS proporciona facilidades para imprimir. Cargue la función 05H en el registro AH, el carácter que quiere imprimir en el DL y emita la INT 21H como sigue:

```
MOV  AH,05H          Petición para imprimir un carácter
MOV DL,  char        Carácter que se va a imprimir
INT  21H             Llama al DOS
```

Estas instrucciones son adecuadas para enviar un solo carácter a la impresora. Sin embargo, por lo regular, la impresión implica una línea de texto completa o parte de ella y requiere pasar por una línea formateada en el área de datos.

El ejemplo siguiente ilustra la impresión de una línea completa. Primero inicializa la dirección de HEADING en el registro SI y pone en el CX la longitud de HEADING. Después, el ciclo en P20 extrae cada carácter de forma sucesiva de HEADING y lo envía a la impresora. Ya que el primer carácter en HEADING es un FF y los últimos dos caracteres son LF, el encabezado se imprime en la parte superior de una página nueva y es seguido por un espacio doble. El código es como sigue:

```
HEADING DB   OCH,'Industrial Bycicle Mfrs' ,ODH, OAH,OAH


        MOV  CX, 2 7          rInicializa la longitud y
        LEA  SI,HEADING       ; la dirección del encabezado
P20 :
        MOV  AH,05H           Petición para imprimir
        MOV  DL, [SI]          un carácter del encabezado
        INT  21H              Llama al DOS
        INC  SI               Siguiente carácter del encabezado
        LOOP P2 0             Repetir 27 veces
```

Si la impresora no está encendida, el DOS regresa el mensaje "No hay papel" de forma repetida. Si enciende la impresora, el programa empieza a imprimir correctamente. También puede utilizar Ctrl+Break para cancelar la ejecución de la operación de imprimir.

## CARACTERES ESPECIALES DE CONTROL PARA LA IMPRESORA

Ya hemos examinado el uso de varios caracteres básicos de control para la impresora, tales como Avance de página y Retorno de carro. Otros comandos adecuados para la mayoría de las impresoras comunes son los siguientes:

| DECIMAL | HEX | ACCIÓN |
|---|---|---|
| 08 | 08 | Retroceso |
| 11 | 0B | Tabulador vertical |
| 15 | OF | Activa modo condensado |

| | | |
|---|---|---|
| 14 | OE | Activa modo expandido |
| 18 | 12 | Desactiva modo condensado |
| 20 | 14 | Desactiva modo expandido |

Algunos comandos necesitan un carácter (1BH) Esc (escape) precediéndolos. Algunos de estos comandos, según la impresora, son:

| | |
|---|---|
| IB 30 | Fija el interlineado a 8 líneas por pulgada |
| IB 32 | Fija el interlineado a 6 líneas por pulgada |
| IB 45 | Activa el modo de impresión enfatizado |
| IB 46 | Desactiva el modo de impresión enfatizado |

Puede enviar caracteres de control a la impresora de dos maneras diferentes:

1. Defina comandos en el área de datos. Lo siguiente fija el modo condensado, 8 líneas por pulgada, imprime un título y provoca un Retorno de carro y un Avance de línea:

```
HEADING DB OFH, 1BH, 30H, 'Título        ODH, OAH
```

**2.** Utilice las instrucciones inmediatas para fijar el modo condensado:

```
MOV    AH,05H
MOV    DL,OFH          Petición de modo condensado
INT    21H             Llama al DOS
```

Todos los caracteres subsecuentes se imprimen en modo condensado hasta que el programa envíe un comando que establezca otro modo.

Otros comandos no necesariamente funcionan en todos los modelos de impresoras. Verifique en su manual los comandos específicos de la impresora.

## FUNCIONES DE LA INT 17H DEL BIOS PARA IMPRESIÓN

La INT 17H proporciona facilidades para imprimir en el nivel del BIOS. Los puertos de impresión válidos para la INT 17H son 0 (por omisión), 1 y 2, para LPT1, LPT2 y LPT3, respectivamente. La INT 17H proporciona tres funciones diferentes, como se especifica en el registro AH:

1. Primero emita la función 02H para determinar el estado de la impresora, vía un número de puerto seleccionado. Incluya esta prueba de estado antes de cada intento de imprimir. Si la impresora está disponible, entonces:
**2.** Emita la función 01H para inicializar el puerto de impresión, y:
**3.** Emita las operaciones de la función OOH para enviar caracteres a la impresora.

Las operaciones regresan el estado de la impresora al AH, con uno o más bits puestos en 1:

| BIT | CAUSA |
|---|---|
| 0 | Se acabó el tiempo |
| 3 | Error de entrada/salida |
| 4 | Seleccionada |

5 No hay papel
6 Reconocimiento desde la impresora
7 No está ocupada

Si la impresora ya está encendida y preparada, la operación regresa 90H (10010000 binario): la impresora no está ocupada, pero está seleccionada, una condición válida. Errores en la impresora son el bit 5 (no hay papel) y el bit 3 (error de salida). Si la impresora no está encendida, la operación regresa BOH, o 10110000 binario, indicando "No hay papel".

### INT 17H, función OOH: Imprimir un carácter

Esta operación hace que se imprima un carácter y permite impresoras en los puertos 0, 1 o 2. Cargue el carácter en el AL y el número del puerto de la impresora en el DX:

```
MOV  AH.0 0H          Petición para imprimir
MOV  AL, char         Carácter que se va a imprimir
MOV  DX, 0 0          Selecciona el puerto 0 para la impresora
INT  17H              Llama al BIOS
```

La operación regresa el estado al registro AH. La práctica recomendada es utilizar primero la función 02H para examinar el estado de la impresora.

### INT 17H, función 01H: Inicializa el puerto de la impresora

Esta operación selecciona un puerto, restablece la impresora y la inicializa para datos:

```
MOV  AH,01H        ;Petición para inicializar el puerto
MOV  DX,00         /Selecciona el puerto 0 para la impresora
INT  17H           /Llama al BIOS
```

Ya que la operación envía un carácter de Avance de página, puede usarla para fijar la impresora en la posición superior de la página, aunque algunas impresoras lo hacen de manera automática cuando se encienden. La operación regresa un código de estado en el AH.

### INT 17H, función 02H: Obtiene el estado de la impresora

El objetivo de esta operación es determinar el estado de la impresora:

```
MOV AH,02H          /Petición para leer el puerto
MOV DX,0 0          /Selecciona el puerto 0 para la impresora
INT 17H             /Llama al BIOS
TESTAH,00101001B    /¿Está preparada?
JNZ errormsg        /  no, mostrar mensaje
```

La operación regresa el mismo estado del puerto de impresión que la función 01H. Cuando el programa corre, si la impresora inicialmente está encendida el BIOS está habilitado para regre-

sar un mensaje de manera automática (su programa se supone que prueba y actúa de acuerdo con el estado de la impresora). Si su programa no examina el estado, su única indicación es el cursor intermitente. Si enciende la impresora en este punto, parte de la información de salida se pierde. En consecuencia, antes de ejecutar cualquiera de las operaciones de impresión del BIOS, verifique el estado del puerto; si hay un error, muestra un mensaje. (Las operaciones del DOS realizan de manera automática esta verificación, aunque su mensaje "No hay papel" se aplica a varias condiciones.) Cuando la impresora es encendida, el mensaje ya no aparece e inicia la impresión de forma normal sin pérdida de información.

En cualquier momento, una impresora puede quedarse sin papel o ser apagada sin advertirlo. Si está escribiendo programas para que los usen otros, incluya una prueba del estado antes de cada intento de imprimir.

## PUNTOS CLAVE

- Después de que la impresión esté terminada, utilice los comandos Avance de línea o Avance de página para limpiar el búfer de impresión.
- La función 40H del DOS (la selección preferida) imprime cadenas de caracteres, mientras que la función 05H del DOS y 17H de BIOS imprimen un solo carácter a la vez.
- El DOS proporciona un mensaje si existe algún error en la impresora; el BIOS sólo regresa un código de estado. Cuando utiliza la INT 17H del BIOS, verifique el estado de la impresora antes de imprimir.

## PREGUNTAS

**20-1.** Proporcione los caracteres de control de la impresora para (a) Tabulador horizontal; (b) Avance de página; (c) Retroceso; (d) Retorno de carro.

**20-2.** Codifique un programa que utilice la función 40H del DOS para los requisitos siguientes: (a) Salte la forma a la página siguiente; (b) imprima su nombre; (c) realice un Retorno de carro y un Avance de línea e imprima su ciudad y estado; (e) salte las formas.

**20-3.** Corrija la pregunta 20-2 para que utilice la función 05H del DOS.

**20-4.** Codifique una línea de encabezado que establezca modo condensado, defina un título (cualquier nombre), proporcione operaciones de Regreso de carro y de Avance de página y desactive el modo condensado.

**20-5.** La INT 17H del BIOS regresa un código de error, (a) ¿En dónde es regresado? (b) ¿Qué significa el código 08H? (c) ¿Qué significa el código 90H?

**20-6.** Corrija la pregunta 20-2 para usar la INT 17H del BIOS. Incluya una prueba para revisar el estado de la impresora.

**20**-7. Corrija la pregunta 20-2 de modo que el programa realice las partes (b), (c) y (d) cinco veces.

**20-8.** Corrija el programa de la figura 20-1 para ejecutar bajo la función 05H del DOS.

**20-9.** Corrija el programa de la figura 20-2, de modo que también muestre las líneas impresas.

CAPÍTULO 21

# Otras facilidades de Entrada/Salida

OBJETIVO

Describir la programación para el ratón, las instrucciones IN y OUT, puertos y la generación de sonido.

## INTRODUCCIÓN

Este capítulo describe el uso del ratón, el acceso a los puertos de la PC y la generación de sonido por medio de la bocina de la PC. Las instrucciones introducidas son:

- La INT 33H para el manejo del ratón
- IN y OUT para accesar los puertos

## CARACTERÍSTICAS DEL RATÓN

El ratón es un dispositivo común para apuntar, básicamente gobernado por un controlador que en general es instalado por una entrada en el archivo CONFIG.SYS o AUTOEXEC.BAT. El controlador debe ser instalado por un programa para responder a las acciones del ratón.

Todas las operaciones del ratón dentro de un programa son realizados por las funciones estándar de la INT 33H de la forma

```
MOV AX, función      ,-Petición para el ratón
...                  /Parámetros (si hay)
INT 33H              /Llama al controlador del ratón
```

Observe que a diferencia de las operaciones que utilizan el registro AH, las funciones de la INT 33H son cargadas en el registro AX completo.

La primer instrucción del ratón que un programa emite es la función OOH, la cual simplemente inicializa el controlador del ratón para el programa. Habitualmente, usted necesita emitir este comando una sola vez, al inicio del programa. La instrucción que sigue a la función OOH debe ser la función 01H, que hace que el apuntador del ratón aparezca en la pantalla. Después de eso, tiene opción de una amplia gama de operaciones con el ratón.

**Algunas definiciones básicas con el ratón**

- Mickey: Una unidad de medida del movimiento del ratón, aproximadamente 1/200 de una pulgada.
- Conteo de mickey: Número de mickey que el ratón rueda horizontal o verticalmente. El conteo de mickey es utilizado por el controlador del ratón para mover el apuntador en la pantalla un cierto número de pixeles.
- Apuntador del ratón: En modo de texto, el apuntador es un cuadro intermitente, en video inverso; en modo gráfico, el apuntador es una punta de flecha.
- Pixel: El elemento de la pantalla más pequeño que se puede direccionar. Por ejemplo, para modo de texto 03 hay ocho pixeles por byte.
- Umbral de velocidad: La velocidad en mickey por segundo que el ratón debe moverse para duplicar la velocidad del apuntador en la pantalla. Por omisión es de 64 mickey por segundo.

## FUNCIONES DEL RATÓN

Las siguientes son las funciones disponibles del ratón para la INT 33H; por lo común, relativamente pocas de ellas son usadas:

| | |
|---|---|
| OOH | Inicializa el ratón |
| 01H | Muestra el apuntador del ratón |
| 02H | Oculta el apuntador del ratón |
| 03H | Obtiene el estado del botón y la posición del apuntador |
| 04H | Establece posición del apuntador |
| 05H | Obtiene información del botón presionado del ratón |
| 06H | Obtiene información acerca de la liberación del botón |
| 07H | Fija límites horizontales para el apuntador |
| 08H | Fija límites verticales para el apuntador |
| 09H | Establece el tipo de apuntador gráfico |
| OAH | Establece el tipo de apuntador en texto |
| OBH | Lee los contadores de movimiento del ratón |
| OCH | Instala el manejador de interrupciones para eventos del ratón |
| ODH | Activa la emulación de una pluma óptica |
| OEH | Desactiva la emulación de una pluma óptica |

| | |
|---|---|
| OFH | Establece la relación mickey a pixel |
| 10H | Establece área de exclusión del apuntador |
| 13H | Establece el umbral de velocidad doble |
| 14H | Intercambia interrupción de evento de ratón |
| 15H | Obtiene tamaño del búfer para estado del controlador del ratón |
| 16H | Guarda el estado del controlador del ratón |
| 17H | Restaura el estado del controlador del ratón |
| 18H | Instala manejador alterno para eventos del ratón |
| 19H | Obtiene dirección del manejador alterno |
| 1AH | Fija la sensibilidad del ratón |
| 1BH | Obtiene la sensibilidad del ratón |
| 1CH | Establece la velocidad de interrupción del ratón |
| 1DH | Selecciona página de despliegue para el apuntador |
| 1EH | Obtiene página de despliegue para el apuntador |
| 1FH | Deshabilita el controlador del ratón |
| 20H | Habilita el controlador del ratón |
| 21H | Restablece el controlador del ratón |
| 22H | Establece lenguaje para mensajes de controlador del ratón |
| 23H | Obtiene el número del lenguaje |
| 24H | Obtiene información del ratón |

## OPERACIONES COMUNES DEL RATÓN

En esta sección examinamos las operaciones más comunes necesarias para la mayoría de los programas que utilizan el dispositivo.

### Función OOH: Inicializa el ratón

Éste es el primer comando para manejo del ratón que un programa emite; necesita ser utilizado sólo una vez. Basta cargar el AX con la función OOH y emitir la INT 33H. La operación no necesita parámetros de entrada, pero regresa estos valores:

- AX = 0000H si no está disponible el soporte del ratón o FFFFH, si está disponible
- BX = número de botones del ratón (si uno que se le da soporte está disponible)

Si un ratón que se,le da soporte está disponible, la operación *inicializa el controlador del ratón* como sigue:

- Establece el apuntador del ratón en el centro de la pantalla
- Si está visible el apuntador lo oculta
- Fija la página de despliegue del apuntador del ratón en cero
- Establece el apuntador del ratón de acuerdo al modo de pantalla:
  Modo de texto = rectángulo, color inverso
  Modo gráfico = forma de flecha

- Establece la razón mickey a pixel:
  Razón horizontal = 8 a 8
  Razón vertical = 16 a 8
- Establece los límites horizontal y vertical para el apuntador al mínimo y máximo
- Habilita el modo de emulación de pluma óptica
- Establece el umbral de velocidad doble a 64 mickey por segundo.

**Función 01H: Despliega el apuntador del ratón**

Después de emitir la función OOH, utilice esta operación para hacer que el apuntador del ratón sea mostrado. La operación no necesita parámetros de entrada y no regresa valores.

El controlador del ratón mantiene una *bandera del apuntador* que determina si se despliega o no el apuntador. Despliega el apuntador si la bandera es cero y lo oculta para cualquier otro número. Inicialmente, la cifra es - 1 ; la función 01H aumenta la bandera, por lo tanto, hace que el apuntador sea desplegado. (Véase también la función 02H.)

**Función 02H: Oculta el apuntador del ratón**

La práctica estándar es emitir esta función al final de la ejecución del programa, lo cual hace que el apuntador sea ocultado. La operación no necesita parámetros de entrada y no regresa valores.

La bandera del apuntador es desplegada cuando contiene un cero y se oculta para cualquier otro número. Esta función disminuye la bandera para forzarlo a que se oculte.

**Función 03H: Obtiene el estado del botón y la posición del apuntador**

Esta función regresa información útil acerca del ratón. No necesita parámetros de entrada, pero regresa estos valores:

- BX = Estado de los botones de acuerdo con la posición del bit, como sigue:
  Bit 0, botón izquierdo, donde 0 = arriba, 1 = abajo
  Bit 1, botón derecho, donde 0 = arriba, 1 = abajo
  Bit 2, botón central, donde 0 = arriba, 1 = abajo
  Bits 3-15 reservados
- CX = Coordenada horizontal (x)
- DX = Coordenada vertical (y)

Las coordenadas horizontal y vertical son expresadas en términos de *pixeles,* aun en modo de texto (8 por byte para modo de video 03). Las cifras siempre están dentro de los límites mínimo y máximo para el apuntador.

**Función 04H: Establece la posición del apuntador**

Utilice esta operación para fijar las coordenadas horizontal y vertical para el apuntador del ratón en la pantalla (las cifras para la posición están en términos de pixeles —8 bytes para modo de video 03):

```
MOV  AX, 04H          ,-Petición para colocar el apuntador del ratón
MOV  CX,horizon-loch  /Posición horizontal
MOV  DX, vertl-loch   /Posición vertical
INT  33H              /Llama al controlador del ratón
```

La operación coloca el apuntador en la nueva posición, ajusfando como sea necesario si está fuera de los límites mínimo y máximo.

### Código ilustrativo

El código siguiente ejemplifica el uso de las instrucciones del ratón estudiadas hasta este momento:

```
MOV  AX,OOH           /Petición para inicializar el ratón
INT  33H
CMP  AX,00H           /¿Ratón disponible?
JE   exit             / no, salir
MOV  AX,01H           /Petición para mostrar el apuntador
INT  33H              /Llama al controlador del ratón
MOV  AX,04H           /Petición para colocar el apuntador
MOV  CX,24            /Posición horizontal
MOV  DX,16            /Posición vertical
INT  33H              /Llama al controlador del ratón

MOV AX,02H            /Petición para ocultar el apuntador
INT 33H               /Llama al controlador del ratón
```

### Función 05H: Obtiene información de la pulsación del ratón

Para utilizar esta función para regresar información acerca de la presión del botón, coloque el número del botón en el BX, en donde 0 = izquierdo, 1 = derecho y 2 = central:

```
MOV   AX.05H          Petición para información de presión del botón
MOV   BX,button-no    Número de botón
INT   33H             Llama al controlador del ratón
```

La operación regresa el estado arriba abajo de todos los botones y el conteo de presiones y posición de botón requerido:

- AX = Estado de los botones de acuerdo con la posición del bit, como sigue:

  Bit 0, botón izquierdo, donde 0 = arriba, 1 = abajo

  Bit 1, botón derecho, donde 0 = arriba, 1 = abajo

Bit 2 = Botón central, donde 0 = arriba, 1 = abajo

Bits 3-15 reservados

- BX = Contador de presiones del botón
- CX = Coordenada horizontal (x) de la última presión del botón
- DX = Coordenada vertical (y) de la última presión del botón

La operación reestablece el contador de presiones de botón en cero.

**Función 06H: Información de liberación del botón**

Usar está función para regresar información acerca de la liberación del botón, coloque el número del botón en el BX (0 = izquierdo, 1 = derecho, 2 = central):

```
MOV AX.06H          ;Petición de información liberación del botón
MOV BX,button-no    /Número de botón
INT 33H             ,-Llama al controlador del ratón
```

La operación regresa el estado arriba abajo de todos los botones y el conteo de liberaciones y posición del botón requerido, como sigue:

- AX = Estado de los botones de acuerdo con la posición del bit, como sigue:

  Bit 0, botón izquierdo, donde 0 = arriba, 1 = abajo

  Bit 1, botón derecho, donde 0 = arriba, 1 = abajo

  Bit 2, botón central, donde 0 = arriba, 1 = abajo

  Bits 3-15 reservados

- BX = Contador de liberaciones del botón
- CX = Coordenada horizontal (x) de la última presión del botón
- DX = Coordenada vertical (y) de la última presión del botón

La operación restablece en cero el contador de liberaciones del botón.

**Función 07H: Fija los límites horizontales para el apuntador**

Esta operación fija los límites horizontales mínimo y máximo para el apuntador:

```
MOV   AX,07H        /Petición para fijar límite horizontal
MOV   CX,min-loch   /Límite inferior
MOV   DX,max-loch   /Límite superior
INT   33H           /Llama al controlador del ratón
```

Si el número mínimo es mayor que el máximo, la operación intercambia los números. También la operación mueve el apuntador al área nueva, si es necesario. Véase también las funciones 08H y 10H.

### Función 08H: Fija límites verticales para el apuntador

Esta operación fija los límites verticales mínimo y máximo para el apuntador:

```
MOV  AX,08H          /Petición para fijar límite vertical
MOV  CX,min-loch     ,-Límite inferior
MOV  DX,max-loch     ,•Límite superior
INT  33H             ,-Llama al controlador del ratón
```

Si el número mínimo es mayor que el máximo, la operación intercambia los números. También la operación mueve el apuntador al área nueva, si es necesario. Véase también las funciones 07H y 10H.

### Función OBH: Lee contadores de movimiento del ratón

Esta operación regresa el conteo de mickeys horizontales y verticales, desde la última llamada a la función (dentro del intervalo -32,768 a +32,767). Los números regresados son:

- CX = Conteo horizontal (un número positivo significa recorrido a la derecha, negativo significa a la izquierda)
- DX = Conteo vertical (un número positivo significa recorrido hacia abajo, negativo hacia arriba)

### Función OCH: Instala manejador de interrupciones para eventos del ratón

Su programa puede necesitar determinar de manera automática cuándo ha ocurrido alguna actividad (o evento) con el ratón. El objetivo de la función OCH es proporcionar un manejador de eventos, por eso el software del ratón interrumpe su programa y llama al manejador de eventos, el cual realiza la función que requiere y, cuando termina la tarea, regresa a su programa en el punto de ejecución.

Cargue el CX con una máscara del evento para indicar las acciones para las cuales el manejador debe responder y el ES:DX con la dirección segmento:desplazamiento de la rutina para manejar la interrupción:

```
MOV  AX,0CH          Petición de manejador de interrupción
LEA  CX,mask         Dirección de la máscara del evento
LEA  DX,handler      Dirección del manejador (ES:DX)
INT  33H             Llama al controlador del ratón
```

Defina la máscara del evento con los bits en uno necesarios:

| | |
|---|---|
| 0 | se movió el apuntador del ratón |
| 1 | se presionó el botón izquierdo |
| 2 | se liberó el botón izquierdo |
| 3 | se presionó el botón derecho |
| 4 | se liberó el botón derecho |
| 5 | se presionó el botón central |

- 6 = se liberó el botón central
- 7-15 = reservados, define como enO

Defina el manejador de interrupción como un procedimiento FAR. El controlador del ratón utiliza una llamada lejana para entrar al manejador de interrupción con estos registros establecidos:

- AX = La máscara del evento como se definió, salvo que los bits están en uno solo si la condición ha ocurrido
- BX = Estado del botón; si está establecido, los bits significan lo siguiente:
  0 botón izquierdo abajo
  1 botón izquierdo abajo
  2 botón central abajo
- CX = Coordenada horizontal (x)
- DX = Coordenada vertical (y)
- SI = Último conteo vertical de mickey
- DI = Último conteo horizontal de mickey
- DS = Segmento de datos para el controlador del ratón

A la entrada del programa al manejador de interrupción, guarde todos los registros e inicialice el registro DS con la dirección de su segmento de datos. Dentro del manejador, utilice sólo interrupciones del *BIOS*, no del DOS. Al salir saque todos los registros guardados.

**Función 10H: Fija el área de exclusión del apuntador**

Esta operación define un área en la pantalla en la que el apuntador no es mostrado:

```
MOV  AX,10H          ;Petición para fijar el área de exclusión
MOV  CX,upleft-x     ;Coordenada x de la esquina superior izquierda
MOV  DX,upleft-y     ,-Coordenada y de la esquina superior izquierda
MOV  SI,lowrgt-x     ,-Coordenada x de la esquina inferior derecha
MOV  DI,lowrgt-y     ,-Coordenada y de la esquina inferior derecha
INT  33H             ;Llama al controlador del ratón
```

Para reemplazar el área de exclusión, llame otra vez a la función con parámetros diferentes o vuelva a emitir la función OOH o 01H.

**Función 13H: Establece el umbral de velocidad doble**

Esta operación establece el umbral de velocidad en la que el movimiento del apuntador en la pantalla es duplicada. Cargue el DX con el nuevo valor (por omisión es 64 mickeys por segundo). (Véase también la función 1AH.)

**Función 1AH: Establece la sensibilidad del ratón**

La sensibilidad concierne al número de mickeys que el ratón necesita moverse antes que el apuntador se mueva. La función 1AH fija el movimiento horizontal y vertical del ratón en términos del número de mickeys por 8 pixeles, así como el umbral de velocidad en la que el movimiento del apuntador en la pantalla es duplicado (véase también las funciones OFH, 13H y 1BH):

```
MOV  AX, 1AH          |,-Petición para establecer sensibilidad del ratón
MOV  BX,horzon        |,-Mickeys horizontales (por omisión = 8)
MOV  CX,vertic        |,-Mickeys verticales (por omisión = 16)
MOV  DX,threshold     |/Umbral de velocidad (por omisión = 64)
INT  33H              |;Llama al controlador del ratón
```

**Función 1BH: Obtiene sensibilidad del ratón**

Esta operación regresa el movimiento horizontal y vertical de ratón en términos del número de mickeys por 8 pixeles, así como el umbral de velocidad a la cual el movimiento del apuntador en la pantalla es duplicado. (Véase la función 1AH para los registros y cifras que son regresados.)

**Función 1DH: Selecciona la página de despliegue para el apuntador**

La página de despliegue de video es establecida con la función 05H de la INT 10H. Para operaciones del ratón, coloque el número de la página en el BX y emita la función 1DH de la INT 33H.

**Función 1EH: Obtiene página de despliegue para el apuntador**

Esta operación regresa, en el BX, la actual página de despliegue de video.

**Función 24H: Obtiene información del ratón**

Esta operación regresa información acerca de la versión y tipo del ratón que está instalado:

- BH = Número principal de la versión
- BL = Número secundario de la versión
- CH = Tipo de ratón, en donde 1 = ratón de bus, 2 = ratón serial, 3 = ratón InPort, 4 = ratón PS/2 y 5 = ratón HP

## PROGRAMA PARA EL RATÓN

El programa de la figura 21-1 ilustra el uso del ratón. La pantalla muestra las posiciones horizontal y vertical del apuntador a medida que el usuario mueve el ratón. Los procedimientos principales son:

| | |
|---|---|
| BEGIN | Inicializa el programa, llama a B10INIT, D10PTR, G10CONV y a Q30DISP y sale al DOS cuando el usuario presiona el botón izquierdo. |
| B10INIT | Emite la función OOH de la INT 33H para inicializar el ratón (o para indicar que no está presente un controlador de ratón) y emite la función 01H para hacer que el apuntador del ratón aparezca. |
| D10PTR | Emite la función 03H para verificar y salir si el usuario ha presionado el botón izquierdo. Si no, el programa convierte las posiciones horizontal y vertical de número de pixeles en números binarios (por corrimiento de 3 bits a la derecha, efectivamente dividiendo entre 8). Si la posición es la misma que cuando se había verificado, la rutina repite la emisión de la función 03H; si la posición ha cambiado, el control regresa a donde fue llamado. |

```
TITLE     P21MOUSE (EXE)   Manejo del ratón
          .MODEL SMALL
          .STACK 64
          .DATA
XBINARY   DW  ¡    0                  Coordenada X binaria
YBINARY   DW       0                  Coordenada Y binaria
ASCVAL    DW       ?                  Campo ASCII

;                  Campos de despliegue en pantalla:
DISPDATA  LABEL    BYTE
XMSG      DB       'X = '             Mensaje X
XASCII    DW       ?                  Valor ASCII de X
          DB       ¡  ¡
YMSG      DB       'Y = ¡             Mensaje Y
YASCII    DW                          Valor ASCII de Y

          .CODE
BEGIN     PROC     FAR
          MOV      AX,@data           Inicializa
          MOV      DS,AX                el registro DS
          CALL     Q10CLEAR           Limpia la pantalla
          CALL     B10INIT            Inicializa el ratón
          CMP      AX, 00             ¿Ratón instalado?
          JE       A90                  no, salir
A10:
          CALL     D10PTR             Obtener apuntador del ratón
          CMP      BX, 01             ¿Botón presionado?
          JE       A80                  sí, salir
          CALL     Q2 0CURS           Colocar el cursor
          MOV      AX,XBINARY
          CALL     G10CONV            X a ASCII
          MOV      AX,ASCVAL
          MOV      XASCII,AX
          MOV      AX,YBINARY
          CALL     G10CONV            Y a ASCII
          MOV      AX, ASCVAL
          MOV      YASCII,AX
          CALL     Q30DISP            Desplegar valores de X y Y
          JMP      A10                Repetir
A80:
          CALL     H10HIDE             Ocultar apuntador del ratón
A90:
          CALL     Q10CLEAR            Limpiar pantalla
          MOV      AX,4C00H            Salir al DOS
          INT      21H
BEGIN     ENDP
B10INIT   PROC     NEAR
          MOV      AX,OOH             ,-Inicializar ratón
          INT      33H
          CMP      AX, 00             /¿Ratón instalado?
          JE       B90                /  no, salir
          MOV      AX,01H             /Mostrar apuntador
          INT      33H
B90:
          RET                         /Regresar a donde fue llamado
B10INIT   ENDP

.286
D10PTR    PROC     NEAR
D20:      MOV      AX,03H             /Obtener posición del apuntad
          INT      33H
          CMP      BX, 01             /¿Botón derecho presionado?
          JE       D90                   sí, significa salir
          SHR      CX, 03             /Dividir el número de pixel
```

Figura 21-1 Uso del ratón

```
          SHR     DX, 03              entre 8
          CMP     CX,XBINARY          ¿Ha cambiado la posición
          JNE     D30                 del apuntador?
          CMP     DX,YBINARY
          JE      D20                 no, repetir la operación
D30:                                  sí,
          MOV     XBINARY,CX          guardar la nueva posición
          MOV     YBINARY,DX
D90:
          RET                         Regresar a donde fue llamado
DIOPTR    ENDP

GIOCONV   PROC    NEAR                AX = X o Y binario
          MOV     ASCVAL,202OH        Limpia el campo ASCII
          MOV     CX,10               Fija el factor de división
          LEA     SI.ASCVAL+l         Carga la dirección de ASCVAL
          CMP     AX,CX               Compara la posición con 10
          JB      G30                 menor, pasar
          DIV     CL                  mayor, dividir entre 10
          OR      AH,3OH              Insertar 3 ASCII
          MOV     [SI],AH             Almacenar en el byte de la derecha
          DEC     SI                  Disminuir dirección de ASCVAL
G30:
          OR      AL,30H              •Insertar 3 ASCII
          MOV     [SI],AL             •Almacenar en el byte más a la izqu
          RET                         Regresar a donde fue llamado
GIOCONV   ENDP
HIOHIDE   PROC    NEAR
          MOV     AX,02H              ,•Ocultar apuntador
          INT     33H
          RET                         ¡Regresar a donde fue llamado
HIOHIDE   ENDP

QIOCLEAR  PROC    NEAR
          MOV     AX,0600H            ;Petición para limpiar la pantalla
          MOV     BH,30H              ;Colores
          MOV     CX,00               ;Pantalla
          MOV     DX,184FH            ;  completa
          INT     10H
          RET                         /Regresar a donde fue llamado
QIOCLEAR  ENDP

Q20CURS   PROC    NEAR
          MOV     AH,02H              ;Colocar el cursor
          MOV     BH, 0               ,-Página0
          MOV     DH, 0               ,-Renglón
          MOV     DL, 25              ,•Columna
          INT     10H
          RET                         ,-Regresar a donde fue llamado
Q20CURS   ENDP

Q30DISP   PROC    NEAR
          MOV     AH,40H              •Petición para desplegar
          MOV     BX,01               •Pantalla
          MOV     CX,14               ,-Número de caracteres
          LEA     DX,DISPDATA         •Área de despliegue
          INT     21H
          RET                         ;Regresar a donde fue llamado
Q30DISP   ENDP
          END     BEGIN
```

Figura 21-1 (continuación)

G10CONV Convierte los números binarios horizontal y vertical a caracteres ASCII desplegables. Observe que con 8 pixeles por byte, el valor horizontal regresado para la columna 79 (la posición de más a la derecha) es 79 x 8 = 632. El procedimiento divide este número entre 8 para obtener, en este caso, 79, el caso máximo. En consecuencia, la conversión puede suponer correctamente que los números regresados están ente 0 y 79.

Q30DISP Despliega las cifras horizontal y vertical.

Una manera de mejorar este programa es emitir la función OCH para establecer un manejador de interrupción. De esta manera, las instrucciones necesarias son llamadas automáticamente, siempre que el ratón esté activo.

## PUERTOS

Un *puerto* es un dispositivo que conecta a un procesador con el mundo exterior. Por medio de un puerto, el procesador recibe una señal desde un dispositivo de entrada y envía una señal a un dispositivo de salida. Los puertos son identificados por sus direcciones en el intervalo de OH—3FFH, o 1,024 puertos en total. Note que no son direcciones convencionales de memoria. Puede usar las instrucciones IN y OUT para manejar E/S directamente a nivel de puerto:

IN transfiere información desde un puerto de entrada al AL si es un byte y al AX si es una palabra. El formato general es

```
IN reg-acum,puerto
```

OUT transfiere información desde un puerto de salida al AL si es un byte y al AX si es una palabra. El formato general es

```
OUT puerto,reg-acum
```

Puede especificar una dirección de puerto estática o dinámicamente:

**Estáticamente.** Utilice un operando desde 0 hasta 255 directamente como:

```
Input    IN AL,port#     /Entrada de un byte
Output   OUT port#,AX    ,-Salida de una pe-labra
```

**Dinámicamente.** Utilice el contenido del registro DX, 0 a 65,535, indirectamente. Este método es adecuado para que incrementando el DX se procese de forma consecutiva las direcciones de los puertos. El ejemplo siguiente utiliza el puerto 60H:

```
MOV DX.60H          ;Puerto 60H (teclado)
IN  AL,DX           ;Obtiene un byte
```

Algunas de las direcciones de puertos principales son:

| | |
|---|---|
| 020H-023H | Registros de la máscara de interrupción |
| 040H-043H | Temporizador/contador |
| 060H | Entrada desde el teclado |
| 061H | Bocina (bits 0 y 1) |
| 200H-20FH | Controlador de juego |
| 278H-27FH | Adaptador paralelo de impresora LPT3 |
| 2F8H-2FFH | Puerto serial COM2 |
| 378H-37FH | Adaptador paralelo de impresora LPT2 |
| 3B0H-3BBH | Adaptador de despliegue monocromo |
| 3BCH-3BFH | Adaptador paralelo de impresora LPT1 |
| 3COH-3CFH | EGA/VGA |
| 3D0H-3DFH | Adaptador gráfico de color (CGA) |
| 3F0H-3F7H | Controlador de disco |
| 3F8H-3FFH | Puerto serial COMÍ |

Aunque la práctica recomendada es utilizar las interrupciones del DOS y del BIOS, puede con seguridad pasar por alto el BIOS cuando accese los puertos 21H, 40-42H, 60H, 61H y 201H. Por ejemplo, al arranque una rutina en ROM del BIOS busca el sistema por las direcciones de los adaptadores de puertos paralelos y seriales. Si la dirección del puerto serial es encontrada, el BIOS la coloca en su área de datos, empezando en la localidad de memoria 40:00H; si las direcciones de los puertos paralelos son encontradas, el BIOS las coloca en su área de datos, empezando en la localidad 40:08H. Cada localidad tiene espacio para entradas de una palabra. La tabla del BIOS para un sistema con dos puertos seriales y dos puertos paralelos podría verse así:

```
40 00  F803  COMÍ
40 02  F802  COM2
40 04  0000  no usada
40 06  0000  no usada
40 08  7803  LPT1
40 0A  7802  LPT2
40 OC  0000  no usada
40 0E  0000  no usada
```

Para utilizar la INT 17H del BIOS para imprimir un carácter, inserte el número del puerto de la impresora en el registro DX:

```
MOV  AH.0 0H          Petición para imprimir
MOV  AL,char          Carácter que se va a imprimir
MOV  DX,0             Puerto de impresora 0 = LPT1
INT  17H              Llama al BIOS
```

```
TITLE    P21PORT (COM)  Intercambia los puertos de impresión LPT1 y 2
BIOSDAT  SEGMENT AT 40H              Área de datos del BIOS
         ORG     8H                  Direcciones de los puertos de impresión
PARLPRT  DW      4 DUP(?)            4.palabras
BIOSDAT  ENDS

CODESG   SEGMENT PARA 'code'
         ASSUME  DS:BIOSDAT,CS:CODESG
         ORG     100H
BEGIN:
         MOV     AX, BIOSDAT
         MOV     DS,AX

         MOV     AX,PARLPRT(0)       Dirección de LPT1 al AX
         MOV     BX,PARLPRT(2)       Dirección del LPT2 al BX
         MOV     PARLPRT(0),BX       Intercambia direcciones
         MOV     PARLPRT(2),AX       Intercambia direcciones
         MOV     AX,4C00H            Salir al DOS
         INT     21H
CODESG   ENDS
         END     BEGIN
```

Figura 21-2 Intercambiar puertos de impresión

Algunos programas permiten imprimir sólo por medio de LPT1. Si tiene dos impresoras conectadas, como LPT1 y LPT2, podría usar el programa de la figura 21-2 para invertir (conmutar) sus direcciones en la tabla del BIOS.

## GENERACIÓN DE SONIDOS

La PC genera sonido por medio de una bocina integrada de imán permanente. Puede seleccionar una de dos formas para controlar la bocina o combinar ambas: (1) Utilice el bit 1 del puerto 61H para activar el circuito de Interfaz Programable de Periférico (PPI) Intel 8255A-5, o (2) emplee la compuerta del Temporizador Programable de Intervalo (PIT) Intel 8353-5. El reloj genera una señal de 1.19318 Mhz. El PPI controla la compuerta 2 en el bit 0 del puerto 61H.

El programa de la figura 21-3 genera una serie de notas de frecuencia ascendente. DURTION proporciona la duración de cada nota y TONE determina la frecuencia. Al inicio el programa accesa el puerto 61H y guarda el número que la operación envía. Una instrucción CLI limpia la bandera de interrupción para permitir un tono constante. El temporizador de intervalo genera un pulso de reloj de 18.2 pulsos por segundo que (a menos que usted codifique CLI) interrumpe la ejecución de su programa y hace que el tono oscile.

El contenido de TONE determina su frecuencia; números grandes provocan frecuencias bajas y números pequeños causan frecuencias altas. Después que la rutina B10SPKR toca cada nota, aumenta la frecuencia en TONE por medio de un corrimiento a la derecha de un bit (forma eficaz de dividir el número entre dos). Ya que TONE disminuye en este ejemplo se reduce la duración de lo que toca, también la rutina aumenta DURTION por medio de un corrimiento a la izquierda (forma eficaz de duplicar su número).

El programa termina cuando TONE es reducido a cero. Las cifras iniciales en DURTION y TONE no tienen significado técnico. Puede experimentar con otros números y tratar de ejecutar el programa sin la instrucción CLI.

```
TITLE   P21S0UND (COM)   Produce sonido desde la bocina
S0UNSG  SEGMENT PARA 'Code'
        ASSUME  CS:SOUNSG,DS:SOUNSG,SS:SOUNSG
        ORG     100H
BEGIN:  JMP     SHORT MAIN

DURTI0N DW      1000            ;Duración del tono
TONE    DW      256H            ;Frecuencia

MAIN    PROC    NEAR
        IN      AL,61H          Obtener datos del puerto
        PUSH    AX                y guardar
        CLI                     Limpiar interrupciones
        CALL    B10SPKR         Producir sonido
        POP     AX              Restablecer
        OUT     61H.AL            número del puerto
        STI                     Restablecer interrupciones
        RET
MAIN    ENDP

B10SPKR PROC    NEAR
B20:    MOV     DX,DURTION      ;Fijar duración del sonido
B30:
        AND     AL,11111100B    Poner en cero los bits 0 y 1
        OUT     61H,AL          Transmitir a la bocina
        MOV     CX,TONE         Fijar duración
B40:
        LOOP    B40             •Retraso
        OR      AL,00000010B    •Poner en uno el bit 1
        OUT     61H,AL          •Transmitir a la bocina
        MOV     CX,TONE         •Fijar duración
B50:
        LOOP    B50             Retraso
        DEC     DX              Reducir duración
        JNZ     B30             ¿Continuar?
        SHL     DURTION,1         no, aumentar duración
        SHR     TONE,1          Reducir frecuencia
        JNZ     B20             ¿Ahora es cero?
        RET                       sí, regresar
B10SPKR ENDP
SOUNSG  ENDS
        END     BEGIN
```

Figura **21-3** Generación de sonido

Podría usar cualquier variación lógica para tocar una secuencia de notas, a fin de, por ejemplo, llamar la atención del usuario. Podría también corregir el programa según la pregunta 21-7.

## PUNTOS CLAVE

- En modo de texto, el apuntador del ratón es un cuadro intermitente, en video inverso; en modo gráfico, el apuntador es una punta de flecha.
- Las operaciones del ratón utilizan la INT 33H, con un código de función en el AX.
- La primera operación de ratón a ejecutar es la función OOH, que inicializa el controlador del ratón.
- La función 01H es necesaria para mostrar el apuntador del ratón, 03H obtiene el estado del botón y 04H obtiene la posición del apuntador.

- Por medio de un puerto, un procesador recibe una señal desde un dispositivo de entrada y envía una señal a un dispositivo de salida. Los puertos son definidos por su dirección, en el intervalo 0H-3FFH, o 1,024 en total.
- La PC genera sonido por medio de una bocina de imán permanente. Puede seleccionar una de dos formas para manejar la bocina o combinar ambas.

## PREGUNTAS

21-1. Explique estos términos: (a) mickey; (b) contador del mickey; (c) apuntador del ratón.

21-2. Proporcione la función de la INT 33H para cada una de la operaciones siguientes del ratón:

(a) Leer el.contador de movimiento del ratón
**(b)** Obtener la información acerca de la presión del botón
(c) Ocultar el apuntador del ratón
**(d)** Establecer la posición del apuntador
(e) Obtener información acerca de la liberación del botón
(f) Instalar un manejador de interrupciones para eventos del ratón

21-3. ¿Cuál es el objetivo de la bandera del apuntador del ratón?

21-4. Codifique las instrucciones para los siguientes requisitos:

(a) Inicializar el ratón
**(b)** Mostrar el apuntador del ratón
(c) Obtener información del ratón
**(d)** Colocar el apuntador del ratón al renglón central, en el extremo derecho
(e) Obtener la sensibilidad del ratón
(f) Obtener el estado del botón y la posición del apuntador
(g) Ocultar el apuntador del ratón

21-5. Combinar los requisitos de la pregunta 21-4 en un programa completo. Puede ejecutar el programa bajo DEBUG, aunque a veces DEBUG puede recorrer el apuntador fuera de la pantalla.

21-6. Remítase a la figura 21-2 y codifique las instrucciones para invertir las direcciones de COMÍ y COM2.

21-7. Corrija el programa de la figura 21-3 para las situaciones siguientes: Generar notas que disminuyan en frecuencia; inicializar TONE en 01 y DURTION en un número grande. En cada ciclo, incrementar el número en TONE, disminuir el número en DURTION y terminar el programa cuando DURTION sea igual a cero.

PARTE F — Programación avanzada

CAPÍTULO 22

# Escritura de macros

OBJETIVO

Explicar la definición y uso de las macroinstrucciones.

## INTRODUCCIÓN

Para cada instrucción simbólica que usted codifica, el ensamblador genera una instrucción de lenguaje de máquina. Pero para cada enunciado codificado en un lenguaje de alto nivel, como C o Pascal, el compilador genera muchas instrucciones de lenguaje de máquina. A este respecto, puede pensar en un lenguaje de alto nivel como consistente de *macro* enunciados.

El ensamblador tiene facilidades que el programador puede usar para definir macros. Usted define un nombre específico para la macro, junto con el conjunto de instrucciones en lenguaje ensamblador que la macro va a generar. Después, siempre que necesite codificar el conjunto de instrucciones, sólo codifique el nombre de la macro y el ensamblador genera de manera automática las instrucciones que usted definió.

Las macros son útiles para los siguientes propósitos:

- Simplificar y reducir la cantidad de codificación repetitiva.
- Reducir errores causados por la codificación repetitiva.
- Linealizar un programa en lenguaje ensamblador para hacerlo más legible.

Ejemplos de funciones que pueden ser implementadas por macros son operaciones de entrada/salida que cargan registros y realizan interrupciones, conversiones de información ASCII y binaria, aritmética de palabras múltiples, rutinas para el manejo de cadenas de caracteres y división por sustracción.

## UNA DEFINICIÓN SENCILLA DE UNA MACRO

Para macros que usted necesite incluir con su programa, primero tiene que definirlas (o copiarlas de una biblioteca de macros). Una *definición de macro* aparece antes que cualquier definición de segmento. Examinemos una definición de una macro sencilla que inicializa los registros de segmento para un programa .EXE:

```
INITZ   MACRO                Define macro
        MOV     AX,@data      ) Cuerpo de
        MOV     DS, AX        } la definición
        MOV     ES, AX        } de la macro
        ENDM                 Fin de la macro
```

El nombre de esta macro es INITZ, aunque es aceptable cualquier otro nombre válido que sea único. La *directiva MACRO* en la primer línea le indica al ensamblador que las instrucciones que siguen, hasta *ENDM* ("fin de la macro"), son parte de la definición de la macro. La directiva ENDM termina la definición de la macro. Las instrucciones entre MACRO y ENDM comprenden el *cuerpo* de la definición de la macro.

Los nombres a que se hace referencia en la definición de la macro, ©datos, AX, DS y ES, deben estar definidos en alguna parte del programa o deben ser dados a conocer de alguna otra forma al ensamblador. En forma subsecuente se puede usar la macroinstrucción INITZ en el segmento de código en donde quiera inicializar los registros. Cuando el ensamblador encuentra la macroinstrucción INITZ, busca en una tabla de instrucciones simbólicas y, a falta de una entrada, busca macroinstrucciones. Ya que el programa contiene una definición de la macro INITZ, el ensamblador sustituye el cuerpo de la definición generando las instrucciones: la *expansión de la macro.* Un programa usaría la macroinstrucción INITZ sólo una vez, aunque otras macros están diseñadas para ser utilizadas cualquier número de veces y cada vez el ensamblador genera la misma expansión de la macro.

La figura 22-1 proporciona un listado del programa ensamblador. Esta versión particular del ensamblador lista la expansión de la macro con el número 1 a la izquierda de cada instrucción para indicar que una macroinstrucción la generó. Una expansión de macro sólo indica instrucciones para las cuales el código objeto es generado, de modo que directivas como ASSUME o PAGE no aparecerán.

Es difícil y molesto definir una macro para usar sólo una vez, pero podría catalogar esa macro en una biblioteca para usarla con todos los programas. Una sección posterior explica cómo catalogar macros en una biblioteca y cómo incluirlas de forma automática en cualquier programa.

## USO DE PARÁMETROS EN MACROS

Para hacer una macro flexible, puede definir nombres en ella como argumentos mudos (ficticios). La definición de la macro siguiente, llamada PROMPT, proporciona el uso de la función 09H del

```
                                page 60,132
                        TITLE   P22MACR1 (EXE)  Macro para inicializar
                        ------------------------------------------------
                                INITZ  MACRO            ;Define macro
                                MOV    AX,@data
                                MOV    DS,AX
                                MOV    ES,AX
                                ENDM                    /Termina macro
                        ------------------------------------------------
                                .MODEL SMALL
                                .STACK 64
                        ------------------------------------------------
                                .DATA
0000 54 65 73 74 20 6F  MESSGE   DB    'Test of macro instruction',13,10,'$
     66 20 6D 61 63 72
     6F 20 69 6E 73 74
     72 75 63 74 69 6F
     6E 0D 0A 24
                        ------------------------------------------------
                                .CODE
0000                    BEGIN   PROC   FAR
                                INITZ                   /Macroinstrucción
0000 B8    -- R      1          MOV    AX,@data
0003 8E D8           1          MOV    DS,AX
0005 8E C0           1          MOV    ES,AX
0007 B4 09                      MOV    AH,09H           ,-Petición para desplegar
0009 8D 16 0000 R               LEA    DX,MESSGE        /Mensaje
000D CD 21                      INT    21H
OOOF B8 4C0O                    MOV    AX,4C00H         /Salir al DOS
0012 CD 21                      INT    21H
0014                    BEGIN   ENDP
                                END    BEGIN
```

```
Macros:
          Ñ a m e                 Lines
INITZ                               3

Segments and Groups:
            Ñ a m e             Length   Align   Combine   Class
DGROUP . . . . . .               GROUP
 _DATA   . . . . . .              001C    WORD    PUBLIC    'DATA'
 STACK   . . . . . .              0040    PARA    STACK     'STACK'
 TEXT. . . . . .                  0014    WORD    PUBLIC    'CODE•

Symbols:
            Ñ a m e                 Type    Valué    Attr
BEGIN. . . . . .                F PROC   0000    _TEXT    Length = 0014
MESSGE. . . . . .               L BYTE   0000     DATA
OCODE . .                       TEXT     _TEXT
SFI LEÑAME                      TEXT     p22macrl
```

Figura 22-1 Macroinstrucción ensamblada y simplificada

DOS para desplegar cualquier mensaje. Cuando se usa la macroinstrucción, el programador tiene que proporcionar el nombre del mensaje, el cual hace referencia a un área de datos terminada por un signo de dólar.

```
PROMPT  MACRO MESSGE        /Argumento mudo

        MOV  AH,0  9H

        LEA    DX,MESSGE

        INT    21H

        ENDM                 /Fin de la macro
```

Un *argumento mudo* en una definición de macro indica al ensamblador que haga coincidir su nombre con cualquier aparición del mismo nombre en el cuerpo de la macro. Por ejemplo, el argumento mudo MESSGE también aparece en la instrucción LEA.

Cuando utiliza la macroinstrucción PROMPT, usted proporciona un parámetro como el nombre real del mensaje que será desplegado, por ejemplo,

```
PROMPT MESSAGE2
```

En este caso, MESSAGE2 tiene que estar apropiadamente definido en el segmento de datos. El parámetro en la macroinstrucción corresponde al argumento mudo en la definición original de la macro:

```
Definición de macro:   PROMPT  MACRO  MESSGE   (argumento)
                                        I
Macroinstrucción:        PROMPT    MESSAGE2  (parámetro)
```

El ensamblador ya ha hecho corresponder el argumento en la definición original de la macro con la instrucción LEA en el cuerpo de la macro. Ahora sustituye el (los) parámetro(s) de la macroinstrucción MESSAGE2 por la presencia de MESSGE en la instrucción LEA y la sustituye por cualquier otra aparición de MESSGE.

La definición de la macro y la expansión de la macro son mostradas completamente en la figura 22-2. El programa también define la macro INITZ al inicio y la usa en el segmento de código.

Un argumento mudo puede contener cualquier nombre válido, incluyendo un nombre de registro tal como CX. Puede definir una macro con cualquier número de argumentos mudos, separados por coma, hasta la columna 120 de una línea. El ensamblador sustituye los par .metros de la macro instrucción por los argumentos mudos en la definición de la macro, entrada por entrada, de izquierda a derecha.

## COMENTARIOS

Puede codificar comentarios en una definición de macro para clarificar su objetivo. Una directiva COMMENT o un punto y coma indican una línea de comentario. El ejemplo siguiente utiliza un punto y coma para indicar un comentario:

```
PROMPT    MACRO    MESSGE
;         Esta macro permite desplegar comentarios
          MOV      AH,09H
          LEA      DX,MESSGE
          INT      21H
          ENDM
```

Puesto que la omisión es listar sólo las instrucciones que generan código objeto, el ensamblador no despliega de forma automática un comentario cuando expande una definición de macro. Si usted quiere que un comentario aparezca dentro de una expansión, utilice la directiva de listado .LALL ("list all, listar todo", incluyendo el punto inicial) antes de solicitar la macroinstrucción:

```
                                        page    60,132
                                TITLE   P22MACR2 (EXE)   Uso de parámetros
------------------------------------------------------------------------------
                                INITZ   MACRO                   ;Define macro
                                        MOV     AX,@data
                                        MOV     DS,AX
                                        MOV     ES,AX
                                        ENDM
                                                                ;Termina macro
                                PROMPT  MACRO   MESSGE          ;Define macro
                                        MOV     AH,09H
                                        LEA     DX,MESSGE
                                        INT     21H
                                        ENDM                    ;Termina macro
------------------------------------------------------------------------------
                                        .MODEL SMALL
                                        .STACK 64
------------------------------------------------------------------------------
                                        .DATA
0000 43 75 73 74 6F 6D          MESSG1  DB      'Customer ñame?'
     65 72 20 6E 61 6D
     65 3F 24
OOOF 43 75 73 74 6F 6D          MESSG2  DB      'Customer address?', '$'
     65 72 20 61 64 64
     72 65 73 73 3F 24
------------------------------------------------------------------------------
                                        .CODE
0000                            BEGIN   PROC    FAR
                                        INITZ
0000 B8 _____R                  1       MOV     AX,©data
0003 8E D8                      1       MOV     DS,AX
0005 8E CO                      1       MOV     ES,AX
                                 PROMPT MESSG2
0007 B4 09                      1       MOV     AH,09H
0009 8D 16 OOOF R               1       LEA     DX.MESSG2
000D CD 21                      1       INT     21H
OOOF B8 4C00                            MOV     AX,4C00H        ;Sale al DOS
0012 CD 21                              INT     21H
0014                            BEGIN   ENDP
                                        END     BEGIN
```

Figura **22-2** Uso de parámetros en macro

```
.LALL

PROMPT  MESSAGE1
```

Una definición de macro puede tener varios comentarios, algunos de los cuales puede necesitar listar y algunos otros suprimir. Aun se utiliza .LALL para listarlo, pero debe cidificar dos punto y coma seguidos (;;) antes de los comentarios que siempre serán suprimidos. (Por omisión, el ensamblador tiene .XALL, que causa un listado sólo de las instrucciones que generan código objeto.) Por otra parte, puede no querer listar el código fuente de una expansión de macro, en especial si la macroinstrucción es usada varias veces en un programa. En ese caso, codifique la directiva de listado .SALL ("suprimir todo"), que reduce el tamaño del programa impreso, aunque no tiene efecto sobre el tamaño del módulo objeto generado.

Una directiva de listado mantiene su efecto a lo largo del programa hasta que encuentre otra. Puede colocarlas en un programa para hacer que cifras macros listen sólo el código objeto generado (.XALL), otras listen el código objeto y los comentarios (.LALL) y algunas más supriman del listado tanto comentarios como código objeto (.SALL).

```
                                        page    60,132
                                TITLE   P22MACR3 (EXE)   Uso de .LALL y .SALL

                                INITZ   MACRO                   ;Define macro
                                        MOV     AX,@data
                                        MOV     DS,AX
                                        MOV     ES,AX

                                        ENDM                    ;Termina macro

                                PROMPT  MACRO MESSGE
                                ;       Esta macro despliega cualquier mensaje
                                ;;      Genera código que llama al servicio del DOS
                                        MOV     AH,09H          ;Petición para desplegar
                                        LEA     DX,MESSGE
                                        INT     21H
                                        ENDM
                                        .MODEL  SMALL
                                        .STACK  64

                                        .DATA
 43 75 73 74 6F 6D              MESSG1  DB      'Customer ñame?', 13, 10, '$
 65 72 20 6E 61 6D
 65 3F OD OA 24
 43 75 73 74 6F 6D              MESSG2  DB      'Customer address?', 13, 10,
 65 72 20 61 64 64
 72 65 73 73 3F OD
 OA 24

                                        .CODE
0000                            BEGIN   PROC    FAR
                                        .SALL
                                        INITZ
                                        PROMPT  MESSG1
                                        .LALL
                                        PROMPT  MESSG2
                          1             Esta macro despliega cualquier mensa

OOOF B4 09                      1               MOV     AH,09H          ;Petición para
0011 8D 16 0011 R               1               LEA     DX,MESSG2
0015 CD 21                      1               INT     21H
0017 B8 4C00                                    MOV     AX,4C00H        ;Sale al DOS
001A CD 21                                      INT     21H
001C                            BEGIN   ENDP
                                        END     BEGIN
```

Figura 22-3 Listado y supresión de expansión de macro

El programa de la figura 22-3 ilustra las características anteriores. Define las dos macros, INITZ y PROMPT, ya descritas. El segmento de código contiene la directiva de listado .SALL para suprimir la expansión de INITZ y la primer expansión de PROMPT. Para el segundo uso de PROMPT, la directiva de listado .LALL hace que el ensamblador liste el comentario y la expansión de la macro. Pero observe que en la definición de la macro para PROMPT el comentario en la expansión de la macro que tiene un doble punto y coma (;;) no es listado.

MASM 6.0 introdujo los términos .LISTMACROALL, .LISTMACRO y .NOLISTMACRO para .LALL, .XALL y .SALL, respectivamente.

## USO DE UNA MACRO DENTRO DE UNA DEFINICIÓN DE MACRO

Una definición de macro puede tener una referencia a otra macro definida. "Considere una macro sencilla llamada DOS21 que carga una función en el registro AH y emite la INT 21H:

```
DOS21  MACRO  DOSFUNC
       MOV    AH,DOSFUNC
       INT    21H
       ENDM
```

Usar esta macro DOS21 para aceptar entrada desde el teclado, codifique

```
LEA    DX,NAMEPAR
DOS21  OAH
```

El código generado por DOS21 cargaría la función OAH en el AH y emitiría la INT 21H para entrada desde el teclado. Ahora suponga que tiene otra macro, llamada DISP, que carga la función 02H de la INT 21H en el registro AH para desplegar un carácter:

```
DISP  MACRO   CHAR
      MOV     AH,02H
      MOV     DL, CHAR
      INT     21H
      ENDM
```

Por ejemplo, para desplegar un signo de interrogación codifique la macro como DISP '?' Podría cambiar DISP para aprovechar la macro DOS21H para hacer referencia a DOS21 dentro de la definición de DISP:

```
DISP   MACRO   CHAR
       MOV     DL, CHAR
       DOS21   02H
       ENDM
```

Ahora, si usted codifica la macro DISP como DISP '?', el ensamblador genera

```
MOV   DL,'?'
MOV   AH,02H
INT   21H
```

## LA DIRECTIVA LOCAL

Algunas macros necesitan que se definan elementos de datos y etiquetas de instrucciones dentro de la definición de la macro. Si utiliza la macro más de una vez en el mismo programa y el ensamblador define los elementos de datos para cada aparición, los nombres duplicados harían que el ensamblador genere un mensaje de error. Para asegurar que cada nombre generado es único, codifique la directiva LOCAL inmediatamente después de la instrucción MACRO, aun antes de los comentarios. Su formato general es

```
LOCAL  mudo-1,  raudo-2,  ...  ;Uno o más argumentos mudos        ,
```

La figura 22-4 ilustra el uso de LOCAL. El objetivo del programa es realizar división por medio de sustracciones sucesivas. La rutina resta el divisor del dividendo y agrega uno al cociente hasta que el dividendo es menor que el divisor. El procedimiento necesita dos etiquetas: COMP para la dirección del ciclo y OUT para salir del procedimiento al terminar. Ambas, COMP y OUT, están definidas como LOCAL y pueden tener cualquier nombre válido.

```
                        TITLE   P22MACR4  (EXE)   Uso de LOCAL

                        INITZ   MACRO                   /Define  macro
                                MOV     AX,@data
                                MOV     DS, AX
                                MOV     ES, AX
                                ENDM                    ,-Fin de macro
                        DIVIDE  MACRO   DIVIDEND, DIVISOR, QUOTIENT
                                LOCAL   COMP
                                LOCAL   OUT
                                AX = dividendo,  BX   divisor,  CX = cociente
                                MOV     AX, DIVIDEND      /Asigna  dividendo
                                MOV     BX,DIVISOR        /Asigna  divisor
                                SUB     CX.CX             /Pone en cero al cociente
                        COMP:
                                CMP     AX,BX              ¿Dividendo < divisor?
                                JB      OUT                  sí,  salir
                                SUB     AX.BX             Dividendo - divisor
                                INC     CX                Sumar al cociente
                                JMP     COMP
                        OUT:
                                MOV     QUOTIENT,CX       /Almacenar el cociente
                                ENDM                      /Termina  macro

                                .MODEL  SMALL
                                .STACK  64

                                .DATA
0000    0096            DIVDND  DW      150               Dividendo
0002    001B            DIVSOR  DW      27                Divisor
0004    0000            QUOTNT  DW      ?                 Cociente

                                .CODE
0000                    BEGIN   PROC    FAR
                                .LALL
                                INITZ
0000                            MOV     AX,©data
0003    8E  D8                  MOV     DS.AX
0005    8E  C0                  MOV     ES, AX
                                DIVIDE  DIVDND,DIVSOR, QUOTNT
                                AX = dividendo,  BX = divisor,  CX = cociente
0007    Al  0000  R             MOV     AX,DIVDND         Asigna  dividendo
000A    8B  1E  0002  R         MOV     BX,DIVSOR         Asigna  divisor
000E    2B  C9                  SUB     CX, CX            Pone en cero al cociente
0010                    ??0000:
0010    3B  C3                  CMP     AX,BX             ¿Dividendo < divisor?
0012    72  05                  JB      ??0001              sí,  salir
0014    2B  C3                  SUB     AX,BX             Dividendo - divisor
0016    41                      INC     CX                Sumar al cociente
0017    EB  F7                  JMP     ??0000
0019                     ??0001:
0019    89  OE  0004  R         MOV     QUOTNT,CX         /Almacenar el cociente
001D    B8  4C00                MOV     AX,4C00H          /Sale al DOS
0020    CD  21                  INT     21H
0022                    BEGIN   ENDP
                                END     BEGIN
```

Figura 22-4 Uso de LOCAL

En la expansión de la macro, la etiqueta simbólica generada para COMP es ??0000 y para OUT es ??0001. Si utiliza la macroinstrucción DIVIDE otra vez en el mismo programa, las etiquetas simbólicas se convertirían en ??0002 y ??0003, respectivamente. De esta manera, la característica asegura que las etiquetas generadas dentro de un programa son únicas.

## INCLUIR (INCLUDE) DESDE UNA BIBLIOTECA DE MACROS

Definir una macro, como INITZ o PROMPT, y usarla sólo una vez en un programa no es muy productivo. El enfoque habitual es catalogar las macros en una biblioteca en disco bajo un nombre descriptivo, como MACRO.LIB. Usted sólo tiene que reunir todas las definiciones de sus macros en ui archivo y almacenar el archivo en disco:

```
INITZ   MACRO

        ENDM

PROMPT  MACRO  MESSGE

        ENDM
```

Para usar cualquiera de las macros catalogadas, en lugar de codificar las definiciones MACRO al inicio del programa utilice la directiva INCLUDE así:

```
        INCLUDE  D:MACRO.LIB

        INITZ
```

El ensamblador accesa el archivo llamado MACRO.LIB en la unidad D e incluye ambas definiciones de macro, INITZ y PROMPT, en el programa. En este ejemplo, sólo INITZ es realmente necesaria. El listado ensamblado contendrá una copia de las definiciones de las macros, indicada con una letra C en la columna 30 del archivo LST. Luego de cada macroinstrucción estará la expansión de la macro, junto con su código objeto generado, indicada por un signo de más (+) en la columna 31.

Ya que un ensamblado MASM (hasta, e incluso la versión 5.1) es una operación de dos pasadas, puede usar las siguientes instrucciones para hacer que INCLUDE suceda sólo en el paso 1 (en lugar de en ambas pasadas):

```
IFl
        INCLUDE  D:\MACRO.LIB
ENDIF
```

IFl y ENDIF son directivas condicionales. IFl le indica al ensamblador que accese la biblioteca sólo en la pasada 1 del ensamblado. ENDIF termina la lógica de IF. Una copia de la definición de la macro ya no aparecerá en el listado, lo que ahorra tiempo y espacio. (MASM versión 6.0 y siguientes no necesitan directivas que hagan referencia a las dos pasadas.)

El programa de la figura 22-5 contiene las instrucciones previamente descritas IFl, INCLUDE y ENDIF, aunque el ensamblador lista sólo el ENDIF en el archivo LST. Las dos macroinstrucciones usadas en el segmento de código, INITZ y PROMPT, son ambas catalogadas en MACRO.LIB.

```
                                page     60,132
                        TITLE   P22MACR5  (EXE)   Prueba de INCLUDE

                                .MODEL   SMALL
                                .STACK   64

                                .DATA
0000 54 65 73 74 20 6F  MESSGE  DB       'Test of
     66 20 6D 61 63 72
     6F 24
                        ;-------------------------------------------
                                .CODE
0000                    BEGIN   PROC     FAR
                                INITZ
0000 B8 _____R      1           MOV      AX,@data
0003 8E D8          1           MOV      DS,AX
0005 8E CO          1           MOV      ES,AX
                                PROMPT   MESSGE
0007 B4 09          1           MOV      AH, 09       /Petición para desplegar
0009 8D 16 0000 R   1           LEA      DX, MESSGE
000D CD 21          1           INT      21H
OOOF B8 4C0O                    MOV      AX,4C00H     ,-Sale al DOS
0012 CD 21                      INT      21H
0014                    BEGIN   ENDP
                                END      BEGIN
```

Figura 22-5 Uso de la biblioteca INCLUDE

Fueron almacenadas juntas simplemente como un archivo en disco bajo ese nombre por medio de un programa editor.

La colocación de INCLUDE no es crítica, pero la directiva debe aparecer antes de cualquier macroinstrucción que haga referencia a una entrada de la biblioteca.

### La directiva PURGE

La ejecución de una instrucción INCLUDE hace que el ensamblador incluya todas las definiciones de macros que están especificadas en la biblioteca. Sin embargo, suponga que una biblioteca contiene las macros INITZ, PROMPT y DIVIDE, pero que el programa sólo necesita INITZ. La directiva PURGE permite que usted "elimine" las macros PROMPT y DIVIDE que no necesita del ensamblado actual:

```
IFl

        INCLUDE D:\MACRO.LIB        ;Incluye la biblioteca completa

ENDIF

PURGE   PROMPT,DIVIDE               ;Elimina las macros no necesarias


INIT    CSEG,DATA,STACK             ,-Utiliza la macro restante
```

Una operación PURGE facilita sólo el ensamblado de un programa y no tiene efecto sobre las macros almacenadas en la biblioteca.

## CONCATENACIÓN

El carácter ampersán (&) indica al ensamblador que una (concatene) texto o símbolos. La siguiente macro MOVE proporciona la generación de la instrucción MOVSB, MOVSW o MOVSD:

```
MOV   MACRO  TAG
      REP  MOVS&TAG
      ENDM
```

Un usuario podría codificar esta instrucción como MOVE B, MOVE W o MOVE D. El ensamblador concatena el parámetro con la instrucción MOVS, para producir REP MOVSB, REP MOVSW o REP MOVSD, respectivamente. (Este ejemplo es muy trivial y sólo es para fines ilustrativos.)

## DIRECTIVAS DE REPETICIÓN

Las directivas de repetición REPT, IRP e IRPC hacen que el ensamblador repita un bloque de instrucciones terminadas por ENDM. (MASM 6.0 introdujo los términos REPEAT, FOR y FORC para REPT, IRP e IRPC, respectivamente.) Estas directivas no tienen que estar contenidas en una definición MACRO, pero si lo están, es necesario un ENDM para finalizar la repetición y un segundo ENDM para terminar la definición MACRO.

### REPT: Repetir

La directiva REPT provoca la repetición de un bloque de instrucciones hasta ENDM de acuerdo con el número de veces en la expresión de entrada:

```
REPT    expresión
```

El ejemplo siguiente inicializa N a cero y después repite la generación de DB N cinco veces:

```
N =     0
REPT    5
N=      N + 1
DB      N
ENDM
```

El resultado es la generación de cinco instrucciones DB, desde DB 1 hasta DB 5. Un uso para REPT podría ser para definir una tabla o parte de una tabla. El ejemplo siguiente define una macro que utiliza REPT para hacer sonar la bocina cinco veces:

```
BEEPSPKR  MACRO
          MOV     AH,02H       Petición de salida
          MOV     DL,07        Carácter de campana
           REPT   5            Repetir cinco veces
            INT   21H          Llama al DOS
           ENDM                Fin de REPT
          ENDM                 Fin de MACRO
```

### IRP: Repetición indefinida

La directiva IRP hace que se repita un bloque de instrucciones hasta ENDM. El formato general es

```
IRP  arg_mudo, <argumentos>
```

Los argumentos, contenidos en paréntesis angulares, son cualesquier número de símbolos válidos, incluyendo cadenas de caracteres, numéricos o constantes aritméticas. El ensamblador genera un bloque de código para cada argumento. En el ejemplo siguiente el ensamblador genera DB 3, DB 9, DB 17, DB 25 y DB 28:

```
IRP   N,<3,19,17,25,28>
DB    N
```

### ERPC: Repetición indefinida con carácter

La directiva **IRPC** hace **que** se repita un **bloque** de instrucciones hasta ENDM. El formato general es

```
IRPC  arg_mudo,cadena
```

El ensamblador genera un **bloque** de código para cada carácter en la cadena. En el ejemplo siguiente, el ensamblador genera desde DW 3 hasta DW 8:

```
IRPC    N,345678
DW      N
ENDM
```

## DIRECTIVAS CONDICIONALES

El lenguaje ensamblador permite usar varias directivas condicionales. Usamos IF1 anteriormente para incluir una entrada de biblioteca sólo durante la pasada 1 de un ensamblado. Las directivas condicionales son muy útiles dentro de una definición de macro, pero no están limitadas a ese propósito. Cada directiva IF debe tener su correspondiente ENDIF para terminar una condición que se prueba. Un ELSE opcional puede proporcionar una acción alterna. A continuación está el formato general para la familia IF de directivas condicionales:

```
IFxx     (condición)
                              bloque
ELSE     (opcional)
                              condicional
ENDIF    (fin del IF)  )
```

La omisión de ENDIF provoca el mensaje de error "Condicional no determinado". Si una condición examinada es verdadera, el ensamblador ejecuta el bloque condicional hasta el ELSE o, si no está ELSE, hasta el ENDIF. Si la condición es falsa, el ensamblador ejecuta el bloque condicional que sigue al ELSE; si no está presente un ELSE, no genera código alguno para el bloque condicional.

A continuación se explican las diferentes directivas condicionales:

- IF expresión Si la expresión que se evalúa es diferente de cero, el ensamblador ensambla las instrucciones dentro del bloque condicional.
- IFE expresión Si la expresión que se evalúa es cero, el ensamblador ensambla las instrucciones dentro del bloque condicional.
- IFl (sin expresión) Si el ensamblador está procesando la pasada 1, actúa sobre las instrucciones en el bloque condicional.
- IF2 (sin expresión) Si el ensamblador está procesando la pasada 2, actúa sobre las instrucciones en el bloque condicional.
- IFDEF símbolo Si el símbolo está definido en el programa o es declarado como EXTRN, el ensamblador procesa las instrucciones en el bloque condicional.
- IFNDEF símbolo Si el símbolo no está definido en el programa o no es declarado como EXTRN, el ensamblador procesa las instrucciones en el bloque condicional.
- IFB < argumento > Si el argumento está en blanco, el ensamblador procesa las instrucciones en el bloque condicional. El argumento necesita los paréntesis angulares.
- IFNB < argumento > Si el argumento no está en blanco, el ensamblador procesa las instrucciones en el bloque condicional. El argumento necesita los paréntesis angulares.
- IFIDN < arg-1 >, < arg-2 > Si la cadena del argumento 1 es idéntica a la cadena del argumento 2, el ensamblador procesa las instrucciones en el bloque condicional. El argumento necesita los paréntesis angulares.
- IFDIF <arg-l >, < arg-2 > Si la cadena del argumento 1 es diferente de la cadena del argumento 2, el ensamblador procesa las instrucciones en el bloque condicional. El argumento necesita los paréntesis angulares.

IF e IFE pueden usar operadores relaciónales EQ (igual), NE (diferente), LT (menor que), LE (menor o igual a), GT (mayor que) y GE (mayor o igual a), como, por ejemplo, en la instrucción

```
IF expresiónl EQ expresión2
```

A continuación está un ejemplo sencillo del uso de IFNB (si no es blanco). Toda INT 21H requiere de una función en el registro AH, y algunas peticiones también necesitan un número en el DX. La macro DOS21 utiliza IFNB para probar un argumento no blanco para el DX; si el resultado es verdadero (el argumento no es blanco), el ensamblador genera la instrucción MOV que carga el DX:

```
DOS21  MACRO   DOSFUNC,DXADDRES
       MOV     AH,DOSFUNC
       IFNB    <DXADDRES>
       MOV     DX,OFFSET  DXADDRES
       ENDIF
       INT     21H
       ENDM
```

El uso de DOS21 para entrada sencilla desde el teclado sólo necesita cargar el AH con un número, en este caso la función 01H:

```
DOS21  01
```

El ensamblador genera MOV AH,01 y la INT 21H. La entrada de una cadena de caracteres necesita de la función OAH en el AH y la entrada de la dirección en el DX. Podía codificar la macro DOS21 como

```
DOS21   OAH,IPFIELD
```

Entonces el ensamblador genera ambas instrucciones MOV y la INT 21H.

### La directiva EXITM

Una definición de macro puede contener una directiva condicional que pruebe buscando una condición grave. Si la condición es verdadera, el ensamblador sale desde cualquier expansión posterior de macro. La directiva EXITM sirve para este propósito:

```
IFxx  [condición]
...   (condición no válida)
EXITM


ENDIF
```

Si el ensamblador encuentra EXITM en una expansión de una macroinstrucción, descontinúa la expansión de la macro y reasume el procesamiento después de ENDM. También puede utilizar EXITM para terminar las directivas REPT, IRP e IRPC, aun si ellas están contenidas dentro de una definición de macro.

### Macro que utiliza las condiciones IF e IFNDEF

La estructura del programa de la figura 22-6 contiene una definición de macro llamada DIVIDE que genera una rutina para realizar la división por medio de restas sucesivas. El usuario tiene que codificar la macroinstrucción con parámetros para el dividendo, divisor y cociente, en ese orden. La macro utiliza IFNDEF para verificar si el programa realmente tiene sus definiciones. Para cualquier entrada no definida, la macro incrementa un campo llamado CNTR. Técnicamente, CNTR podría tener cualquier nombre válido y es para uso temporal en una definición de macro. Después de verificar los tres parámetros, la macro verifica CNTR para saber si es diferente de cero:

```
IF      CNTR
,•   Expansión de macro terminada
EXITM
ENDIF
```

Si CNTR tiene un valor diferente de cero, el ensamblador genera el comentario y sale (EXITM) de cualquier expansión de macro. Observe que una instrucción pone en cero a CNTR y también que los bloques IFNDEF sólo necesitan poner en 1 a CNTR en lugar de incrementarlo.

Si el ensamblador pasa todas estas pruebas de seguridad, genera la expansión de la macro. En el segmento de código, la segunda macroinstrucción DIVIDE contiene un dividendo y cociente no válidos y sólo genera comentarios. Una manera de mejorar la macro sería probar si el divisor no es cero y si dividendo y divisor tienen el mismo signo; para ello, utilice instrucciones de ensamblador en lugar de directivas condicionales.

```
                                page    60,132
                        TITLE   P22MACR6  (EXE)  Prueba de IF y de IFNDEF

                        INITZ   MACRO                   ;Define macro
                                MOV     AX,@data        ,-Inicializa
                                MOV     DS,AX           ;  registros de
                                MOV     ES,AX           ;  segmentos
                                ENDM                    ;Fin de la macro
                        DIVIDE  MACRO   DIVIDEND,DIVISOR,QUOTIENT
                                LOCAL   COMP
                                LOCAL   OUT
                                CNTR    = 0
                                AX = divdo,  BX = dvsor,  CX = ente
                                IFNDEF DIVIDEND
                                        Dividendo no definido
                                CNTR    = CNTR +1
                                ENDIF
                                IFNDEF  DIVISOR
                                        Divisor no definido
                                CNTR    = CNTR +1
                                ENDIF
                                IFNDEF  QUOTIENT
                                        Cociente no definido
                                CNTR    = CNTR + 1
                                ENDIF
                                IF      CNTR
                                        Expansión de macro terminada
                                EXITM
                                ENDIF
                                MOV     AX,DIVIDEND     Asigna dividendo
                                MOV     BX,DIVISOR      Asigna divisor
                                SUB     CX,CX           Pone en cero al cociente
                        COMP:
                                CMP     AX,BX           ¿Dividendo < divisor?
                                JB      OUT               sí, salir
                                SUB     AX,BX           Dividendo - divisor
                                INC     CX              Sumar al cociente
                                JMP     COMP
                        OUT:
                                MOV     QUOTIENT,CX     ,-Almacenar cociente
                                ENDM

                                .MODEL  SMALL
                                .STACK  64
                                .DATA
0000  0096              DIVDND  DW      150             Dividendo
0002  001B              DIVSOR  DW      27              Divisor
0004  0000              QUOTNT  DW                      Cociente

                                .CODE
0000                    BEGIN   PROC    FAR
                                .LALL
                                INITZ
0000 B8 _____R      1           MOV     AX,@data        Inicializa
0003 8E D8          1           MOV     DS,AX             registro de
0005 8E   CO        1           MOV     ES.AX             segmento
                                DIVIDE  DIVDND,DIVSOR,QUOTNT
= 0000              1           CNTR    = 0
                    1           AX = divdo,  BX = dvsor,  CX = ente
0007 Al 0000 R      1           MOV     AX,DIVDND       Asigna dividendo
000A 8B 1E 0002 R   1           MOV     BX,DIVSOR       Asigna divisor
000E 2B C9          1           SUB     CX,CX           Pone en cero al cociente
0010                1 ??0000:
0010 3B C3          1           CMP     AX,BX           /¿Dividendo < divisor?
0012 72 05          1           JB      ??0001          ;  sí, salir
```

Figura 22-6 Uso de IF y de IFNDEF

```
0014  2B C3          1              SUB     AX.BX              /Dividendo -
0016  41             1              INC     CX                ,-Sumar al coc
0017  EB F7          1              JMP     ??0000
0019                 1  ??0001:
0019  89 OE 0004 R    1              MOV     QUOTNT,CX          ;Almacena coc
                                    DIVIDE  DIDND,DIVSOR,QUOT
= 0000               1              CNTR    = 0
                     1  ;           AX = divdo,  BX = dvsor,  CX = ente
                     1              IFNDEF  DIDND
                     1  /                   Dividendo no definido
= 0001               1              CNTR    = CNTR +1
                     1              ENDIF
                     1              IFNDEF  QUOT
                     1  ;                   Cociente no definido
= 0002               1              CNTR    = CNTR + 1
                     1              ENDIF
                     1              IF      CNTR
                     1  ;                   Expansión de macro terminada
                     1              EXITM
001D   B8 4C00                      MOV     AX,4C00H           /Salir al DOS
0020   CD 21                        INT     21H
0022                   BEGIN        ENDP
                                    END     BEGIN
```

Figura 22-6 (continuación)

### Macro que utiliza la condición IFIDN

La estructura del programa en la figura 22-7 contiene la definición de una macro llamada MOVIF que genera MOVSB o MOVSW, dependiendo del parámetro proporcionado. Un usuario tiene que codificar la macro instrucción con el parámetro B (byte) o W (palabra) para indicar si MOVS se convierte en MOVSB o en MOVSW.

Las primeras dos instrucciones de la definición de la macro son

```
MOVIF   MACRO    TAG

        IFIDN    <&TAG>,<B>
```

En la definición, el primer IFIDN genera REP MOVSB si usted codifica MOVIFB como una macroinstrucción. El segundo IFIDN genera REP MOVSW si usted codifica MOVIFW como una macroinstrucción. Si el usuario no proporciona B o W, el ensamblador genera un comentario y por omisión MOVSB. (El uso común del operador ampersán (&) es para concatenación.)

Los tres ejemplos en el segmento de código de MOVIF prueban la B, la W y una condición no válida. No intente ejecutar el programa como está, ya que los registros CX y DX necesitan cifras apropiadas para las instrucciones MOVS. (Esta macro no es muy útil, ya que su objetivo es ilustrar el uso de directivas condicionales de una manera sencilla. Sin embargo, ahora ya debe ser capaz de desarrollar macros significativas.)

## PUNTOS CLAVE

- Una definición de macro necesita una directiva MACRO, un bloque de una o más instrucciones, conocido como el cuerpo que la definición genera, y una directiva ENDM para terminar la definición.

```
                                page    60,132
                        TITLE   P22MACR7 (EXE)  Pruebas de IFIDN

                        INITZ   MACRO                   /Define macro
                                MOV     AX,@data
                                MOV     DS,AX
                                MOV     ES,AX
                                ENDM                    /Fin de la macro
                        MOVIF   MACRO   TAG             /Define macro
                                IFIDN   <&TAG>,<B>
                                REP MOVSB
                                EXITM
                                ENDIF
                                IFIDN   <&TAG>,<W>
                                REP MOVSW
                                ELSE
                        ;       Ni B ni W, por omisión B
                                REP MOVSB
                                ENDIF
                                ENDM                    /Fin de la macro

                                .MODEL  SMALL
                                .STACK  64
                                .CODE
0000                    BEGIN   PROC    FAR
                                .LALL
                                INITZ
0000 B8_____R       1           MOV     AX,@data
0003 8E D8          1           MOV     DS,AX
0005 8E C0          1           MOV     ES,AX
                                MOVIF   B
                    1           IFIDN   <B>,<B>
0007 F3/ A4         1           REP MOVSB
                    1           EXITM
                                MOVIF   W
                    1           IFIDN   <W>,<W>
0009 F3/ A5         1           REP MOVSW
                    1           ENDIF
                                MOVIF
                    1           ELSE
                    1   /       Ni B ni W, por omisión B
000B F3/ A4         1           REP MOVSB
                    1           ENDIF
000D B8 4C00                    MOV     AX,4C00H        /Sale al DOS
0010 CD 21                      INT     21H
0012                    BEGIN   ENDP
                                END     BEGIN
```

Figura 22-7 Uso de IFIDN

- Una instrucción de macro es el uso de la macro en un programa. El código que una instrucción genera es la expansión de la macro.
- Las directivas .SALL, .LALL y .XALL controlan el listado de comentarios y el código objeto generado en una expansión de macro.
- La directiva LOCAL facilita el uso de nombres dentro de una definición de macro y debe aparecer inmediatamente después del enunciado de la macro.
- El uso de argumentos mudos (ficticios) en una definición de macro permite a un usuario codificar parámetros con más flexibilidad.
- Una biblioteca de macros hace que estén disponibles para otros programas.
- Las directivas condicionales le permiten validar parámetros de la macro.

PREGUNTAS

22-1. ¿Bajo qué circunstancias recomendaría el uso de macros?

22-2. Codifique la primera y la última línea para una macro sencilla llamada SETUP.

22-3. Escriba las diferencias entre el cuerpo de una definición de macro y la expansión de la macro.

22-4. ¿Qué es un argumento mudo (ficticio)?

22-5. Codifique los siguientes enunciados: (a) Suprima todas las instrucciones que genera una macro; (b) liste sólo las instrucciones que generan código objeto.

22-6. Codifique dos definiciones de macro que realicen multiplicación: (a) MULTBY es para generar código que multiplique un byte por un byte; (b) MULTWD es para generar código que multiplique una palabra por una palabra. Incluya multiplicandos y multiplicadores como argumentos mudos en la definición de la macro. Pruebe la ejecución de las macros con un pequeño programa que también defina los campos de datos necesarios.

22-7. Almacene las macros definidas en la pregunta 22-6 en una biblioteca de macros. Corrija el programa para incluir (INCLUDE) las entradas de la biblioteca durante la pasada 1 del ensamblador.

22-8. Escriba una macro llamada BIPRINT que use la INT 17H del BIOS para imprimir. La macro debe incluir una prueba para el estado de la impresora y debe prever cualquier línea que se imprima con cualquier longitud.

22-9. Corrija la macro de la figura 22-6 de modo que pase por alto la división si el divisor es cero.

# CAPÍTULO 23

# Enlace de subprogramas

OBJETIVO

Estudiar las técnicas de programación implicadas en el enlace y ejecución de programas ensamblados por separado.

## INTRODUCCIÓN

Hasta este capítulo, los programas que hemos presentado han consistido en un solo módulo ensamblado y autónomo. Sin embargo, es posible desarrollar un programa que conste de un programa principal enlazado con uno o más subprogramas ensamblados por separado. Hay varias razones para organizar un programa en subprogramas:

- Enlazar entre lenguajes; por ejemplo, combinar la potencia del cómputo de un lenguaje de alto nivel con el procesamiento eficaz del lenguaje ensamblador.
- Facilitar el desarrollo de proyectos grandes, en los que diferentes equipos producen sus módulos por separado.
- Traslapar partes de un programa durante la ejecución a causa del gran tamaño del programa.

Cada programa es ensamblado por separado y genera su propio módulo de código objeto (.OBJ). Entonces, el enlazador enlaza los módulos objeto en un módulo ejecutable (.EXE). Habitualmente, el programa principal es el que inicia la ejecución y llama a uno o más subprogramas. Los subprogramas a su vez pueden llamar a otros subprogramas.

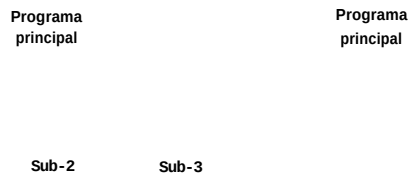



Figura 23-1 Jerarquía de programa

La figura 23-1 muestra dos ejemplos de una jerarquía de un programa principal y tres subprogramas. En la parte (a), el programa principal llama a los subprogramas 1, 2 y 3. En la parte (b), el programa principal llama a los subprogramas 1 y 2 y sólo el subprograma 1 llama al subprograma 3.

Existen numerosas formas de organizar subprogramas, pero la organización tiene que tener sentido para el ensamblador, para el enlazador y para la ejecución. También tiene que tener cuidado de situaciones en las que, por ejemplo, el subprograma 1 llama al subprograma 2, que llama al subprograma 3, quien a su vez llama al subprograma 1. Este proceso, conocido como recursión, puede hacerse funcionar, pero si no se maneja con cuidado, puede provocar interesantes errores de ejecución.

## SEGMENTOS

Esta sección cubre varias opciones utilizadas para los segmentos. El formato general para una directiva completa SEGMENT es

```
n_seg  SEGMENT  [alinear]  [combinar]  ['clase']
```

### Tipo align

El operador alinear (align) le indica al ensamblador que alinee el segmento nombrado al inicio de una frontera particular de almacenamiento:

| | |
|---|---|
| BYTE | Frontera de byte, para un segmento de un subprograma que será combinado con el de otro programa. La alineación de byte en general es más adecuada para programas que corren en un procesador 8088. |
| WORD | Frontera de palabra, para un segmento de un subprograma que será combinado con el de otro programa. La alineación de palabra por lo general es más adecuada para programas que corren en los procesadores 8086/80286. |
| DWORD | Frontera de palabra doble, normalmente para el 80386 y procesadores posteriores. |
| PARA | Frontera de párrafo (divisible ente 16 o 10H), el valor por omisión y el más comúnmente utilizado para alineación de programas principales y subprogramas. |
| PAGE | Frontera de página (divisible entre 256 o 100H). |

Si se omite el operador alinear del primer segmento causa que el valor por omisión se PARA. La omisión en segmentos subsecuentes hace que el valor por omisión se PARA, si el nombre es único; si el nombre no es único, el valor por omisión es el tipo de alineación del segmento previamente definido con el mismo nombre.

Tipo combine (combinar)

El operador combine (combinar) le indica al ensamblador y al enlazador si combina segmentos o los mantiene separados. Ya hemos usado el tipo combinar STACK. Otros tipos importantes para este capítulo son NONE, PUBLIC y COMMON:

| | |
|---|---|
| NONE | El segmento será separado de manera lógica de los otros segmentos, aunque se encuentren físicamente adyacentes. Éste es el tipo por omisión para directivas completas de segmento. |
| PUBLIC | El enlazador combina el segmento con todos los demás segmentos que están definidos como PUBLIC y tienen el mismo nombre de segmento y de clase. El ensamblador calcula los desplazamientos desde el inicio del primer segmento. De hecho, el segmento combinado contiene varias secciones, cada una iniciando con una directiva SEGMENT y finalizando con ENDS. Éste es el tipo por omisión para directivas simplificadas de segmentos. |
| COMMON | Si segmentos comunes (COMMON) tienen el mismo nombre y clase, el enlazador les da la misma dirección base. Durante la ejecución, el segundo segmento se traslapa en el primero. El segmento más grande determina la longitud del área común. |

Tipo class (clase)

Ya hemos usado los nombres de clase 'Stack', 'Data' y 'Code'. Se puede asignar el mismo nombre de clase a segmentos relacionados de modo que el ensamblador y el enlazador los agrupen. Esto es, aparecerán como segmentos uno después del otro, pero no combinados en un segmento a menos que también se codifique la opción combinar PUBLIC. La entrada clase puede contener cualquier nombre válido, contenido entre apóstrofos, aunque se recomienda el nombre 'Code' para el segmento del código.

Las dos instrucciones siguientes SEGMENT no relacionadas generan resultados idénticos, es decir, un segmento de código independiente alineado en una frontera de párrafo:

```
CODESEG   SEGMENT   PARA NONE  'Code'
CODESEG   SEGMENT   'Code'
```

En el capítulo 4 explicamos completamente las directivas de segmentos definidas, pero en los capítulos subsecuentes se usaron las directivas simplificadas de segmentos. Puesto que las directivas completas de segmento pueden proporcionar un control más estricto cuando se ensamblan o enlazan subprogramas, la mayoría de los ejemplos en este capítulo las utilizan.

Los ejemplos de programas en éste y en posteriores capítulos ilustran muchas de las opciones Align, Combine y Class.

## LLAMADAS INTRASEGMENTO

Las instrucciones CALL usadas hasta este momento han sido llamadas *intrasegmento;* esto es, el procedimiento llamado está en el mismo segmento de código que el procedimiento que llama. Una llamada (CALL) intrasegmento es *cercana* si el procedimiento llamado está definido o si es por omisión es NEAR (esto es, dentro de 32K). La operación CALL empuja el registro IP a la pila y

reemplaza el IP con el desplazamiento de la dirección destino. Por tanto una llamada cercana hace referencia a un procedimiento (cercano) que se encuentra en el mismo segmento.

Ahora considere una llamada (CALL) intrasegmento que consista del código objeto E8 2000, en donde E8 es el código de la operación y 2000 es el desplazamiento de un procedimiento llamado. La operación guarda el IP en la pila y almacena el 2000 como desplazamiento 0020 en el IP. Entonces el procesador combina la dirección actual en el CS con el desplazamiento en el IP para la siguiente instrucción a ejecutar. Al salir del procedimiento llamado, un RET (cercano) saca de la pila el IP almacenado y regresa a la siguiente instrucción después de CALL:

```
            CALL  proc cerc    ,•Llamada cercana: guarda en la pila
                               ; el IP, enlaza a proc cerc
proc cerc   PROC  NEAR

            RET                ,•Regreso cercano: saca IP y regresa
proc cerc   ENDP
```

Una llamada intrasegmento puede ser cercana, como se describió, o lejana si la llamada es a un procedimiento definido como lejano dentro del mismo segmento. RET es cercano si aparece en un procedimiento NEAR y lejano si aparece en un procedimiento FAR.

## LLAMADAS INTERSEGMENTO

Una llamada (CALL) es clasificada como *lejana* si el procedimiento llamado está definido como FAR o como EXTRN, con frecuencia en otro segmento. La operación CALL primero guarda en la pila el contenido del registro CS e inserta una nueva dirección de segmento en el CS. Después guarda en la pila el IP e inserta un nuevo desplazamiento de dirección en el IP. (Los datos guardados en el CS e IP proporcionan la dirección de la instrucción que sigue de forma inmediata a CALL.) De esta manera, ambas direcciones del segmento de código y el desplazamiento son guardados para regresar del procedimiento llamado. Una llamada a otro procedimiento siempre es una llamada a un *intersegmento* lejano:

```
            CALL   proc_lej   ;Llamada lejana: guarda en la pila
                              ; el CS, el IP enlaza a proc lej
proc__lej   PROC   NEAR

            RET               ,•Regreso lejano: remueve IP, CS y
proc lej    ENDP              ; regresa
```

Considere una llamada (CALL) intersegmento que consta del código objeto 9A 0002 AF04. El 9A hex es el código de la operación para un CALL intersegmento. La operación guarda en la pila el

```
          EXTRN   SUBPROG:FAR
MAINPROG  PROC    FAR

          CALL    SUBPROG

MAINPROG  ENDP

          PUBLIC  SUBPROG
SUBPROG   PROC    FAR


          RET
SUBPROG   ENDP
```

Figura 23-2 Llamada (CALL) intersegmento

IP actual y almacena el nuevo desplazamiento 0002 como 0200 en el IP. Después guarda en la pila el CS y almacena la nueva dirección de segmento AF04 como 04AF en CS. Los números en el CS e IP se combinan para establecer la dirección de la primera instrucción a ejecutar en el subprograma llamado:

| | |
|---|---|
| Segmento de código: | 04AF0H |
| Desplazamiento en IP: | + 02OOH |
| Dirección efectiva: | 04CF0H |

Al salir del procedimiento llamado, un RET intersegmento (lejano) revierte la operación CALL removiendo de la pila las direcciones originales IP y CS y enviándolas a sus respectivos registros. La pareja CS:IP ahora apunta a la dirección de la siguiente instrucción después del CALL original, en donde la ejecución se reasume.

La diferencia entre un CALL cercano y uno lejano es básicamente que un CALL cercano sólo reemplaza el desplazamiento IP, mientras que un CALL lejano reemplaza tanto la dirección del segmento CS como del desplazamiento IP.

## ATRIBUTOS EXTRN Y PUBLIC

Observe la figura 23-2, en la que el programa principal (MAINPROG) llama a un subprograma (SUBPROG). El requisito aquí es de una llamada (CALL) intersegmento.

El CALL en MAINPROG tiene que saber qué SUBPROG hay fuera de MAINPROG (en caso contrario el ensamblador genera un mensaje de error de que SUBPROG es un símbolo no definido). La directiva EXTRN SUBPROG:FAR le notifica al ensamblador que cualquier referencia a SUBPROG es a una etiqueta FAR que en este caso está definida de forma externa, en otro ensamblado. Puesto que el ensamblador no tiene manera de saber la dirección a la hora de la ejecución, genera operandos con código objeto "empty" ("vacío") en CALL lejano (ceros para el desplazamiento y guiones para el segmento) que el enlazador posteriormente llena:

```
9A 0000_____E      ;CALL (llamada) al subprograma
```

SUBPROG a su vez contiene una directiva PUBLIC que le indica al ensamblador y al enlazador que otro módulo debe conocer la dirección de SUBPROG. En un paso posterior, cuando MAINPROG y SUBPROG sean ensamblados con éxito en módulos objetos, pueden ser enlazados como sigue:

```
El enlazador solicita        Respuesta
Object Modules [.OBJ]:       D:MAINPROG+D:SUBPROG
Run File [filespec.EXE]      D:COMBPROG (o cualquier nombre válido)
List File [NUL.MAP]:         CON
Libraries [.LIB]:            [Enter]
```

El enlazador hace corresponder los EXTRN en un módulo objeto con los PUBLIC en el otro e inserta las direcciones de desplazamiento requeridas. Después combina los dos módulos objetos en un módulo ejecutable. Si es incapaz de establecer la correspondencia entre las referencias, el enlazador envía mensajes de error; espérelos antes de intentar ejecutar el módulo.

### La directiva EXTRN

La directiva EXTRN indica al ensamblador que el elemento llamado un dato —procedimiento o etiqueta— está definido en otro ensamblado. (MASM 6.0 introdujo el término EXTERN.) EXTRN tiene el formato siguiente:

```
EXTRN nombre:tipo [, ...]
```

Puede definir más de un nombre, hasta el final de la línea, o bien codificar instrucciones adicionales EXTRN. El otro módulo ensamblado a su vez debe definir el nombre e identificarlo como PUBLIC. La entrada tipo puede ser ABS (una constante), BYTE, DWORD, FAR, NEAR, WORD o un nombre definido por un EQU, y debe ser válido en términos de la definición real de un nombre;

- BYTE, WORD y DWORD identifican datos a los que hace referencia un módulo, pero otro módulo los define.
- NEAR y FAR identifican a un procedimiento o etiqueta de instrucción a los que hace referencia un módulo pero otro módulo los define.

### La directiva PUBLIC

La directiva PUBLIC indica al ensamblador y al enlazador que la dirección de un símbolo especificado definido en el ensamblado actual estará disponible para otros módulos. El formato general para PUBLIC es

```
PUBLIC símbolo [ ...]
```

Puede definir más de un símbolo, hasta el final de la línea, o bien codificar instrucciones PUBLIC adicionales. La entrada símbolo puede ser una etiqueta (incluyendo etiquetas PROC), una variable o un número. Entradas no válidas incluyen nombres de registros y símbolos EQU que definen valores mayores de dos bytes.

La llamada de procedimientos lejanos y el uso de EXTRN y PUBLIC ofrecería un poco de dificultad, aunque se requiere de mucho cuidado para crear datos definidos en un módulo conocido en otros módulos.

Examinemos tres diferentes tipos de crear datos conocidos entre programas: por medio de EXTRN y PUBLIC, definición de datos en subprogramas y paso de parámetros.

## USO DE EXTRN Y PUBLIC PARA UNA ETIQUETA

El programa de la figura 23-3 consiste en un programa principal, P23MAIN1, y un subprograma, P23SUB1; ambos utilizan directivas completas de segmento. El programa principal define segmentos para la pila, los datos y el código. El segmento de datos define QTY y PRICE. El segmento de código carga el AX con PRICE y el BX con QTY y después llama al subprograma. Un EXTRN en el programa principal define el punto de entrada al subprograma como P23SUB1.

El subprograma contiene una instrucción PUBLIC (después de ASSUME) que hace a P23SUB1 conocida para el enlazador como el punto de entrada para la ejecución. Este subprograma sólo multiplica el contenido del AX (precio) por el BX (cantidad) y desarrolla el producto en la pareja DX:AX como 002E 4000H.

Puesto que el subprograma no define datos, no necesita un segmento de datos; podría hacerlo, pero sólo el mismo subprograma reconocería los datos.

También el subprograma no define un segmento de la pila, ya que referencia a la misma pila de direcciones que el programa principal. En consecuencia, la pila definida en el programa principal está disponible para el subprograma. Con el enlazador requiere la definición de al menos una pila para un programa .EXE, la pila en el programa principal sirve para este propósito.

Examinemos ahora la tabla de símbolos después de cada ensamblado. Observe que la tabla de símbolos para el programa principal muestra P23SUB1 como lejano (Far) y externo (External). La tabla de símbolos para el subprograma muestra P23SUB1 como F (Far, lejano) y Global. El término *global* implica que el nombre es conocido en otros subprogramas fuera de P23SUB1.

El mapa de enlace al final del listado muestra la organización del programa en la memoria. Observe que existen dos segmentos de código, uno en cada ensamblado, pero en diferentes direcciones de inicio, ya que sus tipos combinar son NONE. Éstos aparecen en la secuencia en que usted los ingresó cuando los enlazó, por lo común primero el programa principal. En este ejemplo, el segmento de código para el programa principal inicia en el desplazamiento 00090H y el segmento de código para el subprograma en OOOBOH.

Un rastreo de la ejecución del programa reveló que el registro CS para P23MAIN1 contenía 0F20[0] y la instrucción CALL P23SUB1 generada

```
9A 0000 220F (el valor de su segmento puede ser diferente)
```

El código de máquina para un CALL intersegmento es 9AH. La operación guarda en la pila el registro IP y carga 0000 en el IP. Después guarda en la pila el CS que contiene 0F20[0] y carga 0F22[0] (desde el operando CALL) en el CS. (Mostraremos el contenido de los registros en orden normal de bytes, no en orden inverso de bytes.)

La siguiente instrucción a ejecutarse es CS:IP, o 0F22[0] más 0000. ¿Qué está en 0F220? Está el punto de entrada a P23SUB1 en su primer instrucción ejecutable, la cual puede calcular. El programa principal inicia con el registro CS que contiene 0F20[0]. De acuerdo con el mapa, el desplazamiento del segmento principal de código inicia en el desplazamiento 00090H y el desplazamiento del subprograma inicia en el desplazamiento OOOBOH, a 20H bytes de distancia. Sumando 20H al CS del programa principal proporciona la dirección efectiva del segmento de código del subprograma:

```
                        TITLE     P23MAIN1 (EXE)  Llama al subprograma
                                  EXTRN   P23SUB1:FAR

0000                    STACKSG   SEGMENT PARA STACK 'Stack'
0000 0040[????]                   DW      64 DUP(?)
0080                    STACKSG   ENDS

0000                    DATASG    SEGMENT PARA 'Data'
0000 0140               QTY       DW      0140H
0002 2500               PRICE     DW      2500H
0004                    DATASG    ENDS

0000                    CODESG    SEGMENT PARA 'Code'
0000                    BEGIN     PROC    FAR
                                  ASSUME  CS:CODESG,DS:DATASG,SS:STACKSG
0000 B8 ____ R                    MOV     AX,DATASG
0003 8E D8                        MOV     DS,AX
0005 Al 0002 R                    MOV     AX,PRICE      ;Configura precio y
0008 8B 1E 0000 R                 MOV     BX,QTY        ;  cantidad
000C 9A 0000 ____ E               CALL    P23SUB}.      ;Llama al subprograma
0011 B8 4C00                      MOV     AX,4C00H      ;Sale al DOS
0014 CD 21                        INT     21H
0016                    BEGIN     ENDP
0016                    CODESG    ENDS
                                  END     BEGIN

Segmentos y grupos
            Ñ a m e         Length    Align    Combine    Class
CODESG. . . . . .           .0016     PARA     NONE       'CODE'
DATASG. . . . . .           .0004     PARA     NONE       'DATA'
STACKSG. . . . . .          .0080     PARA     STACK      'STACK'
Symbols:
            Ñ a m e          Type     Valué       Attr
BEGIN. . . . . .            .F PROC   0000     CODESG      Length = 0016
P23SUB1. . . . . .          .L FAR    0000     External
PRICE. . . . . .            .L WORD   0002     DATASG
QTY. . . . . .              .L WORD   0000     DATASG


                        TITLE     P23SUB1  Subprograma  llamado

0000                    CODESG    SEGMENT PARA 'Code'
0000                    P23SUB1   PROC    FAR
                                  ASSUME  CS:CODESG
                                  PUBLIC  P23SUB1
0000 F7 E3                        MUL     BX        ;AX = precio,  BX = cant.
0002 CB                           RET               ;DX:AX = producto
0003                    P23SUB1   ENDP
0003                     CODESG   ENDS
                                  END     P23SUB1

Segmentos y grupos
            Ñ a m e         Length    Align    Combine    Class
CODESG. . . . . .           .0003     PARA     NONE         'CODE'
Symbols:
            Ñ a m e         Type      Valué     Attr
P23SUB1. . . . . .          .F PROC    0000     CODESG  Global Length=0003


    Link Map
    Object Modules:  P23MAIN1+P23SUB1

    Start  Stop   Length Ñame                 Class
    OOOOOH 0007FH 00080H STACKSG              STACK
```

Figura 23-3 Uso de EXTRN y PUBLIC

```
00080H 00083H 00004H DATASG              DATA
00090H 000A5H 00016H CODESG              CODE <--  Nota: 2 segmentos
OOOBOH 000B2H 00003H CODESG              CODE <--        „  código

Program entry point at 0009:0000
```

Figura 23-3 (continuación)

Dirección CS para P23MAIN1: 0F200H
Tamaño de P23MAIN1: +00020H
Dirección CS para P23SUB1: 0F220H

El cargador de programa determina esta dirección al igual que nosotros y la sustituye en el operando de CALL. P23SUB1 multiplica los dos números en el AX y BX, con el producto en el DX:AX, y realiza un regreso lejano a P23MAIN1 (porque RET está en un procedimiento FAR).

## USO DE PUBLIC EN EL SEGMENTO DE CÓDIGO

La figura 23-4 muestra una variación de la figura 23-3. Hay un cambio en el programa principal, P23MAIN2, y un cambio en el subprograma, P23SUB2, los dos con el uso de PUBLIC en la directiva SEGMENT para ambos segmentos de código:

```
CODESG SEGMENT PARA PUBLIC 'Code'
```

En el mapa de enlace y en el código objeto de CALL aparecen resultados interesantes. En la tabla de símbolos, el tipo combinar CODESG es PUBLIC, mientras que en la figura 23-3 era NONE. También, el mapa de enlace al final muestra ahora un solo segmento de código. El hecho de que ambos segmentos tienen el mismo nombre (DATASG), clase ('Code') y atributo PUBLIC hizo que el enlazador combinara los dos segmentos lógicos de código en un segmento físico de código. Además, un rastreo de la ejecución de máquina mostró que el CALL es lejano, porque es a un procedimiento FAR; esto es, aunque la llamada es dentro del mismo segmento:

```
9A 2000 200F (la dirección de su segmento puede ser diferente)
```

Este CALL lejano almacena 2000H en el IP como 0020H y 200FH en el registro CS como 0F20[0]. Como el subprograma comparte un segmento de código común con el programa principal, el registro CS se establece con la misma dirección de inicio, 0F20H. Pero el CS:IP para P23SUB2 ahora proporciona lo siguiente:

Dirección CS para P23MAIN2 y P23SUB2: 0F200H
Desplazamiento IP para P23SUB2: + 0020H
Dirección efectiva de P23SUB2: 0F22OH

Por lo tanto, el segmento de código del subprograma presumiblemente inicia en 0F220H. ¿Es esto correcto? El mapa del enlace no deja claro este punto, pero puede inferir la dirección del listado del programa principal, que termina en el desplazamiento 0015H. (El mapa muestra 16H, que es la siguiente localidad disponible.) Ya que el segmento de código para el subprograma está definí-

```
                        TITLE     P23MAIN2 (EXE)   Llama al subprograma
                                  EXTRN   P23SUB2:FAR

0000                    STACKSG   SEGMENT PARA STACK 'Stack'
0000  0040 [????]                 DW      64 DUP(?)
0080                    STACKSG   ENDS

0000                    DATASG    SEGMENT  PARA  Data'
0000  0140              QTY       DW      0140H
0002  2500              PRICE     DW      2500H
0004                    DATASG    ENDS

0000                    CODESG    SEGMENT PARA PUBLIC  'Code'
0000                    BEGIN     PROC    FAR
                                  ASSUME  CS:CODESG,DS:DATASG,SS:STACKSG
0000 B8 _____R                    MOV     AX, DATASG
0003 8E D8                        MOV     DS, AX
0005 Al 0002 R                    MOV     AX,PRICE        Configura precio y
0008 8B 1E 0000 R                 MOV     BX,QTY            cantidad
OOOC 9A 0000 --    E              CALL    P23SUB2         Llama al subprograma
0011 B8 4C00                      MOV     AX,4C00H        Sale al DOS
0014 CD 21                        INT     21H
0016                    BEGIN     ENDP
0016                    CODESG    ENDS
                                  END     BEGIN

Segmentos y grupos
        Ñ a m e             Length   Align    Combine    Class
CODESG. . . . . .           .0016    PARA     PUBLIC     'CODE'
DATASG. . . . . .           .0004    PARA     NONE       'DATA'
STACKSG. . . . . .          .0080    PARA     STACK      'STACK'
Symbols:
        Ñ a m e             Type     Valué    Attr
BEGIN. . . . . .            .F PROC  0000     CODESG      Length = 0016
P23SUB2. . . . . .          .L FAR   0000     External
PRICE. . . . . .            .L WORD  0002     DATASG
QTY. . . . . .              .L WORD  0000     DATASG


                        TITLE     P23SUB2  Subprograma llamado

0000                    CODESG    SEGMENT PARA PUBLIC  'Code'
0000                    P235UB2   PROC    FAR
                                  ASSUME  CS:CODESG
                                  PUBLIC  P23SUB2
0000 F7 E3                        MUL     BX       ;AX = precio,  BX = cant.
0002 CB                           RET              ;DX:AX = producto
0003                    P23SUB2   ENDP
0003                    CODESG    ENDS
                                  END     P23SUB2
Segmentos y grupos
       Ñ a m e              Length   Align    Combine    Class
CODESG. . . . . .           .0003    PARA     PUBLIC     'CODE'
Symbols:
       Ñ a m e              Type     Valué     Attr
P23SUB2. . . . . .          .F PROC  0000     CODESG  Global Length=0003


   Link Map
   Object Modules:  P23MAIN2+P23SUB2
```

Figura 23-4 Segmento de código definido como PUBLIC

```
Code segment
```

Figura 23-4 (continuación)

do como PARA, inicia en una frontera de párrafo (divisible exactamente entre 10H, de modo que el dígito de más a la derecha es 0):

```
programa principal  ...  (no usado)  subprograma

 I                       I             I
0F200                   141F0         0F220
```

El enlazador coloca al subprograma en la primera frontera de párrafo posterior al programa principal, en el desplazamiento 00020H. Por lo tanto, el segmento de código del subprograma inicia en 0F200H más 0020H, o 0F220H.

Examinemos ahora este mismo programa definido con directivas simplificadas de segmento.

## DIRECTIVAS SIMPLIFICADAS DE SEGMENTO

La figura 23-5 muestra el programa anterior ahora definido con directivas simplificadas de segmento. La figura 23-4 define los segmentos de código como PUBLIC, mientras que la figura 23-5 por omisión lo hace PUBLIC, de modo que ambos ejemplos generan un segmento de código. Sin embargo, el uso de directivas simplificadas de segmento origina algunas diferencias importantes. Primera, el enlazador ha reacomodado los segmentos (como está mostrado en el mapa) en secuencia de código, datos y pila, aunque no tiene efecto sobre la ejecución del programa. Segunda, el segmento de código del subprograma (TEXT) se alinea a una frontera de palabra (en lugar de una de párrafo). Un rastreo de la ejecución mostró el siguiente código objeto para CALL:

```
9A 1600 170F (la dirección de su segmento puede ser diferente)
```

Esta vez, el nuevo valor de desplazamiento es 16H y la dirección del segmento es 0F17H. Puesto que el subprograma comparte un segmento de código común con el programa principal, el registro CS se establece con la misma dirección de inicio, 0F17(0), para ambos. Por tanto, la dirección de P23SUB3 puede ser calculada como sigue:

| | |
|---|---|
| Dirección CS para P23MAIN3 y P23SUB3: | **F170H** |
| Desplazamiento IP para P23SUB3: | **+ 016H** |
| Dirección efectiva de P23SUB3: | **F186H** |

Puede inferir la dirección a partir del listado del programa principal, que termina en el desplazamiento 0015H. (El mapa revela 16H, que es la siguiente localidad disponible.) Como el mapa muestra el segmento de código principal que inicia en 00000H, la siguiente frontera de palabra después de 0015 está en 00016H, en donde P23SUB3 inicia.

```
                        TITLE     P23MAIN3 (EXE)   Llama al subprograma
                                  .MODEL   SMALL
                                  .STACK   64
                                  EXTRN    P23SUB3:FAR

                                  .DATA
0000 0140               QTY       DW       0140H
0002 2500               PRICE     DW       2500H

                                  .CODE
0000                    BEGIN     PROC     FAR
0000 B8 ------ R                  MOV      AX,@data
0003 8E D8                        MOV      DS,AX
0005 Al 0002 R                    MOV      AX,PRICE      /Configura precio y
0008 8B 1E 0000 R                 MOV      BX,QTY        /  cantidad
000C 9A 0000 ------ E             CALL     P23SUB3       /Llama al subprograma
0011 B8 4C00                      MOV      AX,4C00H      /Sale al DOS
0014 CD 21                        INT      21H
0016                    BEGIN     ENDP
                                  END      BEGIN

Segmentos y grupos
        Ñ a m e                 Length   Align    Combine    Class
DGROUP. . . . . .               .GROUP
 _DATA. . . . . .               .0004    WORD     PUBLIC     'DATA'
 STACK. . . . . .               .0040    PARA     STACK      'STACK'
_TEXT. . . . . .                .0016    WORD     PUBLIC     'CODE'
Symbols:
        Ñ a m e                 Type     Valué    Attr
BEGIN. . . . . .                .F PROC  0000     _TEXT      Length = 0016
P23SUB3. . . . . .              .L FAR   0000       External
PRICE. . . . . .                .L WORD  0002     _DATA
QTY. . . . . .                  .L WORD  0000     _DATA


                        TITLE     P23SUB3 Subprograma llamado
                                  -MODEL SMALL
                                  .CODE
000®                    P23SUB3   PROC     FAR
                                  PUBLIC   P23SUB3
0000 F7 E3                        MUL      BX        ;AX = precio, BX = cant.
0002 CB                           RET                ;DX:AX = producto
0003                    P23SUB3   ENDP
                                  END      P23SUB3
Segmentos y grupos
        Ñ a m e                 Length   Align    Combine    Class
DGROUP. . . . . .               .GROUP
 _DATA. . . . . .               .0000    WORD     PUBLIC     'DATA'
_TEXT . . . . . .               .0003    WORD     PUBLIC     'CODE'
Symbols:
        Ñ a m e                 Type      Valué   Attr
P23SUB3. . . . . .              .F PROC   0000    _TEXT  Global  Length=0003


    Link Map
    Object Modules:  P23MAIN3+P23SUB3



    Program entry point at 0000:0000
```

Figura 23-5 Uso de directivas simplificadas de segmento

## DATOS COMUNES EN SUBPROGRAMAS

Un requerimiento común en programación es procesar en un módulo datos que están definidos en otro módulo. Modifiquemos el ejemplo anterior de manera que, aunque el programa principal aún define QTY y PRICE, el subprograma (en lugar del programa principal) inserta sus valores en el BX y AX. La figura 23-6 da el código revisado, con los cambios siguientes:

- El programa principal, P23MAIN4, define QTY y PRICE como PUBLIC. El segmento de datos también es definido con el atributo de PUBLIC. Observe en la tabla de símbolos el atributo global para QTY y PRICE.
- El subprograma, P23SUB4, define QTY y PRICE como EXTRN y como WORD. Esta definición informa al ensamblador de la longitud de los dos campos. El ensamblador puede generar el código correcto de operación para las instrucciones MOV, pero el enlazador tendrá que completar los operandos. (Observe en la tabla de símbolos que PRICE y QTY ahora son de clase externa.)

  El ensamblador lista las instrucciones MOV en el subprograma como

```
Al 0000 E      MOV       AX, PRICE
8B 1E 0000 E   MOV       BX, QTY
```

El código objeto Al significa mover una palabra desde la memoria hacia el AX, mientras que. 8B significa mover una palabra desde la memoria hacia el BX. (Con frecuencia, las operaciones con el AX requieren de menos bytes.) Para P23SUB4, el ensamblador no tiene manera de conocer las localidades de QTY y PRICE, de modo que almacenó ceros en los operandos para ambos MOV. El rastreo de la ejecución del programa revela que el enlazador ha completado el código objeto de los operandos como sigue:

```
Al   0200
8B   1E 0000
```

El código objeto ahora es idéntico al generado por los tres programas precedentes en donde las instrucciones MOV están en el programa llamado. Este es un resultado lógico, ya que los operandos en los tres programas hacen referencia a la misma dirección del segmento de datos en el registro DS y a los mismos valores de desplazamiento.

El programa principal y el subprograma puede definir otros elementos de datos, pero sólo aquellos definidos como PUBLIC y EXTRN son conocidos en común por ellos.

## DEFINICIÓN DE DATOS EN AMBOS PROGRAMAS

En el ejemplo anterior, P23MAIN4 definió QTY y PRICE, mientras que P23SUB4 no definió ningún dato. La razón de que P23SUB4 pueda hacer referencia a los datos de P23MAIN4 es que ha conservado la dirección del segmento de datos en el registro DS, el cual aún apunta al segmento de datos de P23MAIN4. (La única dirección de segmento cambiada fue la del segmento de código.) Pero los programas no siempre son tan sencillos, y los subprogramas con frecuencia tienen que definir sus propios datos, así como hacer referencia a datos en del programa que los llama.

```
                        TITLE     P23MAIN4 (EXE)   Llama al subprograma
                                  EXTRN    P23SUB4:FAR
                                  PUBLIC   QTY,PRICE

0000                    STACKSG   SEGMENT PARA STACK 'Stack
0000 0040[????]                   DW      64 DUP(?)
0080                    STACKSG   ENDS

0000                    DATASG    SEGMENT PARA PUBLIC 'Data'
0000 0140               QTY       DW      0140H
0002 2500               PRICE     DW      2500H
0004                    DATASG    ENDS

0000                    CODESG    SEGMENT PARA PUBLIC 'Code'
0000                    BEGIN     PROC    FAR
                                  ASSUME  CS:CODESG,DS:DATASG,SS:STACKSG
0000 B8 ---------R                MOV     AX,DATASG
0003 8E D8                        MOV     DS,AX
0005 9A 0000 ---------E           CALL    P23SUB4       ;Llama a subprograma
000A B8 4C00                      MOV     AX,4C00H      ;Sale al DOS
000D CD 21                        INT     21H
OOOF                    BEGIN     ENDP
OOOF                    CODESG    ENDS
                                  END     BEGIN

Segmentos y grupos
        Ñ a m e             Length   Align    Combine    Class
CODESG. . . . . .           .OOOF    PARA     PUBLIC    'CODE'
DATASG. . . . . .           .0004    PARA     PUBLIC    'DATA'
STACKSG. . . . . .          .0080    PARA     STACK     'STACK'
Symbols:
        Ñ a m e             Type     Valué     Attr
BEGIN. . . . . .            .F PROC  0000     CODESG     Length = OOOF
P23SUB4. . . . . .          .L FAR   0000     External
PRICE. . . . . .            .L WORD  0002     DATASG     Global
QTY. . . . . .              .L WORD  0000     DATASG     Global


                        TITLE     P23SUB4 Subprograma llamado
                                  EXTRN   QTY:WORD,  PRICE:WORD

0000                    CODESG    SEGMENT PARA PUBLIC 'CODE'
0000                    P23SUB4   PROC    FAR
                                  ASSUME  CS:CODESG
                                  PUBLIC  P23SUB4
0000 Al 0000 E                    MOV     AX,PRICE
0003 8B 1E 0000 E                 MOV     BX,QTY
0007 F7 E3                        MUL     BX            ;DX:AX = producto
0009 CB                           RET
000A                    P23SUB4   ENDP
000A                    CODESG    ENDS
                                  END     P23SUB4

Segmentos y grupos
        Ñ a m e             Length   Align    Combine    Class
CODESG. . . . . .           .000A    PARA     PUBLIC     'CODE'
Symbols:
        Ñ a m e             Type     Valué     Attr
P23SUB4. . . . . .          .F PROC   0000     CODESG  Global Length=000A
PRICE. . . . . .            .V WORD   0000     External
QTY. . . . . .              .V WORD   0000     External
```

Figura 23-6 Datos comunes en subprogramas

```
Link Map
Object Modules:  P23MAIN4+P23SUB4

Start   Stop    Length Ñame                    Class
00000H  0007FH  00080H STACKSG                STACK
00080H  00083H  00004H DATASG                 DATA
00090H  000A9H  0001AH CODESG                 CODE

Program entry point at 0009:0000
```

Figura 23-6 (continuación)

En una variación del programa precedente, en la figura 23-7 define QTY en P23MAN5, pero define PRICE en P23SUB5. En P23MAIN5 PRICE no existe, aunque P23SUB5 tiene que conocer la localidad de ambos elementos. El segmento de código de P23SUB5 tiene que recuperar QTY de forma inmediata, mientras que el registro DS aún contiene la dirección del segmento de datos de P23MAIN5. Después P23SUB5 guarda en la pila el DS y lo carga con la dirección de su propio segmento de datos. P23SUB5 ahora puede obtener PRICE y realizar la multiplicación de QTY por PRICE.

Antes de regresar a P23MAIN5, P23SUB5 tiene que sacar el DS de la pila de modo que P23MAIN5 pueda accesar su propio segmento de datos. (Técnicamente, esto en realidad no es necesario en este ejemplo, ya que P23MAIN5 regresa al DOS de manera inmediata, pero lo haremos como una práctica estándar.)

Una nota final: podría hacer ambos segmentos de datos PUBLIC, con el mismo nombre y clase. En ese caso, el enlazador los combinaría, y P23SUB5 no tendría que guardar y sacar de la pila el DS, ya que los programas usarían en mismo segmento de datos y la misma dirección de DS. Dejaremos esta variante como ejercicio para que use corrija y rastree con DEBUG. El segmento de código de P23SUB5 se vería así:

```
EXTRN   QTY:WORD
ASSUME  CS:CODESEG,DS:DATASG
PUBLIC  P23SUB5
MOV     AX,PRICE        ;PRICE en su propio segmento de datos
MOV     BX, QTY         ; QTY en P23MAIN5
MUL     BX              /Producto en DX:AX
RET
```

## PASO DE PARÁMETROS

Otra forma de hacer que se conozcan los datos por los subprogramas llamados es por medio del *paso de parámetros,* en el que un programa pasa datos físicamente mediante la pila. En este caso, asegúrese de que cada PUSH hace referencia a una palabra (o una palabra doble en sistemas avanzados), ya sea en memoria o en un registro.

### Pila de la estructura del programa

La pila de la estructura del programa es la parte de la pila que el programa que llama utiliza para pasar parámetros y que el subprograma llamado utiliza para accesarlos. El subprograma llamado

```
                        TITLE     P23MAIN5 (EXE)   Llama al subprograma
                                  EXTRN   P23SUB5:FAR
                                  PUBLIC  QTY

0000                    STACKSG   SEGMENT PARA STACK  'Stack'
0000 0040 [????]                  DW      64 DUP(?)
0080                    STACKSG   ENDS

0000                    DATASG    SEGMENT PARA  'Data'
0000 0140               QTY       DW      0140H
0002                    DATASG    ENDS

0000                    CODESG    SEGMENT PARA  'Code'
0000                    BEGIN     PROC    FAR
                                  ASSUME  CS:CODESG,DS:DATASG,SS:STACKSG
0000 B8 ____R                     MOV     AX, DATASG
0003 8E D8                        MOV     DS,AX
0005 9A 0000 ____E                CALL    P23SUB5     ;Llama al subprograma
000A B8 4C00                      MOV     AX,4C00H    ;Salir al DOS
000D CD 21                        INT     21H
000F                    BEGIN     ENDP
000F                    CODESG    ENDS
                                  END     BEGIN

Segmentos de grupos
        Ñ a m e             Length    Align    Combine    Class
CODESG. . . . . .           .000F     PARA     NONE       'CODE'
DATASG. . . . . .           .0002     PARA     NONE       'DATA'
STACKSG. . . . . .          .0080     PARA     STACK      'STACK'
Symbols:
        Ñ a m e             Type      Valué     Attr
BEGIN. . . . . .            .F PROC   0000      CODESG     Length = OOOF
P23SUB5. . . . . .          .L FAR    0000      External
QTY. . . . . .              .L WORD   0000      DATASG     Global


                        TITLE     P23SUB5  Subprograma  llamado
                                  EXTRN    QTY:WORD

0000                    DATASG    SEGMENT PARA  'Data'
0000 2500               PRICE     DW       2500H
0002                    DATASG    ENDS

0000                    CODESG    SEGMENT PARA 'CODE'
0000                    P23SUB5   PROC    FAR
                                  ASSUME  CS:CODESG
                                  PUBLIC  P23SUB5
0000 8B 1E 0000 E                 MOV     BX,QTY      ;Obtiene  QTY  desde  CALLMUL
0004 1E                           PUSH    DS          /Guarda el DS de CALLMUL
                                  ASSUME  DS:DATASG
0005 B8 ____R                     MOV     AX,DATASG   ;Configura su propio DS:
0008 8E D8                        MOV     DS,AX       ;Precio  desde
000A Al 0000 R                    MOV     AX,PRICE    ;  su propio segmento de datos
000D F7 E3                        MUL     BX          ;DX:AX = producto
OOOF IF                           POP     DS          /Restaura el DS de CALLMUL
0010 CB                           RET
0011                    P23SUB5   ENDP
0011                    CODESG    ENDS
                                  END     P23SUB5

Segmentos de grupos
        Ñ a m e             Length    Align    Combine    Class
CODESG. . . . . .           .0011     PARA     NONE       'CODE'
DATASG. . . . . .           .0002     PARA     NONE       'DATA'
```

Figura 23-7 Definición de datos en ambos programas

```
Symbols:
        Ñ a m e                 Type     Valué     Attr
P23SUB5. . . . . .              F PROC   0000      CODESG   Global  Length=0011
PRICE. . . . . .                L WORD   0000      DATASG
QTY. . . . . .                  V WORD   0000      External


    Link Map
    Object Modules:  P23MAIN5+P23SUB5

    Start  Stop    Length Ñame                Class
    00000H 0007FH 00080H STACKSG              STACK
    00080H 00081H 00002H DATASG               DATA
    00090H 00091H 00002H DATASG               DATA
    00OA0H 000AEH 0000FH CODESG               CODE
    OOOBOH 000C0H 00011H CODESG               CODE

    Program entry point at  OOOA:0000
```

Figura 23-7 (continuación)

también puede utilizar la pila de la estructura del programa para almacenamiento temporal de datos locales. El registro BP actúa como un apuntador a la estructura. Para el paso de parámetros haremos uso de ambos registros, el BP y el SP.

En la figura 23-8, el programa que llama P23MAIN6 guarda en la pila tanto PRICE como QTY antes de llamar al subprograma P23SUB6. Inicialmente, el SP contiene el tamaño de la pila, 80H. Cada palabra que se guarda en la pila disminuye en dos el SP. Después del CALL, la pila de la estructura aparece como sigue:

| | 1200 | 200F | 4001 | 0025 |
|---|---|---|---|---|
| | 78 | 7A | 7C | 7E |

1. Un PUSH cargó PRICE (2500H) en la pila de la estructura en el desplazamiento 7EH.
2. Un PUSH cargó QTY (0140H) en la pila de la estructura en el desplazamiento 7CH.
3. CALL guardó en la pila de la estructura el contenido del CS (0F20H para esta ejecución) en 7AH. Como el subprograma es PUBLIC, el enlazador combina los dos segmentos de código y la dirección CS es la misma para ambos.
4. También CALL guardó en la pila de la estructura el contenido del registro IP en 78H.

El programa llamado requiere el uso del BP para accesar los parámetros en la pila de la estructura. Su primer acción es guardar el contenido del BP para el programa que llama, de modo que guarda en la pila el BP. En este ejemplo, el BP contiene cero, que PUSH almacena en la pila en el desplazamiento 76H:

| 0000 | 1200 | 200F | 4001 | 0025 |
|---|---|---|---|---|
| 76 | 78 | 7A | 7C | 7E |

Después, el programa inserta el contenido del SP (0076H) en el BP ya que el BP (pero no el SP) se puede usar como un registro de índice. Ya que el BP ahora también contiene 0076H, PRICE está en la pila en BP + 8 (desplazamiento 7EH) y QTY está en BP + 6 (desplazamiento 7CH). Sabemos estas localidades relativas porque guardamos en la pila tres palabras (seis bytes) después de que QTY fue guardado en la pila. La rutina transfiere PRICE y QTY desde la pila al AX y al BX, respectivamente, y realiza la multiplicación.

```
                        TITLE     P23MAIN6 (EXE)   Paso de parámetros
                                  EXTRN   P23SUB6:FAR

0000                    STACKSG   SEGMENT PARA STACK 'Stack'
0000 0040[????]                       DW     64 DUP(?)
0080                    STACKSG   ENDS

0000                    DATASG    SEGMENT PARA 'Data'
0000 0140               QTY       DW     0140H
0002 2500               PRICE     DW     2500H
0004                    DATASG    ENDS

0000                    CODESG    SEGMENT PARA PUBLIC 'Code'
0000                    BEGIN     PROC   FAR
                                  ASSUME CS:CODESG,DS:DATASG,SS:STACKSG
0000 B8 ____R                     MOV    AX,DATASG
0003 8E D8                        MOV    DS,AX
0005 FF 36 0002 R                 PUSH   PRICE
0009 FF 36 0000 R                 PUSH   QTY
000D 9A 0000 ____E                CALL   P23SUB6       ;Llama al subprograma
0012 B8 4C00                      MOV    AX,4C00H      ,-Sale al DOS
0015 CD 21                        INT    21H
0017                    BEGIN     ENDP
0017                    CODESG    ENDS
                                  END    BEGIN

Segmentos y grupos
        Ñ a m e             Length   Align   Combine   Class
CODESG. . . . . .           0017     PARA    PUBLIC    'CODE'
DATASG. . . . . .           0004     PARA    NONE      'DATA'
STACKSG. . . . . .          0080     PARA    STACK     'STACK'
Symbols:
        Ñ a m e             Type     Valué    Attr
BEGIN. . . . . .            F PROC   0000     CODESG    Length = 0017
P23SUB6. . . . . .          L FAR    0000     External
PRICE. . . . . .            L WORD   0002     DATASG
QTY. . . . . .              L WORD   0000     DATASG
```

```
                        TITLE     P23SUB6  Subprograma llamado
0000                    CODESG    SEGMENT PARA PUBLIC Code'
0000                    P23SUB6   PROC   FAR
                                  ASSUME CS:CODESG
                                  PUBLIC P23SUB6
0000 55                           PUSH   BP
0001 8B EC                        MOV    BP,SP
0003 8B 46 08                     MOV    AX,[BP+8]     ;Obtiene precio
0006 8B 5E 06                     MOV    BX,[BP+6]     ;Obtiene cantidad
0009 F7 E3                        MUL    BX            ;DX:AX = producto
000B 5D                           POP    BP
OOOC CA 0004                      RET    4
000F                    P23SUB6   ENDP
000F                    CODESG    ENDS
                                  END

Segmentos y grupos
        N                   Length   Align   Combine   Class
CODESG . .                  000F     PARA    PUBLIC    'CODE'
Symbols:
        N                   Type     Valué    Attr
P23SUB6                     F PROC   0000     CODESG  Global Length=000F
```

Figura 23-8 Paso de parámetros

```
Link Map
Object Modules:  P23MAIN6+P23SUB6

Start   Stop    Length Ñame              Class
00000H  0007FH  00080H STACKSG           STACK
00080H  00083H  00004H DATASG            DATA
00090H  OOOBEH  0002FH CODESG            CODE

Program entry point at 0009:0000
```

Figura 23-8 (continuación)

Antes de regresar al programa que llama, la rutina remueve de la pila el BP (regresando la dirección cero al BP), que incrementa en dos el SP, de 76H a 78H.

La última instrucción, RET, es un regreso lejano al programa que llama, el cual realiza lo siguiente:

- Remueve la palabra que está en el tope de la pila de la estructura (1200H) al IP e incrementa en dos el SP, de 78H a 7AH.
- La palabra que ahora está en el tope de la pila (0F20) la envía al CS e incrementa en dos el SP, de 7AH a 7CH.

A causa de los dos parámetros pasados en los desplazamientos 7CH y 7EH, la instrucción RET es codificada como

```
RET    4
```

El 4, conocido como *valoree la operación pop,* contiene el número de bytes en los parámetros que se pasan (en este caso dos parámetros de una palabra cada uno). La operación RET suma el valor de la operación pop al SP, corrigiéndolo a 80H. En efecto, puesto que los parámetros en la pila ya no se necesitan, la operación los deshecha y regresa correctamente al programa que llama. Note que las operaciones POP y RET incrementan el SP, pero en realidad no borran el contenido de la pila.

Si sigue las reglas generales estudiadas en este capítulo debe ser capaz de enlazar un programa que conste de más de dos módulos ensamblados y hacer que los datos sean conocidos en todos los módulos. Pero tenga cuidado del tamaño de la pila: para programas grandes, definirlo de 64 palabras podría ser una precaución sensata, a causa de que podrían tener muchas operaciones PUSH y CALL.

El capítulo 24 trata algunos conceptos importantes sobre la administración de la memoria y la ejecución de programas traslapados. El capítulo 26 proporciona características adicionales de los segmentos, incluyendo la definición de más de un segmento de código o de datos en el mismo módulo ensamblado y el uso de GROUP para combinarlos en un segmento común.

## ENLACE DE PROGRAMAS EN PASCAL Y EN LENGUAJE ENSAMBLADOR

Esta sección explica cómo enlazar un programa Pascal a un subprograma en lenguaje ensamblador. El sencillo programa en Pascal de la figura 23-9, se enlaza a un subprograma en lenguaje ensamblador cuyo propósito es sólo colocar el cursor. El programa Pascal está compilado para producir un módulo .OBJ y el programa en lenguaje ensamblador está ensamblado para producir un módulo .OBJ. Entonces el enlazador combina estos dos módulos .OBJ en un módulo ejecutable .EXE.

El programa Pascal define dos variables llamadas `temprow` y `tempcol` y acepta desde el teclado, los valores de renglón y columna, para estas dos variables. El programa define el nombre del subprograma en lenguaje ensamblador como `setcurs` y define los dos parámetros como extern.

```
program p23pascal ( input, output ) ;

  procedure set_curs( const row: integer;
                      const col: integer ); extern;
  var
       temp_row:       integer;
       temp_col:       integer;

  begin
       write( 'Enter cursor row: ' );
       readln( temp_row ) ;

       write( 'Enter cursor column: ' ),-
       readln( temp_col );

       set_curs( temp_row, temp_col );
       write( 'New cursor location' ) ;
  end.
```

```
TITLE     23SETCUR  Subprograma ensamblador llamado por Pascal
          PUBLIC SET CURS

     SET_CURS:              Coloca el cursor en la pantalla en la posición que se pasa
                            Renglón        Renglón y columna en donde
     Parámetros pasados:    Columna        el cursor será colocado
                            Nada
     Regresados:
CODESEG   SEGMENT PARA PUBLIC 'CODE'
SET_CURS  PROC    FAR
          ASSUME  CS:CODESEG
          PUSH    BP          :Registro BP del que llama
          MOV     BP,SP       ;Apunta a los parámetros que se pasan

          MOV     SI, [BP+8]  ;SI apunta al renglón
          MOV     DH, [SI]    ,-Mueve al renglón DH

          MOV     SI, [BP+6]  ;SI apunta a la columna
          MOV     DL, [SI]    ,-Mueve la columna al DH

          MOV     AH,02H      ;Petición para colocar el cursor
          MOV     BH,0        ;Página de video
          INT     10H

          POP     BP          ;Regresa a donde fue llamado
          RET     4
SET_CURS  ENDP
CODESEG   ENDS
          END
```

Figura 23-9 Enlace de Pascal con ensamblador

Envía las direcciones de temprow y tempcol como parámetros al subprograma para colocar el cursor en esa posición. La instrucción Pascal que "llama" al nombre del subprograma y pasa los parámetros es

```
!             set_curs( temp_row, temp_col );
```

Los valores guardados en la pila son: el apuntador a la pila del programa que llama, el apuntador al segmento de regreso, el desplazamiento de regreso y la dirección de los dos parámetros que se pasaron. En seguida se muestran los desplazamientos para cada entrada en la pila:

00 Apuntador a la pila de la rutina que llamó
02 Apuntador al segmento de regreso de la rutina que llamó
04 Desplazamiento de regreso a la rutina que llamó
06 Dirección del segundo parámetro
08 Dirección del primer parámetro

Ya que el subprograma en lenguaje ensamblador tiene que usar el registro BP, usted tiene que guardar en la pila el BP para conservar su dirección para el regreso al programa Pascal. Note que los pasos en el subprograma llamado son similares a los del programa de la figura 23-7.

Por lo regular, el registro SP direcciona entradas en la pila. Pero como usted no puede utilizar SP para actuar como un registro de índices, el paso después de guardar en la pila el BP es mover la dirección del SP en el BP, lo que le permite usar el BP como un registro de índice para accesar las entradas en la pila de la estructura.

El siguiente paso es accesar las direcciones de los dos parámetros en la pila de la estructura. El primer parámetro que se pasa, el renglón, está en el desplazamiento 08H en la pila de la estructura y puede ser accesado por BP + 08H. El segundo parámetro pasado, la columna, está en el desplazamiento 06H y puede ser accesado por BP + 06H.

Cada una de las dos direcciones en la pila de la estructura tiene que ser transferida a uno de los registros de índice disponibles: BX, DI o SI. Este ejemplo utiliza [BP+08] para mover la dirección del renglón al SI y después utiliza [SI] para mover el contenido del parámetro pasado al registro DH.

La columna es transferida al registro DL de manera similar. Después el subprograma utiliza el renglón y la columna en el registro DX en la INT 10H para colocar el cursor. Al salir, el subprograma remueve de la pila el BP. La instrucción RET necesita un valor del operando que es dos veces el número de parámetros; en este caso, 2 x 2, o 4. De manera automática, los números son removidos de la pila y el control se transfiere de regreso al programa que realizó la llamada.

Si usted cambia un registro de segmento, asegúrese de guardar en la pila (PUSH) la entrada y de removerla de la pila (POP) al salir del subprograma. La práctica recomendada en Pascal es conservar los registros DI, SI, BP, DS y SS. También puede utilizar la pila para pasar valores desde un subprograma a un programa que realiza la llamada. Aunque el subprograma de la figura 23-9 no regresa valores, Pascal esperaría que un subprograma los regrese como una sola palabra en el AX o como un par de palabras en el DX:AX.

Este programa trivial produce un módulo mayor de 20K bytes. Un lenguaje compilador suele generar considerablemente más, sin que importe el tamaño del programa fuente.

No suponga que otras versiones de Pascal necesariamente, siguen las convenciones que hemos utilizado aquí. La norma adecuada es la descrita en el manual del compilador, por lo regular en una sección cuyo título empieza con "Interfacing ..." o "Mixed Languages ...".

## ENLAZANDO PROGRAMAS EN C Y EN LENGUAJE ENSAMBLADOR

El problema con la descripción del enlace de programas en C a programas en lenguaje ensamblador es que las versiones dé C tienen diferentes convenciones. (Para los requerimientos precisos, consulte su manual de C.) Algunos puntos de interés son los siguientes:

- Para versiones de C que son sensibles al uso de mayúsculas y minúsculas, el nombre de los módulos en lenguaje ensamblador debe ser escrito exactamente igual que en la referencia del programa en C.

- La mayoría de las versiones de C pasan los parámetros a la pila en una secuencia *inversa* que la de otros lenguajes. Por ejemplo, considere la instrucción en C

```
Adds (m, n),-
```

La instrucción guarda en la pila n y después la m en ese orden, y luego llama a Adds. Al regresar del módulo llamado, el módulo en C (no el módulo en lenguaje ensamblador) suma 4 al SP para desechar los parámetros pasados. El procedimiento común en el módulo de lenguaje ensamblador llamado para accesar los dos parámetros pasados es como sigue:

```
PUSH   BP
MOV    BP,SP
MOV    DH, [BP + 4]
MOV    DL, [BP+S]

POP    BP
RET
```

- Algunas versiones de C necesitan que un módulo en ensamblador que cambia los registros DI y SI los guarde en la pila al entrar y los saque al salir del subprograma en ensamblador.
- El módulo en ensamblador debe regresar los valores, si se necesitan, como una palabra en el AX o dos palabras en el par DX:AX.
- En algunas versiones de C, un programa en ensamblador que pone en uno la bandera DF debe ponerla en cero (CLD) antes de regresar.

### Cómo enlazar Microsoft C con Microsoft Assembler (C y ensamblador de Microsoft)

**Convención de nombres.** En C y ensamblador de Microsoft, los módulos de lenguaje ensamblador deben usar una convención de nombres de segmentos y variables compatible con la de C. Todas las referencias del ensamblador a funciones y variables en el módulo de C deben iniciar con un carácter de subrayado (_). Además, como C es *sensible a mayúsculas y minúsculas,* el módulo en ensamblador debe utilizar el mismo tipo (mayúsculas o minúsculas) para cualquier nombre de variable en común con el módulo de C.

**Registros.** El módulo en lenguaje ensamblador debe conservar los valores originales en los registros BP, SP, CS, DS, SS, DI y SI.

**Paso de parámetros.** Existen dos métodos para el paso de parámetros:

1. Por *referencia,* ya sea como cercano (un desplazamiento en el segmento por omisión) o como lejano (un desplazamiento en otro segmento). El módulo ensamblador llamado puede alterar directamente el valor definido en el módulo de C.
2. Por *valor,* en el que el llamador en C pasa una copia de la variable a la pila. El módulo en ensamblador llamado puede alterar los valores pasados, pero no tiene acceso al valor original de C. Si existe más de un parámetro, C los guarda en la pila de derecha a izquierda.

**Compatibilidad de los tipos de datos.** La lista siguiente muestra los tipos de variables de **C** y sus tipos equivalentes en ensamblador:

| TIPO DE DATO EN C | TIPO EN MASM 5.X | TIPO EN MASM 6.X |
|---|---|---|
| char | DB | BYTE |
| unsigned short/int | DW | WORD |
| int, short | DW | SWORD |
| unsigned long | DD | DWORD |
| long | DD | SWORD |

**Valores regresados.** El módulo en ensamblador llamado utiliza los registros siguientes para cualquier valor regresado:

| TIPO DE DATO EN C | REGISTRO |
|---|---|
| char | AL |
| short, near, int (16 bits) | AX |
| short, near, int (32 bits) | EAX |
| long, far (16 bits) | DX:AX |
| long, far (32 bis) | EDX:EAX |

Al regresar de un módulo llamado, emita RET sin ningún valor removido de la pila.

**Compilación y ensamblado.** Utilice el mismo modelo de memoria para ambos lenguajes. El enunciado .MODEL de ensamblador indica la convención en C, como .MODEL SMALL,C. También utilice el interruptor de ensamblador apropiado para conservar el tipo (no local) de nombres.

## Enlace de Turbo C con Turbo Assembler

**Interfaces del lenguaje.** Turbo C proporciona dos formas de interfacear con Turbo Assembler, por módulos separados o por código en línea:

1. *Módulos separados.* Para este método, codifique los programas en C y en ensamblador por separado. Utilice TCC para compilar el módulo de C, TASM para ensamblar el módulo de ensamblador y TLINK para enlazarlos.
2. *Código ensamblado en línea.* Para compilar el módulo en C, solicite TCC.EXE (la versión del comando de Turbo C). Sólo inserte las instrucciones en ensamblador, precedidas por la palabra clave asm, en el código fuente, como por ejemplo,

```
asm INC WORD PTR FLDX
```

**Segmentos.** El segmento de código debe ser llamado TEXT. El segmento de datos (dos, si se necesitan) son llamados DATA para datos que se inicializarán a la entrada del bloque y _BSS para datos no inicializados.

**Convención de nombres.** Los módulos de Turbo Assembler deben utilizar la convención de nombres para segmentos y variables que sea compatible con la de Turbo C. Todas las referen-

cías a funciones y nombres en el módulo de C deben iniciar con un carácter de subrayado (_). Además, como C es sensible a mayúsculas y minúsculas, el ensamblador debe utilizar el mismo tipo de letra (mayúsculas o minúsculas) para cualesquiera nombres de variables **en** común con el módulo de C.

**Registros.** El módulo en ensamblador puede utilizar libremente los registros AX, BX, CX, DX, ES y de las banderas. También puede emplear los registros BP, SP, CS, DS, SS, DI y SI, siempre y cuando los guarde (en la pila) y los restaure (los remueva de ella).

**Paso de parámetros.** Turbo C pasa los parámetros por valor. Si hay más de un parámetro, Turbo los guarda en la pila de derecha a izquierda.

**Regreso.** El programa en ensamblador sólo utiliza RET (sin valor de la operación pop) para regresar al módulo en C. El módulo en C lo saca de la pila al regresar.

### **Ejemplo de** un **programa C**

La figura 23-10 ilustra el enlace de un programa en Turbo C con un módulo en ensamblador. El programa realiza las mismas acciones que el de Pascal de la sección anterior: el programa C acepta valores desde el teclado para el renglón y la columna y los pasa al subprograma en ensamblador. El subprograma en ensamblador a su vez coloca el cursor y regresa al módulo en C.

## PUNTOS CLAVE

- El operador para alinear le indica al ensamblador que alinee el segmento nombrado, iniciando en una frontera particular de almacenamiento.
- El operador para combinar le indica al ensamblador y al enlazador si combinan segmentos o los mantienen separados.
- Puede asignar la misma clase de segmentos relacionados de modo que el ensamblador y enlazador los agrupen.
- Una llamada (CALL) intrasegmento es cercana si el procedimiento llamado está definido o por omisión es NEAR (cercano, dentro de 32K). Una llamada intrasegmento puede ser lejana si es a un procedimiento lejano dentro del segmento.
- Una llamada (CALL) intersegmento llama a un procedimiento en otro segmento y es definida como FAR o como EXTRN.
- En un programa principal que llama a un subprograma, se define el punto de entrada como EXTRN; en el subprograma, como PUBLIC.
- Si dos segmentos de código van a ser enlazados en un segmento, defínalos con el mismo nombre, la misma clase y el tipo combinar PUBLIC.
- Por lo general es más fácil (pero no necesario) definir datos comunes en el programa principal. El programa principal define los datos comunes como PUBLIC y el subprograma (c subprogramas) define los datos comunes como EXTRN.

```
#include <stdio.h>

int main (void)
/
{
   int temp_row, temp col/

   printf ("Enter cursor row: " ) /
   scanf ("%d", &temp_row)¡

   printf ("Enter cursor column: " ) /
   scanf ("%d", &temp col)/

   set_curs (temp row, temp col);
   printf ("New cursor location\n");
}
```

```
; Utilice modelo de memoria pequeño para C: código y datos cercanos
; Utilice nombres de segmentos y directivas de grupo 'estándar'

_DATA     segment word 'DATA'
row       equ      [bp+4]          /Parámetros
col       equ      [bp+6]            (argumentos)
_DATA     ends


 TEXT     SEGMENT BYTE PUBLIC 'CODE'
DGROUP    GROUP    DATA
          ASSUME   CS:_TEXT, DS:DGROUP, SS:DGROUP

          PUBLIC    set curs
_set curs PROC     NEAR
          PUSH     BP              /Registro BP del que llama
          MOV      BP, SP          /Apunta a los parámetros

          MOV      AH, 02H         /Petición para colocar el cursor
          MOV      BX, 0           /Página de video
          MOV      DH, ROW         /Renglón de BP+4
          MOV      DL, COL         /Columna de BP+G
          INT      10H             /Llama al BIOS

          POP      BP              /Restaura BP
          RET                      /Regresa a donde fue llamado
 set curs ENDP
_TEXT     ENDS
          END
```

Figura 23-10 Enlace de C a ensamblador

## PREGUNTAS

23-1. Proporcione cuatro razones para organizar un programa en subprogramas. Las tres preguntas siguientes se refieren al formato general para la directiva SEGMENT:

```
nom_seg  SEGMENT  [alinear]  [combinar]  ['clase']'
```

23-2. (a) Para la opción alinear de la directiva SEGMENT, ¿cuál es el valor por omisión? (b) ¿Cuál es el efecto de la opción BYTE? (Esto es, ¿qué acción toma el ensamblador?)

23-3. (a) Para la opción combinar de la directiva SEGMENT, ¿cuál es el valor por omisión? (b) ¿Cuándo usaría la opción PUBLIC? (c) ¿Cuándo usaría la opción COMMON?

23-4. (a) ¿Cuál debe ser la opción de clase del segmento de código para la directiva SEGMENT? (b) Dos segmentos tienen la misma clase, pero no la opción combinar PUBLIC. ¿Cuál es el efecto de esto? (c) Dos segmentos tienen la misma clase y la opción combinar PUBLIC. ¿Cuál es el efecto de esto?

23-5. Distinga entre una llamada intrasegmento y una llamada intersegmento.

23-6. Un programa llamado MAINPRO llama a un subprograma llamado SUBPRO. (a) ¿Qué instrucción en MAINPRO informa al ensamblador que el nombre SUBPRO está definido fuera de su propio ensamble? (b) ¿Qué instrucción en SUBPRO es necesaria para hacer conocer su nombre a MAINPRO?

23-7. Suponga que MAINPRO en la pregunta 23-6 ha definido variables llamadas QTY como DB, VALUÉ como DW y PRICE como DW. SUBPRO divide VALUÉ entre QTY y el cociente se almacena en PRICE. (a) ¿Cómo informa MAINPRO al ensamblador que las tres variables son conocidas fuera de este ensamble? (b) ¿Cómo informa SUBPRO a! ensamblador que las tres variables están definidas en otro ensamble?

23-8. Combine las preguntas 23-6 y 23-7 en un programa que funcione y pruébelo.

23-9. Corrija la pregunta 23-6 de modo que MAINPRO pase las tres variables como parámetros. Sin embargo, observe que SUBPRO regresa el precio calculado intacto en su parámetro.

23-10. Extienda la pregunta 23-9 de modo que MAINPRO acepte cualquier cantidad y número desde el teclado, el subprograma SUBCONV convierta las cantidades ASCII a binario, el subprograma SUBCALC calcule el precio y el subprograma SUBDISP convierta el precio binario a ASCII y despliegue el resultado.

CAPÍTULO 24

# Administración de la memoria del DOS

OBJETIVO

Describir el proceso de arranque, la inicialización del DOS, el prefijo de segmento de programa, el entorno (ambiente), control de memoria, el cargador de programas y los programas residentes.

## INTRODUCCIÓN

Este capítulo describe en detalle la organización del DOS. Las operaciones introducidas son la función 4A01H de la INT 2FH del DOS, la interrupción de multiplexión, y estas funciones de la INT 21H:

| | |
|---|---|
| 25H | Establece la dirección de la interrupción |
| 31H | Guarda el programa |
| 3306H | Obtiene la versión del DOS |
| 34H | Obtiene la dirección de bandera ocupada del DOS |
| 35H | Obtiene la dirección de la interrupción |
| 48H | Asigna memoria |
| 49H | Memoria libre asignada |
| 4AH | Modifica el bloque de la memoria asignada |
| 4BH | Carga o ejecuta un programa |
| 51H | Obtiene la dirección del segmento del PSP actual |
| 52H | Obtiene la dirección de la lista interna del DOS |
| 58H | Obtiene/establece la estrategia de asignación de memoria |

## PROGRAMAS PRINCIPALES DEL DOS

Los cuatro programas principales del DOS son el registro de arranque, IO.SYS, MSDOS.SYS y COMMAND.COM:

1. El *registro de arranque* está en la pista O, sector 1 de cualquier disco que usted formatee con FORMAT /S. Cuando inicializa la comutadora, el sistema carga de manera automática el registro de arranque del disco y lo envía a la memoria. El registro de arranque, a su vez, carga IO.SYS del disco a la memoria.
2. *IO.SYS* es una interfaz de bajo nivel con las rutinas del BIOS en ROM. En la iniciación, determina el estado de los dispositivos y el equipo asociado con la computadora y establece las direcciones de la tabla de interrupciones hasta la 20H. IO.SYS también maneja la entrada/ salida entre la memoria y los dispositivos externos, como el monitor o los discos. Después carga el MSDOS.SYS.
3. *MSDOS.SYS* es una interfaz de alto nivel para programas que establece las direcciones de la tabla para las interrupciones de la 20H a la 3FH. Administra el directorio y los archivos en disco, bloqueo y desbloqueo de registros en disco, las funciones de la INT 21H y otros servicios. Después carga el COMMAND.COM.
4. *COMMAND. COM* maneja los diferentes comandos, como DIR y CHKDSK, y ejecuta todas las peticiones de programas .COM, .EXE y .BAT. Es responsable de cargar los programas ejecutables de disco a memoria.

La figura 24-1 muestra un mapa de memoria después que los programas de sistema del DOS han sido cargados. Los detalles varían de sistema a sistema.

```
Dirección
de Inicio            Contenido

  FOOOOH   Área ROM del sistema
  EOOOOH   ROM   BIOS
  DOOOOH   ROM   BIOS                                                    *
  COOOOH   ROM   BIOS
  BOOOOH   Búffers de video
  AOOOOH   Búffers de video
  xxxxOH   Porción transitoria de C0MMAND.COM, en el tope de la RAM

           Programas de usuario
           Programas residentes (si hay alguno)
  xxxxOH   Porción residente de C0MMAND.COM
  xxxxOH   MSDOS.SYS e 10.SY
  00500H   Área de comunicación del DOS
  00400H-  Área de datos del BIOS
  OOOOOH   Tabla de direcciones de_J.nterrupción

   Nota:   La memoria convencional va de OOOOOH a AOOOOH (640K).
           El área de la memoria superior va de AOOOOH a FFFFOH (un mega).
           El área de la memoria alta (HMA por sus siglas en inglés)
           es de 64K y va de FFFFOH a FFFFFH.
           La memoria extendida está arriba de HMA.
```

Figura 24-1 Mapa de la memoria

## ÁREA DE MEMORIA ALTA

El procesador utiliza varias líneas de direcciones para accesar la memoria. Para el 80286 y posteriores, la línea número A20 puede direccionar un espacio de 64K conocido como *área de memoria alta* (HMA) desde FFFF: 10H hasta FFFF:FFFFH, justo abajo del límite de un megabyte del DOS.

Cuando la computadora corre en modo real (8086), por lo regular deshabilita la línea A20 de modo que las direcciones que pasan este límite "dan la vuelta" al inicio de la memoria. Habilitar la línea A20 permite direccionar localidades en la HMA. Desde DOS 5.0, puede indicar al CONFIG.SYS que reubique el DOS de memoria baja en el HMA, y por tanto libere espacio para los programas del usuario. Puede utilizar la función 3306H (obtener versión del DOS) de la INT 21H para determinar la presencia del DOS en el HMA:

```
MOV  AX,3306H    ;Petición de la versión del DOS
INT  21H         /Llama al DOS
```

La operación regresa lo siguiente:

- BL = Número principal de la versión (como el 7 para la versión 7.1)
- BH = Número secundario de la versión (como el 1 en la versión 7.1)
- DL = Número de revisión en los tres bits inferiores (2-0)
- DH = banderas de la versión del DOS, en donde el bit 4 = 1 significa en el HMA

La INT 2FH del DOS (interrupción de multiplexión), entre otros servicios, también proporciona una verificación (por medio de la función 4A01H) del espacio disponible en el HMA:

```
MOV  AX.4A01H    ;Petición de espacio en HMA
INT  2FH         ,•Llama al DOS
```

La operación regresa lo siguiente:

- BX = Número de bytes libres disponibles en el HMA (cero si el DOS no está cargado en el área alta)
- ES:DI = Dirección del primer byte libre en el HMA (FFFF:FFFF si el DOS no está cargado en el área alta)

## COMMAND.COM

El sistema carga las tres partes del C0MMAND.COM en la memoria, ya sea de manera permanente durante una sesión o bien cuando se requiera de manera temporal. A continuación se describen las tres partes:

1. La *parte residente* de C0MMAND.COM carga de forma inmediata MSDOS.SYS (y sus áreas de datos) en donde reside durante el procesamiento. La parte residente maneja errores para E/S de disco y las interrupciones siguientes:

   | | |
   |---|---|
   | INT 22H | Dirección de terminación |
   | INT23H | Manejador de Ctrl+Break |
   | INT 24H | Detección de error en lectura/escritura en disco o incorrecta imagen de la memoria de la FAT |
   | INT 27H | Termina pero permanece residente (TSR; residente en memoria) |

2. La *parte de inicialización* de COMMAND.COM sigue inmediatamente a la parte residente y contiene la configuración para archivos AUTOEXEC. Cuando el sistema inicia, la parte de inicialización toma el control y determina la dirección del segmento en la cual el sistema cargará programas para su ejecución. Ninguna de las rutinas de inicialización se requiere durante la sesión. En consecuencia, su primera petición para cargar un programa desde disco hace que el DOS traslape la parte de inicialización de la parte del COMMAND.COM siempre que resida en memoria.
3. La *parte transitoria* de C0MMAND.COM es cargada en un área alta de memoria. "Transitoria" implica que, si es necesario, el DOS puede traslapar esta área con otros programas requeridos. La parte transitoria muestra una petición común en pantalla y acepta y ejecuta las peticiones. Contiene un cargador de reubicación que carga archivos .EXE y .COM desde disco hacia la memoria para su ejecución. Cuando se pide la ejecución de un programa, la parte transitoria construye un segmento de programa en la más baja localidad de memoria disponible. Crea el PSP en OOH, carga el programa ejecutable pedido en el desplazamiento 100H, establece las direcciones de salida y le pasa el control al programa que cargó.

Cuando se termina de una manera normal un programa, esto produce un regreso a la parte residente de C0MMAND.COM. Si el programa ejecutado se traslapó con la parte transitoria de COMMAND.COM, la parte residente vuelve a cargarla en memoria.

## PREFIJO DE SEGMENTO DE PROGRAMA (PSP)

El DOS carga programas .COM y .EXE para ejecución en un segmento de programa y crea un PSP en el desplazamiento OOH y el programa mismo en el desplazamiento 100H del segmento. El PSP contiene los campos siguientes de acuerdo con la posición relativa:

| | |
|---|---|
| 00-01H | Una instrucción de la INT 20H (CD20H) para facilitar el regreso al DOS. |
| 02-03H | La dirección del segmento del último párrafo de la memoria asignada al programa, como xxxxO. Por ejemplo, 640K se indica como OOAOH, para significar A0000[0]. |
| 04-09H | Reservado por el DOS. |
| OA-ODH | Dirección de terminación (dirección del segmento para INT 22H). |
| OE-11H | Dirección de salida de Ctrl + Break (dirección de segmento para la INT 23H). |
| 12-15H | Dirección de salida de error crítico (dirección de segmento para la INT 24H). |
| 16-17H | Reservado por el DOS. |
| 18-2BH | Tabla de manejadores de archivo por omisión. |
| 2C-2DH | Dirección de segmento del entorno del programa. |
| 2E-31H | Reservado por el DOS. |
| 32-33H | Longitud de la tabla de manejadores de archivo. |
| 34-37H | Apuntador lejano a la tabla de manejadores. |
| 38-4FH | Reservado por el DOS. |
| 50-51H | Llama a la función del DOS (INT 21H y RETF). |
| 52-5BH | Reservado por el DOS. |
| 5C-6BH | Área de parámetro 1, formateado como un FCB (#1) estándar no abierto. |

6C-7FH Área de parámetro 2, formateado como un FCB (#2) estándar no abierto; traslapado, si el FCB en 5CH está abierto.
80-FFH Búfer por omisión para un DTA.

**PSP 18-2BH: Tabla de manejadores de archivo por omisión**

Cada byte en la tabla, de 20 bytes, de manejadores de archivo por omisión hace referencia a una entrada en una tabla del DOS que define el dispositivo o controlador relacionado. Inicialmente, la tabla contiene 0101010002FF ... FF, en donde el primer 01 hace referencia al teclado, el segundo 01 a la pantalla y así sucesivamente:

| TABLA | DISPOSITIVO | MANEJADOR | DISPOSITIVO |
|---|---|---|---|
| 01 | Consola | 0 | Teclado (entrada estándar) |
| 01 | Consola | 1 | Pantalla (salida estándar) |
| 01 | Consola | 2 | Pantalla (error estándar) |
| 00 | COMÍ (puerto serial) | 3 | Auxiliar |
| 02 | Impresora | 4 | Impresora estándar |
| FF | No asignado | 5 | No asignado |

La tabla de 20 manejadores explica por qué el DOS permite un máximo de 20 archivos abiertos al mismo tiempo. Por lo común, la palabra en PSP desplazada 32H contiene la longitud de la tabla (14H o 20) y 34H contiene su dirección de segmento en la forma IP:CS, en donde el IP es 18H (el desplazamiento en el PSP) y el CS es la dirección del segmento del PSP.

Los programas que necesitan más de 20 archivos abiertos tienen que liberar memoria (INT 21H, función 4AH) y utilizar la función 67H (fija el máximo de manejadores):

```
MOV   AH,67H        ;Petición de manejadores
MOV   BX.count      ;Nuevo número (20 a 65,535)
INT   21H           ;Llama al DOS
```

La cantidad de memoria requerida es un byte por cada manejador, redondeado al siguiente byte de párrafo más 16 bytes. La operación crea la nueva tabla de manejadores fuera del PSP y actualiza las localidades 32H y 34H del PSP. Una operación no válida pone en uno la bandera de acarreo y coloca un código de error en el AX.

**PSP 2C-2DH: Dirección del segmento de entorno**

Cada programa cargado para ejecución tiene un *entorno (ambiente)* relacionado que el DOS almacena en la memoria, iniciando en una frontera de párrafo antes del segmento de programa. El tamaño por omisión es de 160 bytes, con un máximo de 32K. El entorno contiene comandos del DOS como COMSPEC, PATH, PROMPT y SET que son aplicables al programa.

**PSP 5C-6BH: FCB #1 estándar no abierto**

El DOS formatea esta área con un FCB #1, ficticio **o** real, con base en los caracteres (si hay) que usted ingresó después de una petición para el nombre de un programa que será ejecutado, como MASM D:PROGRAMl.ASM. El FCB #1 contiene el *primer* (o único) nombre de archivo ingresado.

**PSP 6C-7FH: FCB #2 estándar no abierto**

También el DOS formatea esta área con un FCB *#2* ficticio o real, con base en los caracteres (si hay) que usted ingresó cuando solicitó el nombre de un programa que será ejecutado. El FCB #2 contiene el *segundo* (si hay) nombre de archivo ingresado.

**PSP 80-FFH: Búfer por omisión del DTA**

Esta parte del PSP es llamada el *búfer por omisión* para el DTA. El DOS inicializa esta área con el texto completo (si hay) que el usuario teclea a continuación de la petición por un nombre de programa. El primer byte contiene el número de teclas presionadas (si hay) inmediatamente después del nombre del programa ingresado, seguido por el número real de caracteres ingresados. Después de esto hay "basura" a la izquierda de un programa anterior.

Los siguientes cuatro ejemplos clarifican el contenido y propósito de FCB #1, FCB #2 y del DTA.

**Ejemplo 1:** Comando sin operando. Suponga que un usuario hace que un programa llamado CALCIT.EXE se ejecute al teclear CALCIT[Enter]. Cuando el DOS construye el PSP para este programa, configura FCB #1, FCB #2 y el DTA por omisión, como sigue:

```
5CH FCB #1:    00  20  20  20  20  20  20  20  20  20  20  20  ...
6CH FCB #2:    00  20  20  20  20  20  20  20  20  20  20  20  ...
8OH DTA:       0  0  0D  ...
```

***FCB §1 y FCB* #2:** Ambos son FCB ficticios. El primer byte de cada uno, 00H, se refiere al número de unidad por omisión. Los bytes subsecuentes para el nombre y la extensión del archivo están en blanco, ya que el usuario no ingresó texto después del nombre del programa tecleado.

***DTA:*** El primer byte contiene el número de bytes tecleados después del nombre CALCIT, sin incluir el carácter Enter. Ya que no tecleó algo más después del Enter, el número es cero. El segundo byte contiene el carácter Enter 0DH, que fue presionado.

**Ejemplo 2:** Comando con operando de texto. Suponga que el usuario necesita ejecutar un programa llamado COLOR y pasar un parámetro "BY" que le indica al programa poner azul (B) sobre fondo amarillo (Y). El usuario teclea el nombre del programa seguido por el parámetro: COLOR BY. Entonces el DOS coloca lo siguiente en el PSP:

```
5CH FCB #1:    00  42  59  20  20  20  20  20  20  20  20  20  ...
6CH FCB #2:    00  20  20  20  20  20  20  20  20  20  20  20  ...
8OH DTA:       03  20  42  59  0D
```

***FCB til:*** El DOS configura el FCB #1 con 0OH como la unidad por omisión y 4259H (BY) como el nombre del archivo. Observe que el DOS no sabe si el nombre del archivo es válido.

***DTA:*** Los bytes en 80H significan una longitud de 3, seguido por un espacio, "BY", y el carácter Enter. Además de la longitud, este campo contiene exactamente lo que se tecleó.

**Ejemplo 3:** Comando con un operado nombre de archivo. Programas como DEL del DOS permiten a los usuarios ingresar un nombre de archivo después del nombre del programa. Por ejemplo, si el usuario teclea DEL D:CALCIT.OBJ [Enter], el PSP contiene lo siguiente:

```
5CH FCB #1:   04 43 41 4C 43 49 54 20 20 4F 42 4A ...
                 C  A  L  C  I  T           O  B  J
6CH FCB #2:   00 20 20 20 20 20 20 20 20 20 20 20 ...
80H DTA:      OD 20 44 3A 43 41 4C 43 49 54 2E 4F 42 4A 0D ...
                    D  :  C  A  L  C  I  T  .  O  B  J
```

***FCB*** #/; El primer carácter indica el número de la unidad (04 = D), seguido por el nombre del archivo, CALCIT, al que el programa hace referencia. Después vienen dos espacios en blanco que completan el nombre del archivo a ocho caracteres; finalmente, la extensión, OBJ.

***DTA:*** La longitud de 13 (ODH) es seguida exactamente de lo que fue tecleado, incluyendo el carácter Enter.

Ejemplo **4:** Comando con dos operandos de nombre de archivo. Considere ingresar un comando seguido por dos parámetros, como

```
COPY A:FILEA.ASM D:FILEB.ASM
```

El DOS coloca en los FCB y en el DTA lo siguiente:

```
5CH FCB #1:   01 46 49 4C 45 41 20 20 20 41 53 4D ...
                 F  I  L  E  A              A  S  M
6CH FCB #2:   04 46 49 4C 45 42 20 20 20 41 53 4D ...
                 F  I  L  E  B              A  S  M
80H DTA:      10 20 41 3A 46 49 4C 45 41 2E 41 53 4D 20 etc..
                    A  :  F  I  L  E  A  .  A  S  M    etc. . .
```

***FCB §1:*** El primer byte, 01, se refiere a la unidad A, seguida por el nombre del archivo.

***FCB #2:*** El primer byte, 04, se refiere a la unidad D, seguida por el nombre del archivo.

***DTA:*** Los bytes contienen el número de caracteres ingresados (10H), un espacio (20H), A:FILEA.ASM D:FILEB.ASM y el carácter Enter (ODH).

### Cómo accesar el PSP

Para determinar la dirección del PSP, puede accesar sus datos para procesar archivos especificados o tomar una acción especial. Un programa .EXE no siempre puede suponer que su segmento de código sigue inmediatamente al PSP. La función 51H del DOS envía al registro BX la dirección del segmento del actual PSP. El código siguiente obtiene la dirección del PSP y la guarda en el registro ES:

```
MOV AH,51H      ;Petición de la dirección del PSP
INT 21H         ;Llama al DOS
MOV ES,BX       ;Guarda la dirección del PSP en ES
```

Ahora puede utilizar el ES para accesar los datos en el PSP:

```
CMP   ES:BYTE PTR[80H],0    .-Verifica el búfer del PSP
JE    EXIT                  ; cero, no hay datos
```

Para localizar el DTA para un programa .COM, sólo coloque 80H en el registro SI, DI o BX y accese su contenido:

```
MOV   SI,80H              .Dirección del DTA
CMP   BYTE PTR[SI¡,0      .-Verifica el búfer (DS:SI)
JE    EXIT                ; cero, no hay datos
```

### Extensión del ejemplo que utiliza el PSP

El programa .COM parcial en esta sección coloca el atributo de un archivo solicitado en normal (00H). El usuario tecleará el nombre del programa seguido por el nombre del archivo, como P24ATTRB d:nomarch.ext. El programa busca el DTA para el carácter Enter y lo reemplaza con un byte de ceros hexadecimales, lo que crea una cadena ASCIIZ. También, el usuario podría teclear la ruta del directorio. A continuación está el programa codificado:

```
TITLE   P24ATTRB (.COM)  "Fija el atributo del archivo en normal'
CODESG  SEGMENT PARA
        ASSUME  CS:CODESG
        ORG     100H
BEGIN:  MOV     AL,ODH        ,-Busca carácter (Enter)
        MOV     CX, 21        ,-Número de bytes
        MOV     DI,82H        .-Dirección inicial en PSP
        REPNZ   SCASB         ,-Busca el Enter
        JNZ     ***           ,-No encontrado, error
        DEC     DI            ;Encontrado:
        MOV     BYTE PTR [DI],0   ;Reemplazarlo con 00H
        MOV     AH,43H        ,-Petición para
        MOV     AL, 01        ; establecer atributo
        MOV     CX, 00        ,- a normal
        MOV     DX,82H        ,-Cadena ASCIIZ en el PSP
        INT     21H           ;Llama al DOS
        JC      ***           /¿Error de escritura?...

        ENDS
        END     BEGIN
```

## BLOQUES DE MEMORIA

El DOS permite que cualquier número de programas sean cargados y permanezcan residentes. Ejemplos de esto incluyen a RAMDISK, MOUSE y SIDEKICK. El DOS configura uno o dos *bloques de memoria* para cada programa cargado. Precediendo de manera inmediata a cada bloque de memoria está un *encabezado de arena* (o *registro de control de memoria)* empezando en una frontera de párrafo y que contiene los campos siguientes:

| | |
|---|---|
| 00-00H | Código, en donde 4DH ('M') significa que más bloques a continuación y 5AH ('Z') significa cero bloques a continuación (el último bloque). (Ésta es una interpretación útil, pero no es necesariamente la intención original.) |
| 01-02H | Dirección del segmento del PSP del propietario. 0800H significa que el segmento pertenece al MSDOS.SYS y 0000H significa que está liberado y disponible. |
| 03-04H | Longitud del bloque de memoria, medida en párrafos. |
| 05-07H | Reservada. |
| 08-OFH | Nombre de archivo del propietario, en formato ASCIIZ (desde el DOS 4.0). |

Una lista enlazada hacia adelante conecta los bloques de memoria. El *primer* bloque de memoria, configurado y apropiado por el MSDOS.SYS, contiene: los búfers de archivo del DOS, los FCB usados por funciones de manejadores de archivos y controladores de dispositivos cargados por los comandos DEVICE en el CONFIG.SYS.

El *segundo* bloque de memoria es la parte residente del COMMAND.COM con su propio PSP. Unos cuantos programas especiales, como FASTOPEN y SHARE, pueden ser cargados antes del COMMAND.COM.

El *tercer* bloque de memoria es el entorno maestro que contiene el comando COMSPEC, los comandos PROMPT, los comandos PATH y las cadenas que se establecen por medio de SET.

Los bloques sucesivos incluyen cualesquiera programas residentes (TSR) y el programa actual que se está ejecutando. Cada uno de estos programas tiene dos bloques; el *primero* es una copia del entorno y el *segundo* es un segmento de programa con el PSP y el módulo ejecutable.

### INT 21H, función 52H: Obtiene dirección de la lista interna del DOS

El encabezado de arena para el primer bloque de memoria que pertenece a MSDOS.SYS puede ser localizado por medio de una característica no documentada: la INT 21H, función 52H. La tabla de direcciones del DOS inicia con estas entradas:

| | | |
|---|---|---|
| OOH | DD | Dirección del primer bloque de parámetros de la unidad |
| 04H | DD | Dirección de la lista de tablas de archivo del DOS |
| 08H | DD | Dirección del controlador de dispositivo CLOCK$ |
| OCH | DD | Dirección del controlador de dispositivo CON |

La función 52H regresa la dirección del segmento de la lista de tablas de archivos del DOS (la segunda entrada) en el ES y un desplazamiento en el BX. Por tanto ES:[BX-4] apunta a la entrada precedente, que es una palabra doble en formato IP:CS que contiene la dirección del primer encabezado de arena.

Para encontrar los bloques de memoria subsecuentes en la cadena:

1. Utilice la dirección del encabezado de arena para el bloque de memoria.
2. Sume 1 a la dirección del segmento del encabezado de arena para obtener el inicio de su bloque de memoria. (El tamaño del encabezado de arena es de 10H bytes.)
3. Sume la longitud del bloque de memoria, que se encuentra en los desplazamientos 03-04H del encabezado de arena.

Para determinar los párrafos de memoria disponible para el último programa, encuentre el encabezado de arena que contiene "Z" en el byte O y realice los cálculos anteriores. El último bloque tiene disponible, para él, todo la memoria superior restante.

Ejemplo **de** rastreo **de** bloques **de** memoria

Si utiliza DEBUG para rastrear por los bloques de memoria en su propio sistema, puede utilizar el comando H (hexadecimal) de DEBUG para aritmética hexadecimal. Úselo así:

```
H valor1,valor2
```

El comando H regresa la suma y la diferencia de los dos números.

Para el ejemplo siguiente, DEBUG despliega el contenido de la memoria requerida. Tenga cuidado con la secuencia inversa de byte. El rastreo procede como se muestra a continuación:

1. La función 52H regresó 02CC[0] en el ES y 0026H en el BX. Como queremos cuatro bytes a la izquierda de 0022H, utilice D 02CC:22 para desplegar la dirección del encabezado de arena para el primer bloque de memoria en el formato IP:CS. Esto produce 00 00 56 0B. Por lo tanto la dirección es 0B56[0].
2. Utilice D B56:0 para desplegar el primer encabezado de arena:

   ```
   4D 0 8 0 0 AE 05 ...
   ```

   4D ("M") significa que siguen más bloques de memoria; 0800 (0008H) indica que el bloque de memoria pertenece a MSDOS.SYS, y AE05 (05AEH) es la longitud del bloque de memoria.
3. Localice el segundo encabezado de arena (COMMAND.COM):

   ```
   Localidad del primer encabezado de arena:   B56[0]
   Sume 1 párrafo:                           + 1 [0]
   Sume la longitud de este bloque de memoria: + 5AE [0]
                                               ------
   Localidad del siguiente encabezado de arena: 1105 [0]
   ```

   Utilice D 1105:0 para desplegar el segundo encabezado de arena:

   ```
   4D 06 11 64 01 ...
   ```

   En este momento, también podría examinar el contenido de COMMAND.COM.
4. Localice el tercer encabezado de arena, el entorno maestro:

   ```
   Localidad del encabezado de arena anterior: 1105[0]
   Sume 1 párrafo:                           + 1 [0]
   Sume la longitud de este bloque de memoria: + 164 [0]
                                               ---------
   Localidad del siguiente encabezado de arena: 126A [0]
   ```

   Utilice D 126A:0 para desplegar el tercer encabezado de arena: 4D...

Podría seguir el mismo procedimiento para examinar el contenido del entorno maestro y localizar cualesquiera bloques de memoria restantes. Note que los programas subsecuentes tienen *dos* bloques de memoria cada uno: uno para su entorno y uno para su segmento de programa. El último encabezado de arena tiene 5AH ("Z") en su primer byte. Si despliega desde DEBUG, éste es su propio bloque de memoria, ya que DEBUG sería el último programa cargado en memoria.

### Manejo de bloques de memoria superior

Desde el DOS 5.0, el CONFIG.SYS puede tener una instrucción DOS = UMB (bloque de memoria superior) para asignar memoria a programas por arriba de la memoria convencional, entre las fronteras de los 640K y 1024K. La instrucción hace que el DOS establezca un encabezado dé arena ficticio de 16 bytes antes de la frontera de los 640K y lo marque como suyo. Su campo del . tamaño contiene un número suficientemente grande para pasar cualquier búfer de video y rutinas de ROM.

De esta manera, es posible configurar el último encabezado de arena en memoria convencional para ubicar bloques de memoria en memoria superior. Dentro de la memoria superior, otros encabezados de arena marcados como propios también son utilizados para pasar áreas ya utilizadas por ROM o video.

## ESTRATEGIA DE ASIGNACIÓN DE MEMORIA

El DOS utiliza varias estrategias para determinar en dónde cargar un programa en memoria. La función 58H de la INT 21H proporciona servicios para este fin.

### Función 5800H: Obtiene la estrategia de asignación de memoria

Esta operación permite consultar la estrategia de asignación de memoria:

```
MOV AX,5800H       ;Petición para obtener la estrategia
INT 21H            ;Llama al DOS
```

La operación pone en cero la bandera de acarreo y regresa la estrategia en el AX:

- OOH = Primer ajuste (por omisión): Busca desde la dirección más baja en memoria convencional el primer bloque disponible que es lo bastante grande para cargar el programa.
- 01H = Mejor ajuste: Busca el bloque más pequeño disponible en memoria convencional que sea lo bastante grande para cargar el programa.
- 02H = Último ajuste: Busca, desde la dirección más alta en memoria convencional, el primer bloque disponible.
- 40H = Primer ajuste, sólo arriba: Busca, desde la dirección más baja en memoria superior, el primer bloque disponible.
- 41H = Mejor ajuste, sólo arriba: Busca el bloque disponible más pequeño en memoria superior.
- 42H = Último ajuste, sólo arriba: Busca, desde la dirección más alta en memoria superior, el primer bloque disponible.
- 80H = Primer ajuste, arriba: Busca, desde la dirección más baja en memoria superior, el primer bloque disponible. Si no se encuentra, busca en memoria convencional.
- 81H = Mejor ajuste, arriba: Busca el bloque disponible más pequeño en memoria superior. Si no lo encuentra, busca en memoria convencional.
- 82H = Último ajuste, arriba: Busca, desde la dirección más alta en memoria superior, el primer bloque disponible. Si no lo encuentra, busca en memoria convencional

Las estrategias de mejor y último ajuste son apropiadas para sistemas de multitareas, que podrían tener fragmentada la memoria a causa de los programas que se ejecutan de manera concurrente. Cuando un programa termina su procesamiento, su memoria queda liberada para el sistema.

**Función 5801H: Establece estrategia de asignación de memoria**

Esta operación permite cambiar la estrategia de asignación de memoria. Para establecer una estrategia, coloque el código 01 en el AL y el código de la estrategia en el BX. Un error pone en uno la bandera de acarreo y regresa 01 (función no válida) en el AX.

**Función 5802H: Obtiene enlace a la memoria superior**

Esta operación indica si un programa puede asignar memoria desde el área de la memoria superior (por arriba de los 640K). La operación pone en cero la bandera de acarreo y regresa uno de los siguientes códigos en el AL:

- 00H = Área no está enlazada, no puede asignar
- 01H = Área está enlazada, puede asignar

**Función 5803H: Establece enlace con la memoria superior**

Esta operación puede enlazar o romper el enlace con el área de la memoria superior; si el área está enlazada, puede asignar memoria de ella:

```
MOV  AX.5803H       ;Petición para
MOV  BX,linkflag    ; enlazar/desenlazar
INT  21H            ; área de memoria superior
```

El parámetro de la bandera de enlace tiene el significado siguiente:

- 00H = desenlaza el área
- 01H = enlaza el área

Una operación exitosa pone en cero la bandera de acarreo y permite al programa asignar memoria desde ella. Un error pone en uno la bandera de acarreo y regresa en el AX el código 01 (CONFIG.SYS no contenía DOS = UMB) o el código 07 (enlace de memoria dañado).

## CARGADOR DE PROGRAMA

Al cargar programas .COM y .EXE, el DOS realiza lo siguiente:

1. Configura bloques de memoria para el entorno y el segmento del programa.
2. Crea un prefijo de segmento de programa en su localidad 00H y carga el programa en 100H.

Además de éstos, los pasos de carga y ejecución difieren para programas .COM y .EXE. Una diferencia principal es que el enlazador inserta un registro de encabezado especial en un archivo .EXE cuando lo almacena en disco; el cargador del DOS utiliza este registro para realizar el cargado.

**Carga y ejecución de un programa .COM**

Ya que la organización de un archivo .COM es relativamente fácil, el DOS sólo necesita saber que la extensión del archivo es .COM. Como se describió antes, un prefijo de segmento de

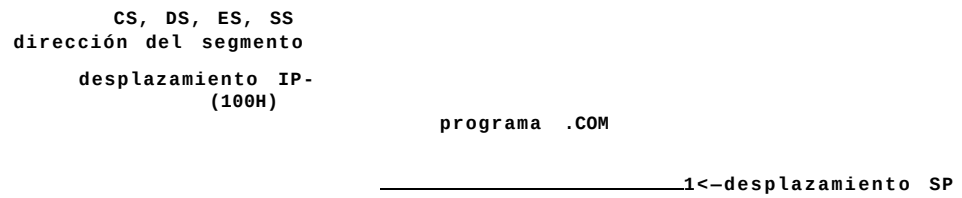



Figura 24-2 Inicialización de un programa .COM

programa precede a los programas .COM y .EXE cargados en memoria. Los primeros dos bytes del PSP contienen la instrucción INT 20H (regreso al DOS). Al cargar un programa .COM, el DOS:

- Establece los cuatro registros de segmento con la dirección del primer byte del PSP.
- Establece el apuntador de la pila (SP) al final del segmento de 64K, desplazamiento FFFEH (o al final de la memoria si el segmento no es lo bastante grande) y guarda en la pila una palabra con ceros.
- Establece el apuntador de instrucciones en 100H (el tamaño del PSP) y permite controlar para proceder a la dirección generada por CS:IP, la primera localidad inmediata posterior al PSP. Éste es el primer byte de su programa y debe contener una instrucción ejecutable. La figura 24-2 ilustra esta inicialización.

### Carga y ejecución de un programa .EXE

El enlazador almacena en disco un módulo .EXE que consta de dos partes: un *registro de encabezado* que contiene información de control y de reubicación, y el *módulo cargado* real.

El encabezado es un mínimo de 512 bytes y puede ser más grande si hay muchos elementos reubicables. El encabezado contiene información acerca del tamaño del módulo ejecutable, dónde será cargado en memoria, la dirección de la pila y los desplazamientos de reubicación que serán insertados para direcciones incompletas de máquina. En lo que sigue, el término *bloque* se refiere a un área de 512 bytes en memoria.

- 00-01H 4D5A hex ('MZ') identifica un archivo .EXE.
- 02-03H Número de bytes en el último bloque del archivo .EXE.
- 04-05H Tamaño del archivo, incluyendo el encabezado, en incrementos de bloque de 512 bytes. Por ejemplo, si el tamaño es 1,025, este campo contendría 2 y 02-03H contendría 1.
- 06-07H Número de elementos en la tabla de reubicación (véase 1CH).
- 08-09H Tamaño del encabezado, en incrementos de 16 bytes (párrafo), para ayudar al DOS a localizar el inicio del módulo ejecutable que sigue al encabezado. El número mínimo es 20H (32) (32 x 16 = 512 bytes).
- 0A-0BH Conteo mínimo de párrafos que deben residir por arriba del final del programa cuando es cargado,
- 0C-0DH Interruptor de cargar alta/baja. Cuando está enlazado, usted decide si el programa, para su ejecución, se carga en una dirección de memoria baja (lo usual) o en una alta. El número 0000H indica alta. De otra forma, esta localidad contiene el conteo máximo de párrafos que deben residir por arriba del final del programa cargado.
- 0E-0F Desplazamiento en el módulo ejecutable del segmento de la pila.

- 10-11H Desplazamiento que el cargador inserta en el registro SP cuando transfiere el control al módulo ejecutable. El valor es el tamaño definido de la pila.
- 12-13H Valor de la verificación de la suma: la suma de todas las palabras en el archivo (ignorando desbordamientos), usada como una verificación de validación por posibles datos perdidos.
- 14-15H Desplazamiento (por lo común, pero no necesariamente, 00H) que el cargador inserta en el registro IP cuando transfiere el control al módulo ejecutable.
- 16-17H Desplazamiento en el módulo ejecutable del segmento de código. El cargador inserta el desplazamiento en el registro CS. Si el segmento de código está primero, el desplazamiento sería cero.
- 18-19H Desplazamiento de la tabla de relocalización (véase el elemento en 1CH).
- 1A-1BH Número de traslape: cero (usual) significa que el archivo .EXE contiene el programa principal.
- ICH-al final Tabla de reubicación que contiene un número variable de reubicación de elementos, como se identifica en el desplazamiento 06-07H. Las posiciones 06-07H del encabezado indican el número de elementos en el módulo ejecutable que son reubicados. Cada elemento reubicado, empezando en el encabezado 1CH, consiste en un número de desplazamiento de dos bytes y un número de segmento de dos bytes.

El sistema construye bloques de memoria para el entorno y el segmento de programa. A continuación están los pasos que el DOS realiza cuando carga e inicializa un programa .EXE:

- Lee la parte formateada del encabezado y la envía a memoria.
- Calcula el tamaño del módulo ejecutable (tamaño total del archivo en la posición 04H, menos el tamaño del encabezado en la posición 08H) y lee el módulo a memoria en el segmento inicial.
- Lee los elementos de la tabla de reubicación y los envía a un área de trabajo y suma el valor de cada elemento al valor del segmento inicial.
- Establece los registros DS y ES con la dirección del segmento del PSP.
- Establece el registro SS con la dirección del PSP, más 100H (el tamaño del PSP), más el desplazamiento SS (en OEH). También, coloca en el registro SP el número 10H, el tamaño de la pila.
- Establece el CS con la dirección del PSP, más 100H (el tamaño del PSP), más el desplazamiento CS en el encabezado (en 16H) para el CS. Además, establece el IP con el desplazamiento en 14H. La pareja CS:IP proporciona la dirección inicial del segmento de código y, en realidad, de la ejecución del programa. La figura 24-3 ilustra esta inicialización.

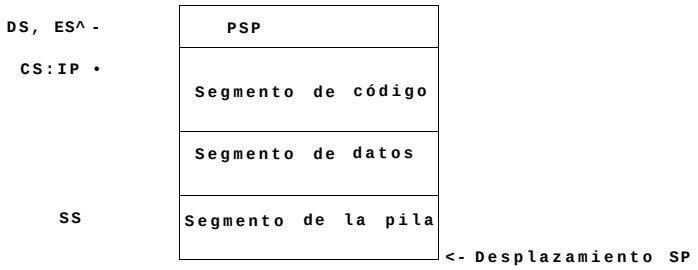



Figura 24-3 Inicialización de un programa .EXE

Después de lo anterior, el DOS ha terminado con el encabezado y los desecha. Los registros CS y SS están correctamente establecidos, pero su programa tiene que establecer el DS (y ES) para su propio segmento de datos:

```
MOV  AX, datasegname     ,-Coloca las direcciones de
MOV  DS,AX               ; los segmentos de datos en los
MOV  ES,AX               ; registros DS y ES
```

Ejemplo de carga de un programa .EXE

Considere el siguiente mapa que el enlazador generó para un programa .EXE:

```
Start     Stop   Length  Ñame         Class

00000H   0003AH   003BH  CSEG          Code
00040H   0005AH   001BH  DSEG          Data
00060H   0007FH   0020H  STACK         Stack

Program entry point at 0000:0000
```

El mapa proporciona la localidad *relativa* (no real) de cada uno de los tres segmentos. Observe que algunos enlazadores los acomodan en orden alfabético de nombre. De acuerdo con el mapa, el segmento de código (CSEG) inicia en 00000H; su posición relativa es el inicio del módulo ejecutable y su tamaño es de 003BH bytes. El segmento de datos, DSEG, inicia en 00040H y tiene un tamaño de 001BH. Ésta es la primera dirección a continuación de CSEG que se alinea con una frontera de párrafo (una frontera es divisible entre 10H). El segmento de la pila, STACK, inicia en 00060H, la primera dirección a continuación de DSEG que se alinea en una frontera de párrafo.

DEBUG no puede desplegar un registro de encabezado después de que un programa es cargado para su ejecución. El DOS reemplaza el registro del encabezado con el PSP. Sin embargo, existen varios programas de utilerías en el mercado (o puede escribir el suyo) que permiten ver el contenido hexadecimal de cualquier sector de disco. El encabezado para el programa que estamos examinando contiene la siguiente información relevante, de acuerdo con su localidad hexadecimal (el contenido de los campos está en secuencia inversa de byte):

OOH 4D5AH ("MZ").
02H Número de bytes en el último bloque: 5B00H.
04H Tamaño del archivo, incluyendo el encabezado, en bloques de 512 bytes: 0200H (0002 x 512 = 1,024 bytes).
06H Número de elementos en la tabla de reubicación siguiendo a la parte formateada del encabezado: 0100H, esto es, 0001.
08H Tamaño del encabezado, en incrementos de 16 bytes: 2000H (0020H = 32, y 32 x 16 = 512 bytes).
OCH Carga en memoria baja: FFFH.
OEH Desplazamiento del segmento de la pila: 6000H o 0060H.
10H Desplazamiento a insertar en el SP: 2000H, o 0020H.
14H Desplazamiento para IP: 0000H.
16H Desplazamiento para CS: 0000H.
18H Desplazamiento para la tabla de reubicación: 1E00H, o 001EH.

Cuando DEBUG cargó este programa, el registro contenía los valores siguientes:

```
SP = 0020   DS = 138F   ES = 138F
SS = 13A5   CS = 139F   IP = 0000
```

Para módulos .EXE, el cargador coloca la dirección del PSP en **el** DS y ES y en CS, IP, SS y SP los valores del registro del encabezado. Veamos ahora cómo el cargador inicializa estos registros.

### Registro CS

De acuerdo con el registro DS, cuando el programa se cargó, la dirección del PSP era 138F[0]H. Ya que el PSP es de 100H de tamaño, el módulo ejecutable sigue inmediatamente a 139F[0]H, que el cargador inserta en el registro CS:

| | |
|---|---|
| Dirección inicial del PSP (véase DS): | **138F0H** |
| Tamaño del PSP: | **+ 100H** |
| Dirección del segmento de código: | **139F0H** |

El CS proporciona la dirección de inicio de la parte de código del programa (CSEG). Puede usar el comando para desplegar de DEBUG, D CS:0000, para ver el código de máquina de un programa en memoria. El código es idéntico a la parte hexadecimal de la impresión .LST en ensamblador, a diferencia de los operandos que .LST marca como R.

### Registro SS

El cargador usó **el** número 60H en el encabezado (en 0EH) para colocar la dirección de la pila en el registro SS:

| | |
|---|---|
| Dirección inicial del PSP (véase DS): | **138F0H** |
| Tamaño del PSP: | **+ 100H** |
| Desplazamiento de la pila (véase localidad 0EH en **el** encabezado: | + **60H** |
| Dirección de la pila: | **13A50H** |

### Registro SP

El cargador usó 20H del encabezado (en 10H) para inicializar el apuntador de la pila al tamaño de la pila. En este ejemplo, la pila fue definida como 16 DUP(?), esto es, 16 campos de dos bytes = 32, o 20H. El SP apunta al tope actual de la pila.

### Registro DS

El cargador usa el registro DS para establecer el punto de inicio para el PSP en 138F[0]. Puesto que el encabezado no contiene una dirección inicial para el DS, su programa tiene que inicializarla:

```
0004 B8----------R      MOV AX,DSEG
0007 SE D8              MOV DS,AX
```

El ensamblador deja sin llenar la dirección de máquina de DSEG, que se convierte en una entrada en la tabla de reubicación en el encabezado, tratada anteriormente. DEBUG muestra la instrucción completada como

```
B8 A313
```

A313 es cargada en el DS como 13A3. La dirección DS es calculada como sigue:

| | |
|---|---|
| DirecciónCS: | 13 9F0H |
| Más desplazamiento para el DS: | 4OH |
| Dirección DS: | 13A30H |

Ahora tenemos estas cifras al inicio de la ejecución:

| REGISTRO | DIRECCIÓN | MAPA DE DESPLAZAMIENTO |
|---|---|---|
| CS | 139F[0]H | OOH |
| DS | 13A3[0]H | 40H |
| SS | 13AS[0]H | SOH |

Como ejercicio, rastree cualquiera de sus programas .EXE enlazados con DEBUG y observe los datos cambiados en los registros:

| INSTRUCCIÓN | REGISTROS CAMBIADOS |
|---|---|
| MOV AX, DSEG | IP y AX |
| MOV DS,AX | IP y DS |
| MOV ES,AX | IP y ES |

El DS ahora contiene la dirección correcta del segmento de datos. Puede usar D DS:00 para ver el contenido de DSEG y usar D SS:00 para ver el contenido de la pila.

## ASIGNACIÓN Y LIBERACIÓN DE MEMORIA

Los servicios del DOS le permiten *asignar, liberar* y *modificar el tamaño de* un área de memoria. Los usos más comunes para estos servicios son con los programas residentes y programas que cargan a otros programas para su ejecución. Bajo el DOS, que fue diseñado como un ambiente de un solo usuario, un programa que necesita cargar otro programa para ejecución tiene que liberar algún espacio de su memoria.

### INT 21H, función 48H: Asignación de memoria

Para asignar memoria para un programa, solicite la función 48H y en el BX coloque el número necesario de párrafos:

```
MOV  AH,48H           ;Petición para asignar memoria
MOV  BX,paragraphs    ,-Número de párrafos
INT  21H              ;Llama al DOS
```

Una operación exitosa pone en cero la bandera de acarreo y en el AX regresa la dirección del segmento del bloque de memoria asignado. La operación inicia en el primer bloque de memoria y pasa por cada bloque hasta que localiza un espacio lo bastante grande para la petición, generalmente en el final de la memoria alta.

Una operación no exitosa pone en uno la bandera de acarreo y regresa en el AX un código de error (07 = bloque de memoria destruido o bien 08 = memoria insuficiente) y en el BX el tamaño, en párrafos del bloque más grande disponible. Un bloque de memoria destruido significa que la operación encontró un bloque en el que el primer byte no era 'M' o 'Z'.

### INT 21H, función 49H: Libera memoria asignada

La función 49H libera memoria asignada; por lo común es usada para liberar un programa residente. Cargue en el ES la dirección del segmento del bloque que será regresado:

```
MOV  AH.49H           Petición para liberar memoria asignada
LEA  ES,seg-address   Dirección del bloque por párrafos
INT  21H              Llama al DOS
```

Una operación exitosa pone en cero la bandera de acarreo y almacena 00H en el segundo y tercer bytes del bloque de memoria, lo que significa que ya no está en uso. Una operación no exitosa pone en uno la bandera de acarreo y regresa en el AX un código de error (07 = bloque de memoria destruido o 09 = dirección no válida de bloque de memoria).

### INT 21H, función 4AH: Modifica el bloque de memoria asignada

La función 4AH puede aumentar o disminuir el tamaño de un bloque de memoria. Inicialice el BX con el número de párrafos conservados para el programa y el ES con la dirección del PSP:

```
MOV  AH,4AH           ,Petición para modificar la memoria asignada
MOV  BX,paragraphs    ;Número de párrafos
LEA  ES, PSP-address  ,-Dirección del PSP
INT  21H              ;Llama al DOS
```

Un programa puede calcular su propio tamaño restando el final del último segmento de la dirección del PSP. Asegúrese que usa el último segmento, si su enlazador reacomoda los segmentos en orden alfabético.

Una operación exitosa pone en cero la bandera de acarreo. Una operación no exitosa la pone en uno y regresa en el AX un código de error (07 = bloque de memoria destruido, 08 = memoria insuficiente y 09 = dirección no válida de bloque de memoria) y regresa en el BX el tamaño máximo posible (si **se** hizo un intento de aumentarlo). Una dirección errónea en el ES provoca un error 07.

## CARGA Y EJECUCIÓN DE UNA FUNCIÓN DE PROGRAMA

Examinemos ahora cómo hacer que un programa ejecutándose cargue y a su vez ejecute un subprograma. La función 4BH permite a un programa cargar un subprograma en memoria para ejecución. Cargue estos registros:

- AL = Código de la función para uno de lo siguiente:
  OOH = Cargar y ejecutar
  01H = Cargar un programa
  03H = Cargar traslape
  05H = Fijar estado de ejecución (no tratado en este texto)
- ES:BX = Dirección de un bloque de parámetro
- DX = Dirección del nombre de la ruta para el programa llamado, una cadena ASCIIZ en letras mayúsculas

El código para cargar el subprograma es como sigue:

```
MOV  AH,4BH              Petición para cargar
MOV  AL,code             Código de la función
LEA  BX,para-block       •Dirección del bloque de parámetros
LEA  DX.path             Dirección del nombre de la ruta
INT  21H                 Llama al DOS
```

Una operación no válida pone en uno la bandera de acarreo y regresa un código de error en el AX.

AL = OOH: Cargar y ejecutar

Esta operación carga un programa .EXE o.uno .COM en la memoria, establece un prefijo de segmento de programa para él y le transfiere el control para la ejecución. Como todos los registros, incluyendo la pila, son cambiados, la operación no es para inexpertos. El bloque de parámetros direccionado por el ES:BX tiene el formato siguiente:

| DESPLAZAMIENTO | OBJETIVO |
|---|---|
| OOH | Dirección del segmento entorno-bloque a ser pasado en PSP+2CH. Una dirección cero significa que el programa cargado es inherente al entorno del programa que lo carga. |
| 02H | Apuntador de palabra doble a una línea de comando para colocar en PSP + 80H. |
| 06H | Apuntador de palabra doble, por omisión FCB #1 para pasar en PSP+5CH. |
| OAH | Apuntador de palabra doble, por omisión FCB #2 para pasar en PSP+6CH. |

Los apuntadores de palabra doble tiene la forma de dirección desplazamiento: segmento.

AL = 01H: Cargar programa

Esta operación carga un programa .EXE o .COM en la memoria y establece un prefijo de segmento de programa para él, pero no le transfiere el control para la ejecución. El bloque de parámetros direccionado por el ES:BX tiene el formato siguiente:

| DESPLAZAMIENTO | OBJETIVO |
|---|---|
| OOH | Dirección del segmento entorno-bloque que es pasado en PSP+2CH. Si la dirección es cero, el programa cargado es inherente al entorno del programa que lo carga. |

| | |
|---|---|
| 02H | Apuntador de palabra doble a una línea de comando para colocar en PSP + 80H. |
| 06H | Apuntador de palabra doble, por omisión FCB #1 para pasar en PSP+5CH. |
| OAH | Apuntador de palabra doble, por omisión FCB #2 para pasar en PSP+6CH. |
| OEH | Dirección inicial de la pila. |
| 12H | Dirección inicial del segmento de código. |

Los apuntadores de palabra doble tienen la forma de dirección desplazamiento:segmento.

**AL = 03H: Cargar traslape**

Esta operación carga un programa o bloque de código, pero no establece un PSP o inicio de ejecución del programa o bloque. Por tanto el código requerido podría ser un programa traslapado. El bloque de parámetros direccionado por el ES:BX tiene el formato siguiente:

| DESPLAZAMIENTO | OBJETIVO |
|---|---|
| 00H | Palabra de la dirección del segmento en donde el archivo será cargado. |
| 02H | Palabra del factor de reubicación para aplicar a la imagen. |

Un error pone en uno la bandera de acarreo y regresa un código de error en el AX, como se describió en la figura 18-1.

**Programa: Cargar y ejecutar**

El programa en la figura 24-4 solicita al DOS que realice el comando DIR para la unidad D. El programa primero utiliza la función 4AH para reducir sus requerimientos de memoria a su tamaño real: la diferencia entre su último (ficticio) segmento ZNDSEG y el inicio de su PSP. Observe que en este momento el ES aún contiene la dirección del PSP, como se cargó al entrar. (Las instrucciones ASSUME anteriores y siguientes a MOV BX,SEG ZNDSEG aparecen porque son requeridas por MASM 5.1, pero no por algunos otros ensambladores.) El módulo es de 80 bytes, así que el PSP (10H párrafos) y el programa (8 párrafos) tienen un total de 18H párrafos.

La función 4BH con código 00 en el AL maneja la carga y ejecución del COMMAND.COM. El programa despliega las entradas del directorio para la unidad D.

**INT 21H, función 4DH: Obtiene el valor de regreso del subprograma**

Esta operación recupera el valor de regreso que el último subprograma envió cuando lo terminó la función 4CH o 31H. Los valores regresados son:

- AH = Método de terminación del subprograma, en donde
  00H = Terminación normal
  01H = Terminado por Ctrl+C
  02H = Error crítico de dispositivo
  03H = Terminado por la función 31H (mantiene el programa)
- AL = Valor de regreso desde el subprograma

**Carga y ejecución de una función de programa**

```
TITLE     P24EXDOS  (EXE)  Función 4BH del DOS para ejecutar DIR

SSEG      SEGMENT  PARA STACK 'Stack'
          DW       32 (?)
SSEG      ENDS

DSEG      SEGMENT  PARA  'Data'
PARAREA   LABEL    BYTE              /Bloque de parámetros para cargar/ej
          DW       0                 ;  dirección de la cadena de entorno
          DW       OFFSET DIRCOM     ;  apuntador a la línea de comando
          DW       DSEG              |
          DW       OFFSET FCB1       ;  apuntador al FCB1 por omisión
          DW       DSEG
          DW       OFFSET FCB2       ;  apuntador al FCB2 por omisión
          DW       DSEG
DIRCOM    DB       17, '/C" DIR D:',13 , 0
FCB1      DB       16 DUP(O)
FCB2      DB       16 DUP(O)
PROGNAM   DB       'D:COMMAND. COM',0
DSEG ENDS

CSEG      SEGMENT  PARA 'Code'
          ASSUME   CS:CSEG,DS:DSEG,SS:SSEG,ES:DSEG
BEGIN     PROC     FAR
          MOV      AH, 4AH           ;Reduce la memoria asignada
          ASSUME   CS:ZNDSEG
          MOV      BX,SEG ZNDSEG     ;Final del segmento
          ASSUME   CS:CSEG
          MOV      CX,ES             ;  menos inicio del
          SUB      BX,CX             ;  segmento del programa
          INT      21H
          JC       E10ERR            ;¿Hay espacio suficiente?
          MOV      AX,DSEG
          MOV      DS, AX            ;Establecer DS y ES
          MOV      ES.AX
          MOV      AH, 4BH           ;Petición para cargar
          MOV      AL, 00            |  y ejecutar
          LEA      BX,PARAREA        ;  COMMAND . COM
          LEA      DX,PROGNAM
          INT      21H
          JC       E20ERR            ¿Error en la ejecución?
          MOV      AL, 00            ;OK,  no hay código de error
          JMP      X10XIT
E10ERR:
          MOV      AL, 01            ;Código de error 1
          JMP      X10XIT
E20ERR:
          MOV      AL, 02            /Código de error 2
          JMP      X10XIT
X10XIT:
          MOV      AH,4CH            ;Petición
          INT      21H                 para salir al DOS
BEGIN     ENDP
CSEG      ENDS

ZNDSEG    SEGMENT                    ;Segmento mudo  (ficticio)
ZNDSEG    ENDS
          END      BEGIN
```

Figura 24-4  Ejecución de DIR desde un programa

## TRASLAPE DE PROGRAMAS

El programa en la figura 24-5 utiliza el mismo servicio que el de la figura 24-4, pero esta vez sólo carga un programa en memoria sin ejecutarlo. El proceso consiste en un programa principal, P24CALLV, y dos subprogramas, P24SUB1 y P24SUB2.

P24CALLV es el programa principal, con estos segmentos:

```
STACKSG  SEGMENT PARA STACK 'Stackl'

DATASG   SEGMENT PARA 'Datal'

CODESG   SEGMENT PARA 'Codel'

ZENDSG   SEGMENT                        ;Segmento ficticio (vacío)
```

P24SUB1 está enlazado y llamado por P24CALLV. Sus segmentos son:

```
DATASG  SEGMENT PARA 'Data2'

CODESG  SEGMENT PARA 'Code2'
```

Los segmentos de P24CALLV están ligados primero, porque sus clases difieren: 'Datal', 'Data2', 'Codel', 'Code2' y así sucesivamente. A continuación está el mapa de enlace para P24CALLV+P24SUB1:

```
Start   Stop    Length Ñame             Class
 00000H 0007FH 00080H STACKSG          Stackl
 00080H 000C2H 00043H DATASG           Datal
 000D0H 0016DH 0009EH CODESG           Codel
 00170H 00170H 00000H ZENDSG
 00170H 00185H 00016H DATASG           Data2
 00190H 001AFH 00020H CODESG           Code2
```

P24SUB2 también es llamado por P24CALLV, pero es enlazado por separado. Sus segmentos son:

```
DATASG  SEGMENT PARA 'Data'

CODESG  SEGMENT PARA 'Code'
```

El mapa de enlace de P24SUB2 se ve como:

```
Start   Stop    Length Ñame             Class
 00000H 00015H 00016H DATASG           Data
 00020H 0003EH 0001FH CODESG           Code
```

Cuando P24CALLV+P24SUB1 es cargado en memoria para su ejecución, P24CALLV llama y ejecuta P24SUB1 de manera normal. La llamada (CALL) cercana inicializa de manera

```
TITLE     P24CALLV (EXE)  Llama a un subprograma y lo traslapa
          EXTRN   P24SUB1:FAR

STACKSG   SEGMENT PARA STACK 'Stackl'
          DW      64 DUP(?)
STACKSG   ENDS

DATASG    SEGMENT PARA  'Datal'
PARABLK   LABEL   WORD              ;Bloque de parámetros
          DW      0
          DW      0
FILENAM   DB      •F:\P24SUB2.EXE',0
ERRMSG1   DB      'Modify mem error'
ERRMSG2   DB      'Allocate error  '
ERRMSG3   DB      'Seg cali error  '
DATASG    ENDS

CODESG    SEGMENT PARA  'Codel'
BEGIN     PROC    FAR
          ASSUME  CS:CODESG,DS:DATASG,SS:STACKSG
          MOV     AX,DATASG
          MOV     DS,AX
          CALL    Q10SCR            /Recorre la pantalla
          CALL    P24SUB1           ;Llama al subprograma 1

          MOV     AH, 4AH           /Comprime la memoria
          ASSUME  CS:ZENDSG
          MOV     BX,SEG ZENDSG     /Dirección del final del programa
          ASSUME  CS:CODESG
          MOV     CX, ES            /Dirección del PSP
          SUB     BX,CX             /Tamaño de este programa
          INT     21H
          JC      A30ERR            /Si hay error, salir

          MOV     AX,DS             /Inicializa ES para
          MOV     ES,AX             /  este servicio
          MOV     AH,48H            /Asigna memoria para traslapar
          MOV     BX,40             ;  hay 40 párrafos
          INT     21H
          JC      A40ERR            /Si hay error, salir
          MOV     PARABLK,AX        /Guarda la dirección del segmento

          MOV     AH.4BH            ;Carga el subprograma 2
          MOV     AL, 03            ;  sin ejecutarlo
          LEA     BX,PARABLK
          LEA     DX, FILENAM
          INT     21H
          JC      A50ERR            /Si hay error, salir
          MOV     AX,PARABLK        /Intercambiar dos palabras
          MOV     PARABLK+2,AX      /  de PARABLK
          MOV     PARABLK,2OH
          LEA     BX, PARABLK       /Pone el desplazamiento CS en 20H
          CALL    DWORD PTR [BX]
          JMP     A90               /Llama al subprograma 2
A30ERR:
          CALL    Q20SET            /Coloca el cursor
          LEA     DX,ERRMSG1
          CALL    Q30DISP           /Despliega mensaje
          JMP     A90
A40ERR:
          CALL    Q20SET            /Coloca el cursor
          LEA     DX,ERRMSG2
          CALL    Q30DISP           /Despliega mensaje
          JMP     A90
```

Figura 24-**5** Cómo llamar a un subprograma y traslaparlo

```
A50ERR:
        CALL    Q20SET          ;Coloca el cursor
        LEA     DX,ERRMSG3
        CALL    Q30DISP         /Despliega mensaje
        JMP     A90
A90:
        MOV     AH,4CH          /Sale
        INT     21H
BEGIN   ENDP
[               Servicio de la pantalla de video:
Q10SCR  PROC    NEAR
        MOV     AX,0600H        /Petición para recorrer
        MOV     BH,1EH          /Designa el atributo
        MOV     CX,0000
        MOV     DX,184FH
        INT     10H
        RET
Q10SCR  ENDP

Q20SET  PROC    NEAR
        MOV     AH,02H          /Petición para
        MOV     BH,00           /  colocar el cursor
        MOV     DH,12
        MOV     DL,00
        INT     10H
        RET
Q20SET  ENDP
Q30DISP PROC    NEAR            /Designa DX
        MOV     AH,40H          /Petición para desplegar
        MOV     BX,01           /Manejador
        MOV     CX,16           /Longitud
        INT     21H
        RET
Q30DISP ENDP
CODESG  ENDS

ZENDSG  SEGMENT                 /Segmento (vacío) mudo
ZENDSG  ENDS
        END     BEGIN
```

```
TITLE   P24SUB1  Subprograma llamado

DATASG  SEGMENT PARA 'Data2'
SUBMSG  DB      'Subprogram 1 reporting'
DATASG  ENDS

CODESG  SEGMENT PARA 'Code2'
P24SUB1 PROC    FAR
        ASSUME  CS:CODESG,DS:DATASG
        PUBLIC  P24SUB1
        PUSH    DS              /Guarda DS del llamador
        MOV     AX,DATASG       /Inicializa DS
        MOV     DS,AX
        MOV     AH,02H          /Petición para colocar
        MOV     BH,00           /  el cursor
        MOV     DH,05
        MOV     DL,00
        INT     10H
        MOV     AH,40H          /Petición para desplegar
        MOV     BX,01           ,-Manejador
        MOV     CX,22           /Longitud
```

Figura 24-5 (continuación)

```
            LEA     DX,SUBMSG
            INT     21H
            POP     DS                ;Restablece el DS para el llamador
            RET
P24SUB1     ENDP
CODESG      ENDS
            END

TITLE       P24SUB2  Subprograma llamado traslapado

DATASG      SEGMENT PARA 'Data'
SUBMSG      DB      'Subprogram 2 reporting'
DATASG      ENDS

CODESG      SEGMENT PARA 'Code'
P24SUB2     PROC    FAR
            ASSUME  CS:CODESG,DS:DATASG
            PUSH    DS                |Guarda DS del llamador
            MOV     AX,CS             |Establece la dirección del primer
            MOV     DS,AX             |  segmento en DS
            MOV     AH,02H            |Petición para colocar
            MOV     BH,00             |  el cursor
            MOV     DH,10
            MOV     DL,00
            INT     10H
            MOV     AH,40H            |Petición para desplegar
            MOV     BX,01             |Manej ador
            MOV     CX,22             |Longitud
            LEA     DX,SUBMSG
            INT     21H
            POP     DS                 Restablece el DS del llamador
            RET
P24SUB2     ENDP
CODESG      ENDS
            END
```

Figura 24-**5** (continuación)

correcta el IP, pero ya que P24SUB1 tiene su propio segmento de datos, tiene que guardar en la pila el DS del P24CALLV y establecer su propia dirección DS. P24SUB1 coloca el cursor, muestra un mensaje, saca de la pila el DS y regresa al P24CALLV.

Para traslapar P24SUB2 sobre P24SUB1, P24CALLV tiene que *comprimir* su propio espacio de memoria, ya que el DOS ha dado todo el espacio disponible. El segmento superior de P24CALLV es ZENDSG, que está vacío. P24CALLV resta la dirección de su PSP (aun en el ES) de la dirección de ZENDSG. La diferencia es 270H (27H párrafos), calculado como el tamaño del PSP (100H) más el desplazamiento de ZENDSG (170H), que es enviado al DOS para la función 4AH.

La función 48H del DOS asigna entonces memoria para dejar espacio y que P24SUB2 sea cargado (traslapado) en la parte superior de P24SUB1, puesto de manera arbitraria en 40H párrafos. La operación regresa la dirección cargada en el registro AX, que el P24CALLV almacena en PARABLK. Esta es la primer palabra de un bloque parámetro que utilizará la función 4BH. La función 4BH con código 03 en AL carga P24SUB2 en la memoria. Observe la definición en el segmento de datos: F:\P24SUB2.EXE,0. La función 4BH hace referencia a CS y PARABLK, la primera palabra contiene la dirección del segmento en donde el traslape será cargado y la segunda palabra es un desplazamiento, en este caso cero. Un diagrama puede ayudar a aclarar estos pasos:

| | Después de la carga inicial | | Después del servicio 4AH se comprime la memoria | | Después del servicio 48H se asigna memoria |
|---|---|---|---|---|---|
| 000 | PSP | 000 | PSP | 000 | PSP |
| 100 | P24CALLV | 100 | P24CALLV | 100 | P24CALLV |
| 270 | P24SUB1 | | | 270 | P24SUB2 |

La llamada (CALL) lejana a P24SUB2 requiere una referencia definida como IP:CS, pero PARABLK está en la forma CS:IP. Por lo tanto, el valor CS es movido a la segunda palabra y 20H es almacenado en la primer palabra para el IP, ya que el mapa de enlace muestra el valor como el desplazamiento del segmento de código de P24SUB2. La siguiente instrucción carga la dirección de PARABLK en el BX y llama a P24SUB2:

```
LEA   BX, PARABLK       /Dirección de PARABLK
CALL  DWORD PTR [BX]    /Llama a P24SUB2
```

Observe que P24CALLV no hace referencia a P24SUB2 por nombre en su segmento de código, así que no necesita la instrucción EXTRN especificando P24SUB2. Como P24SUB2 tiene su propio segmento de datos, primero guarda en la pila el DS e inicializa su propia dirección. Pero P24SUB2 no estaba enlazado con P24CALLV. Como resultado, la instrucción MOV AX,DATASG coloca en el AX sólo la dirección del desplazamiento de DATASG,0[0]H y no su dirección de segmento. Sabemos que CALL establece CS con la dirección del primer segmento, que (de acuerdo con el mapa) produce la dirección del segmento de datos. Mover el CS al DS da la dirección correcta en el DS. Note que si el código de P24SUB2 y de los segmentos de datos estuviera en una secuencia diferente, la codificación tendría que ser un poco distinta.

P24SUB2 coloca el cursor, muestra un mensaje, saca de la pila el DS y regresa a P24CALLV. DEBUG fue indispensable para desarrollar este programa.

## PROGRAMAS RESIDENTES

Varios programas populares y de shareware están diseñados para *residir* en memoria mientras otros corren: se puede activar sus servicios oprimiendo una secuencia especial de teclas. Se cargan los programas residentes después que el DOS y antes de activar otros programas de procesamiento normal. Casi siempre son programas .COM y también son conocidos como "programas residentes en memoria" (TSR; termina pero permanece residente).

La parte fácil de la escritura de un programa de éstos es hacer que resida. En lugar de la terminación normal, se sale por medio de la función 31H de la INT 21H (mantener el programa). La operación necesita el tamaño del programa en el registro DX:

```
MOV   AH,31H            /Petición para TSR
MOV   DX,prog-zize      /Tamaño del programa
INT   21H
```

Cuando se ejecuta la rutina de inicialización, el DOS reserva el bloque de memoria en donde el programa reside y carga los programas subsecuentes superior en la memoria.

La parte no tan fácil implica la activación del programa después de que queda residente, ya que no es un programa interno del DOS como CLS, COPY y DIR. Un enfoque común es modificar la tabla de servicios de interrupción de modo que los programas residentes interrumpan cualquier tecleo, actúen sobre un tecleo especial o una secuencia de tecleos y pasen por alto otros tecleos. El efecto es que un programa residente, por lo común, aunque no necesariamente, conste de las partes siguientes:

1. Una sección que redefina las localidades en la tabla de servicios de interrupción.
2. Un procedimiento de inicialización que ejecuta sólo la primera vez el programa y que realiza lo siguiente:
   - Reemplaza la dirección en las tablas de servicios de interrupción con su propia dirección.
   - Establece el tamaño de la parte del programa que permanece residente.
   - Utiliza una interrupción que le indica al DOS que termine la ejecución del programa actual y conecte a la memoria la parte especificada del programa.
3. Un procedimiento que permanezca residente y que es activado, por ejemplo, por una entrada especial desde el teclado o, en algunos casos, por un reloj.

En realidad, el procedimiento de inicialización configura todas las condiciones para hacer que el programa residente funcione y después se borre él mismo. La organización de la memoria ahora aparece como sigue:

- Resto de la memoria disponible
- Parte del programa de inicialización (traslapado con el siguiente programa)
- Parte residente del programa (permanece en memoria)
- COMMAND.COM
- IO.SYS y MSDOS.SYS
- Tabla de servicios de interrupción

Un programa residente puede utilizar las funciones de la INT 21H para accesar la tabla de servicios de interrupción, puesto que no hay seguridad de que computadoras más avanzadas tendrán la tabla de interrupción localizada en las mismas localidades de memoria.

### INT 21H, función 35H: Obtiene la dirección de interrupción

Para recuperar la dirección de una interrupción particular, cargue el AL con el número de la interrupción requerida:

```
MOV  AH.35H     ,-Petición de interrupción
.MOV  AL,int#   /Número de interrupción
INT  21H
```

La operación regresa la dirección de la interrupción en el ES:BX como segmento desplazamiento. Para memoria convencional, una petición para la dirección de la INT 09H regresa OOH en el ES y 24H (36) en el BX.

### INT 21H, función 25H: Establece dirección de interrupción

Para establecer una nueva dirección de interrupción, cargue el número de la interrupción en el AL y la nueva dirección en el DX:

```
MOV   AH,25H          Petición de dirección de interrupción
MOV   AL,#Ínt         Número de interrupción
LEA   DX.newaddr      Nueva dirección para la interrupción
INT   21H
```

La operación reemplaza la dirección actual de la interrupción con la nueva dirección. Entonces, en realidad, cuando la interrupción especificada ocurre, el proceso enlaza a su programa (residente) en lugar de a la dirección normal de interrupción.

### Ejemplo de un programa residente

El programa residente de la figura 24-6, llamado P24TSTNM, suena la bocina cuando utiliza el panel numérico y la tecla NumLock está activada. Su objetivo es avisar que está ingresando números en lugar de, digamos, presionar las teclas de flechas para mover el cursor. Este programa intercepta INT 09H, la entrada desde el teclado, para examinar la tecla presionada.

Los puntos siguientes acera del programa residente son de interés:

BIODATA define el segmento de datos del BIOS iniciando en 40[0]; en particular, el byte de la bandera del teclado, llamada aquí KBSTAT, que refleja el estado del teclado. El bit 5 activado (1) significa que la tecla NumLock está activada.

CODESG inicia el segmento de código de P24TSTNM. La primera instrucción ejecutable, JMP INITZE, transfiere la ejecución pasando la parte residente al procedimiento cercano INITZE en el final. Esta rutina utiliza primero CL para prevenir cualquier interrupción adicional que pudiera ocurrir en este momento. Después utiliza la función 35H del DOS para localizar la dirección de la INT 09H en las tablas de servicios de interrupción. La operación regresa la dirección en el ES:BX, que la rutina INITZE almacena en INT9SAV. Después, la función 25 H establece la dirección del propio programa para la INT 09H en la tabla de interrupciones, TESTNUM, el punto de entrada al programa residente. En realidad, el programa guarda la dirección de la INT 09H y la reemplaza con su propia dirección. El último paso establece el tamaño de la parte residente (todo el código hasta INITZE) en el DX y utiliza la función 31H del DOS (terminar pero permanecer residente) para salir. El código de INITZE al final traslapa el siguiente programa cargado para ejecución.

TESTNUM es el nombre del procedimiento residente que es activado cuando un usuario presione una tecla. El sistema transfiere la ejecución a la dirección de la INT 09H en la tabla de servicios de interrupción, que ha sido cambiada por la dirección de TESTNUM. Como la interrupción puede suceder cuando, por ejemplo, el usuario está en el DOS o en un editor o en un procesador de textos, P24TSTNM tiene que guardar los registros que usa. El programa accesa la bandera del teclado para determinar si NumLock está activada y si el teclado numérico fue presionado (un código de rastreo de teclado entre 71 y 83, inclusive). Si es así, el programa hace sonar la bocina. (El uso de la bocina se explicó en el capítulo 21, sección "Generación de sonido".) Las instrucciones finales implican restablecer los registros guardados en la pila —en orden inverso—

```
TITLE     P24TSTNM (COM)   Programa residente: verifica NumLock en
BIODATA   SEGMENT AT 40H           el área de datos del BIOS
          ORG     17H
KBSTAT    DB      ?                ;Byte de estado del teclado
BIODATA   ENDS
-------------------------------------------------------------------------
CODESG    SEGMENT PARA
          ASSUME  CS:CODESG,DS:BIODATA
          ORG     100H
BEGIN:
          JMP     INITZE           ;Salto a la inicialización
SAVINT9   DD      ?
TESTNUM:
          PUSH    AX               ;Guarda registros
          PUSH    CX
          PUSH    DS

          MOV     AX,BIODATA       |Dirección del segmento
          MOV     DS,AX            |  del área de datos del BIOS
          MOV     AL,KBSTAT        |Obtiene la bandera del teclado
          TEST    AL,00100000B     |¿NumLock?
          JZ      EXIT             |No, salir

          IN      AL,60H           Obtiene tecleos desde el puerto
          CMP     AL,71            ¿Código de rastreo < 71?
          JL      EXIT               sí, salir
          CMP     AL,83            ¿Código de rastreo > 83?
          JG      EXIT               sí, salir
                                   ,-Debe ser del teclado numérico
          MOV     AL,10110110B     Fijar frecuencia
          OUT     43H,AL
          MOV     AX,1000
          OUT     42H,AL
          MOV     AL,AH
          OUT     42H,AL
          IN      AL,61H           Activar la bocina
          MOV     AH,AL
          OR      AL,03
          OUT     61H,AL
          MOV     CX,5000          •Fijar duración
PAUSE:
          LOOP    PAUSE
          MOV     AL,AH            •Desactivar bocina
          OUT     61H,AL
EXIT:
          POP     DS               Restablecer registros
          POP     CX
          POP     AX
          JMP     CS:SAVINT9       Reasumir INT 09H
          Rutina de inicialización
-------------------------
INITZE:
          CLI                      |Prevenir interrupciones posteriores
          MOV     AH,35H           |Obtener dirección de la INT 09H
          MOV     AL,09            |  en ES:BX
          INT     21H
          MOV     WORD PTR SAVINT9,BX ;   y guardarla
          MOV     WORD PTR SAVINT9+2,ES

          MOV     AH,25H
          MOV     AL,09            ;Establecer nueva dirección para la     09H
          MOV     DX,OFFSET TESTNUM ;  en TESTNUM
          INT     21H
```

Figura 24-6 Programa residente

```
            MOV     AH,31H              ,-Petición para permanecer residente
            MOV     DX,OFFSET INITZE    ;Fijar tamaño de la parte residente
            STI                         ;Restablecer interrupciones
            INT     21H
CODESG      ENDS
            END     BEGIN
```

Figura 24-6 (continuación)

y pasar a INT9SAV, que contiene la dirección original de la INT 09H. Ahora liberamos en control para la interrupción.

El ejemplo siguiente ayudará a clarificar el procedimiento. Primero explicamos una operación convencional sin un TSR interceptando la interrupción:

1. Un usuario presiona una tecla y el teclado envía la interrupción INT 09H al BIOS.
2. El BIOS utiliza la dirección de la INT 09H en la tabla de servicios de interrupción para localizar su rutina de BIOS.
3. Entonces el control se transfiere a la rutina de BIOS.
4. La rutina obtiene el carácter y (si es un carácter estándar) lo envía al búfer del teclado.

A continuación está el proceso para el programa residente:

1. Un usuario presiona una tecla y el teclado envía la interrupción INT 09H al BIOS.
2. El BIOS utiliza la dirección de la INT 09H en la tabla de servicios de interrupción para localizar su rutina de BIOS.
3. Pero ahora la tabla contiene la dirección TESTNUM, el programa residente, al cual se transfiere el control.
4. Si NumLock está activada y el carácter es un número del teclado numérico, TESTNUM hace sonar la bocina.
5. TESTNUM sale por medio de un salto a la dirección original de la INT 09H guardada, que transfiere el control a la rutina del BIOS.
6. La rutina obtiene el carácter y (si es un carácter estándar) lo envía al búfer del teclado.

Como este programa tiene la intención de ser ilustrativo, puede modificarlo o expandirlo para sus propios objetivos. Algunos programas comerciales que también reemplazan la dirección en la tabla de la interrupción 09H no permiten el uso concurrente de un programa residente como éste.

**INT 21H,** función **34H:** Obtiene **la** dirección de **la** bandera ocupada del DOS

Aunque es utilizada de manera interna por el DOS, algunos TSR la usan cuando solicitan una interrupción-del DOS para verificar si otra interrupción está activa. Ya que el DOS no es reentrante (esto es, no puede volver a entrar el DOS mientras está activo), el TSR tiene que esperar hasta que el DOS no esté ocupado, como lo indica la bandera de DOS ocupado, inDOS.

```
MOV   AH,34H                ;Petición de ocupado
INT   21H                   ;Llama al DOS
CMP   ES:BYTE PTR[BX],0     ;Prueba si la bandera es cero
JE
```

El servicio regresa la dirección de inDOS en el ES:BX. La bandera contiene el número de funciones del DOS que están activas en ese momento, donde 0 significa ninguna. Puede ingresar el DOS sólo si inDOS es 0.

## PUNTOS CLAVE

- El registro de arranque está en la pista cero, sector 1, de cualquier disco que utilice FORMAT /S para formatearlo. Cuando inicia el sistema, carga de manera automática el registro de arranque del disco y lo envía a la memoria. Entonces, el registro de arranque carga el IO.SYS del disco a la memoria.
- IO.SYS es una interfaz de nivel bajo con las rutinas del BIOS en ROM. Al inicio, IO.SYS determina el estado de todos los dispositivos y equipo asociados con la computadora y establece la tabla de direcciones para las interrupciones hasta la 20H. También el IO.SYS maneja las E/S entre la memoria y los dispositivos externos.
- MSDOS.SYS es una interfaz de alto nivel para programas que están cargados en la memoria después del IO.SYS. Sus operaciones incluyen el establecimiento, la tabla de direcciones para interrupciones desde la 20H hasta la 3FH, la administración del directorio y de los archivos en disco, el manejo de bloqueo y desbloqueo de registros en disco y el manejo de las funciones de la INT 21H.
- COMMAND.COM maneja los distintos comandos del DOS y ejecuta los archivos .COM, .EXE y .BAT solicitados. Consiste en una pequeña parte residente, una parte de inicialización y una parte transitoria. COMMAND.COM es el responsable de cargar los programas ejecutables desde el disco y enviarlos a la memoria.
- El módulo .EXE que el enlazador crea consiste en un registro de encabezado que contiene la información de control y reubicación y el módulo real cargado.
- Al cargar un programa .COM o uno .EXE, el DOS configura bloques de memoria para el entorno y el segmento del programa. Precediendo a cada bloque de memoria está un encabezado de arena de 16 bytes iniciando en una frontera de párrafo. También el DOS crea un PSP en la localidad OOH del segmento del programa y lo carga en 100H.
- Al cargar un programa .COM, el DOS establece los registros de segmento con la dirección del PSP, coloca el apuntador de la pila al final del segmento, guarda en la pila una palabra con ceros y coloca el apuntador de instrucciones en 100H (el tamaño del PSP). Después, el control procede a la dirección generada por CS:IP, la primera localidad que sigue al PSP.
- Al cargar un programa .EXE, el DOS lee el registro de encabezado y lo envía a la memoria, calcula el tamaño del módulo ejecutable y lee el módulo en memoria en el segmento inicial. Suma el valor de cada elemento en la tabla de reubicación para el valor del segmento de inicio. Coloca la dirección del PSP en el DS y ES; en el SS coloca la dirección del PSP, más 100H, más el valor de desplazamiento SP; en el SP coloca el tamaño de la pila y en el CS la dirección del PSP, más 100H, más el valor de desplazamiento CS en el encabezado. También el DOS establece el IP con el desplazamiento en 14H. El par CS:IP proporciona la dirección inicial del segmento de código para la ejecución del programa.
- Los campos útiles dentro del PSP incluyen el área 1 de parámetros en 5CH, el área 2 de parámetros en 6CH y el área de transferencia a disco en 80H.
- Carga los programas residentes antes de activar otros programas de procesamiento normal. Sale por medio de la función 31H de la INT 21H, la cual requiere el tamaño del programa en el DX.

PREGUNTAS

24-1. (a) ¿En dónde está ubicado el registro de arranque? (b) ¿Cuál es su objetivo?

24-2. ¿Cuál es el objetivo de IO.SYS (IBMBIO.COM)?

24-3. ¿Cuál es la finalidad de MSDOS.SYS (IBMDOS.COM)?

24-4. Por lo general, ¿en dónde están, en la memoria, las siguientes partes de C0MMAND.COM y cuál es su objetivo? (a) Residente; (b) transitoria.

24-5. (a) ¿En dónde está ubicado el prefijo del segmento del programa? (b) ¿Cuál es su tamaño?

24-6. Un usuario teclea FUDGE C:ALF.DOC para pedir la ejecución de un programa FUDGE. Muestre el contenido hexadecimal en ei PSP del programa en (a) 5CH, el área 1 de parámetros (FCB #1), y (b) 80H, el DTA por omisión.

24-7. Su programa debe determinar si los comandos PATH están dispuestos para su entorno. Explique dónde puede el programa encontrar sus propio entorno. *(Nota:* La petición es para el entorno del programa, no para el entorno principal del DOS.)

24-8. Un programa .COM está cargado para su ejecución con su PSP iniciando en la localidad 2BA1[0]H. ¿Qué dirección almacena el DOS en cada uno de los siguientes registros (no tome en cuenta la notación inversa de bytes)?: (a) CS; (b) DS; (c) ES; (D) SS.

24-9. Un mapa de enlace para un programa .EXE se muestra a continuación:

| START | STOP | LENGTH | ÑAME | CLASS |
|---|---|---|---|---|
| 00000H | 0002FH | 00030H | STACK | STACK |
| 00030H | 0005BH | 0002CH | CODESG | CODE |
| 00060H | 0007CH | 0001DH | DATASG | DATA |

DOS carga el programa con el PSP comenzando en la localidad 1A25[0]H. Mostrando los cálculos donde sea apropiado, el estado de los contenidos de cada uno de los registros en el tiempo de la carga (no tome en cuenta la notación inversa de bytes): (a)CS; (b)DS; (c)ES; (d)SS; (e)SP.

24-10. Un encabezado de arena inicia en la localidad EB6[0] y contiene lo siguiente: 4D COOE OAOO .... (a) Para el DOS, ¿qué significa el 4D (M)? (b) Si éste fuera el último bloque de memoria, ¿en que diferiría el contenido? (c) ¿Cuál es la localidad de memoria del siguiente encabezado de arena? Muestre los cálculos.

24-11. (a) Los programas residentes por lo común interceptan las entradas desde el teclado. ¿Exactamente en dónde y cuál es esta dirección interceptada? (b) ¿En cuáles dos formas significativas difieren los códigos para la terminación de un programa residente y de un programa normal?

PARTE G — Capítulos de referencia

# CAPÍTULO 25

# Áreas de datos e interrupciones del BIOS

OBJETIVO

Describir las áreas de datos del BIOS y los servicios de interrupción del BIOS.

## INTRODUCCIÓN

El BIOS contiene un extenso conjunto de rutinas de entrada/salida y tablas que indican el estado de los dispositivos del sistema. El DOS y los programas usuarios pueden solicitar ratinas del BIOS para la comunicación con los dispositivos conectados al sistema. El método para realizar la interfaz con el BIOS es el de las interrupciones de software. Este capítulo examina las áreas de datos (o tablas) a las que el BIOS da soporte, el procedimiento de interrupción y varios servicios de interrrupción.

El capítulo cubre las interrupciones siguientes:

| | | | |
|---|---|---|---|
| OOH | División entre cero | OFH | Control de LPT1 |
| 01H | Un solo paso | 10H | Despliegue en video |
| 02H | Interrupción no enmascarable | 11H | Determinación del equipo |
| 03H | Punto de ruptura | 12H | Determinación del tamaño de la memoria |
| 04H | Desbordamiento | 13H | Entrada/salida de disco |
| 05H | Impresión de la pantalla | 14H | Comunicación de entrada/salida |

| | | | |
|---|---|---|---|
| 08H | Cronómetro del sistema | 16H | Entrada desde el teclado |
| 09H | Interrupción del teclado | 17H | Salida a la impresora |
| OBH | Control de COMÍ | 18H | Entrada a BASIC de ROM |
| OCH | Control de COM2 | 19H | Cargador de arranque |
| ODH | Control de LPT2 | 1AH | Leer y establecer |
| OEH | Control del disco flexible | 1BH | Tomar control en una interrupción de teclado |

## PROCESO DE ARRANQUE

En la PC, el ROM reside iniciando en la localidad FFFFOH. Al encender la computadora se provoca un "arranque en frío". El procesador ingresa un estado de restablecer, pone todas las localidades de la memoria en cero, realiza una verificación de la paridad de memoria y coloca FFFF[0]H en el registro CS y cero en el IP. Por lo tanto, la primera instrucción a ejecutar está en FFFF:0, el punto de entrada al BIOS. El BIOS también almacena el número 1234H en 40[0]:72H para señalar un Ctrl+Alt+Del subsecuente (rearranque), que no realiza la autoprueba precedente cuando se enciende.

El BIOS verifica los diferentes puertos para identificar e inicializar dispositivos que están conectados, incluyendo INT 11H (determinación del equipo) y la INT 12H (determinación del tamaño de la memoria). Después, empezando en la localidad 0 de memoria, el BIOS establece la tabla de servicios de interrupción que contiene las direcciones de las rutinas de interrupción.

Enseguida, el BIOS determina si está presente un disco que contenga el DOS, y si es así, ejecuta la INT 19H para accesar el primer sector de disco que contiene el cargador de arranque. Este programa es un sistema operativo temporal al cual la rutina del BIOS transfiere el control después de cargarlo en memoria. El cargador de arranque tiene una sola tarea: cargar en memoria la primer parte del sistema operativo real. Los archivos del DOS: IO.SYS, MSDOS.SYS y COMMAND.COM son cargados entonces desde el disco a la memoria.

## ÁREA DE DATOS DEL BIOS

El BIOS mantiene su propia área de datos de 256 bytes (100H) en memoria baja, empezando en la dirección de segmento 40[0]H. Un útil ejercicio es utilizar DEBUG para examinar estos campos. A continuación están listados por desplazamiento.

**Área de datos del puerto seriales**

00H-07H Cuatro palabras, direcciona hasta cuatro puertos seriales

**Área de datos del puerto paralelo**

08H-0FH Cuatro palabras, direcciona hasta cuatro puertos paralelos

**Área de datos del equipo del sistema**

10H-11H Estado del equipo, una indicación primitiva del estado de los dispositivos instalados. Puede emitir la INT 11H, que regresa lo siguiente en el AX:

| BIT | DISPOSITIVO |
|---|---|
| 15,14 | Número de puertos paralelos conectados |
| 11-9 | Número de adaptadores RS232 seriales |
| 7,6 | Número de dispositivos de discos flexibles: Bit 00 = 1, 01 = 2, 10 = 3 y 11 =4 |
| 5,4 | Modo de video inicial. Los valores de los bits son: |
| | 00 = no usado |
| | 01 = 40 x 25 color |
| | 10 = 80 **X** 25 color |
| | 11 = 80 x 25 monocromo |
| 2 | Dispositivo apuntador (ratón); 1 = instalado |
| 1 | 1 = coprocesador matemático está presente |
| 0 | 1 = unidad de disco flexible está presente |

**Área de datos varios**

12H Bandera de prueba del fabricante

**Área de datos del tamaño de la memoria**

13H-14H Cantidad de memoria en la tarjeta del sistema, en kilobytes
15H-16H Cantidad de expansión de memoria, en kilobytes

**Área 1 de datos del teclado**

17H-17H Primer byte del estado actual del shift:

| BIT | ACCIÓN | BIT | ACCIÓN |
|---|---|---|---|
| 7 | Insert activada | 3 | Alt presionada |
| 6 | CapsLock activada | 2 | Ctrl presionada |
| 5 | NumLock activada | 1 | Shift izquierdo presionado |
| 4 | Scroll Lock activada | 0 | Shift derecho presionado |

18H-18H Segundo byte del estado actual del shift:

| BIT | ACCIÓN | BIT | ACCIÓN |
|---|---|---|---|
| 7 | Insert presionada | 3 | Ctrl/NumLock presionada |
| 6 | CapsLock presionada | 2 | SysReq presionada |
| 5 | NumLock presionada | 1 | Alt izquierdo presionado |
| 4 | Scroll Lock presionada | 0 | Ctrl derecho presionado |

19H Entrada alterna de teclado para caracteres ASCII.
1AH-1BH Apuntador al inicio del búfer del teclado

1CH-1DH Apuntador al final del búfer del teclado
1EH-3DH Búfer del teclado (32 bytes)

**Área de datos de la unidad de discos flexibles**

3EH Estado de búsqueda en disco. Bit número 0 se refiere a la unidad A, 1 a la B, 2 a la C y 3 a la D. Un valor de bit 0 significa que la siguiente búsqueda es para reubicarse en el cilindro 0 para recalibrar la unidad.

3FH Estado del motor del disco. Si el bit 7 = 1, se está llevando a cabo una operación de escritura. Bit número 0 se refiere a la unidad A, 1 a B, 2 a C y 3 a la D; un valor 0 del bit significa que el motor está encendido.

40H Conteo del tiempo que tarda el motor hasta que se para

41H Estado del disco, la indicación de un error en disco en la última operación:

| | | | |
|---|---|---|---|
| 00H | No hubo error | 09H | Intento de hacer que el DMA cruce la frontera de los 64K |
| 01H | Parámetro no válido de unidad | OCH | Tipo de medio no encontrado |
| 02H | Marcador de dirección no encontrado | 10H | Error CRC en la lectura |
| 03H | Error de protección contra escritura | 20H | Error del controlador |
| 04H | Sector no encontrado | 40H | Falló la búsqueda |
| 06H | Disco flexible cambiado en la línea activa | 80H | Unidad no preparada |
| 08H | Sobrepasó la DMA | | |

42H-48H Estado del controlador del disco flexible

**Área de datos 1 de video**

49H Modo de video actual, indicado por un bit en uno:

| BIT | MODO | BIT | MODO |
|---|---|---|---|
| 7 | Monocromo | 3 | 80 X 25 color |
| 6 | 640 x 200 monocromo | 2 | 80 x 25 monocromo |
| 5 | 320 x 200 monocromo | 1 | 40 x 25 color |
| 4 | 320 x 200 color | 0 | 40 x 25 monocromo |

4AH-4BH Número de columnas en la pantalla
4CH-4DH Tamaño del búfer de la página de video
4EH-4FH Desplazamiento inicial del búfer de video
50H-5FH Ocho palabras para la posición actual para cada una de las ocho páginas, numeradas desde 0 hasta 7
60H-61H Línea inicial y final del cursor
62H Página de despliegue actualmente activa
63H-64H Dirección del puerto de despliegue activo, en donde monocromo es 3B4H y color es 3D4H
65H Configuración actual del registro del modo de video
66H Paleta de colores actual

**Área de datos del sistema**

| | |
|---|---|
| 67H-68H | Conteo de la hora y fecha |
| 69H-6AH | Registro de verificación de redundancia cíclica (CRC) |
| 6BH | Último valor de entrada |
| 6CH-6DH | Mitad inferior del cronómetro |
| 6EH-6FH | Mitad superior del cronómetro |
| 70H | Desbordamiento del tiempo (1 si el cronómetro pasó de la medianoche) |
| 71H | Ctrl+Break pone en uno el bit 7 |
| 72H-73H | Bandera de restablecer la memoria. Si el contenido es 1234H, Ctrl + ALt + Del provocan un rearranque (en lugar de un arranque) |

**Área de datos del disco duro**

| | |
|---|---|
| 74H | Estado de la última operación en el disco duro (mayores detalles en el capítulo 19) |
| 75H | Número de discos duros conectados |

**Área de datos de tiempo terminado**

| | |
|---|---|
| 78H-7BH | Tiempo terminado para los puertos paralelos (LPT1-LPT4) |
| 7CH-7FH | Tiempo terminado para los puertos seriales (COM1-COM4) |

**Área 2 de datos del teclado**

| | |
|---|---|
| 80H-81H | Direcciones de desplazamientos para el inicio del búfer del teclado |
| 82H-83H | Direcciones de desplazamientos para el final del búfer del teclado |

**Área 2 de datos del video**

| | |
|---|---|
| 84H | Número de renglones en la pantalla (menos 1) |
| 85H | Altura del carácter, en líneas de rastreo |
| 86H-8AH | Información varia de video |

**Área de datos del disco flexible/duro**

| | |
|---|---|
| 8BH-95H | Controlador y estado de error |

**Área 3 de datos del teclado**

96H Estado del modo del teclado y banderas de tecleo

| BIT | ACCIÓN | BIT | ACCIÓN |
|---|---|---|---|
| 7 | Identificación (ID) en progreso | 3 | Alt derecho presionado |
| 6 | Último código fue ACK | 2 | Ctrl derecho presionado |
| 5 | Si ID leído y KBX forzar NumLock | 1 | Último código de rastreo fue E0 |
| 4 | Teclado de 101/102 teclas instalado | 0 | Último código de rastreo fue El |

97H Banderas de los LED del teclado (bit 0 = ScrollLock, 1 = NumLock, y 2 = CapsLock)

**Área de datos del reloj de tiempo real**

98H-A7H Estado de las banderas de espera

**Área de datos del apuntador**

A8H-ABH Apuntadores a varias tablas de BIOS

**Área 2 de datos varios**

ACH-FFH Reservado para el DOS

## SERVICIOS DE INTERRUPCIÓN

Una interrupción es una operación que suspende la ejecución de un programa de modo que el sistema pueda realizar una acción especial. Ya hemos usado varias interrupciones de despliegue de video, E/S de disco, impresión y para programas residentes. La rutina de interrupción ejecuta y por lo regular regresa el control al procedimiento que fue interrumpido, el cual entonces reasume su ejecución. El BIOS maneja las interrupciones 00H-1FH y el DOS maneja las interrupciones 20H -3FH.

### Tabla de servicio de interrupción

Cuando la computadora se enciende, el BIOS y el DOS establecen una tabla de servicios de interrupción en las localidades de memoria 000H-3FFH. La tabla permite el uso de 256 (100H) interrupciones, cada una con un desplazamiento:segmento relativo de cuatro bytes en la forma IP:CS. El operando de una instrucción de interrupción tal como INT 05H identifica el tipo de solicitud. Como existen 256 entradas, cada una de cuatro bytes, la tabla ocupa los primeros 1,024 bytes de memoria, desde 00H hasta 3FFH. Cada dirección en la tabla relaciona a una rutina de BIOS o del DOS par un tipo específico de interrupción. Por lo tanto los bytes 0-3 contienen la dirección para la interrupción 0, los bytes 4-7 para la interrupción 1, y así sucesivamente:

```
INT 00H|INT 01H¡INT 02H|INT 03H|INT 04H|INT 05H|INT 06H|  ...
 IP:CS | IP:CS | IP:CS | IP:CS | IP:CS | IP:CS | IP:CS | ...
00H     04H     08H     OCH     10H     14H     18H
```

### Ejecución de una interrupción

Una interrupción guarda en la pila el contenido del registro de banderas, el CS, y el IP. Por ejemplo, la dirección en la tabla de INT 05H (que imprime la que se encuentra en la pantalla cuando el usuario presiona Ctrl + PrtSC) es 0014H (05H x 4 = 14H). La operación extrae la dirección de cuatro bytes de la posición 0014H y almacena dos bytes en el IP y dos en el CS. La dirección en el CS:IP entonces apunta al inicio de la rutina en el área del BIOS, que ahora se ejecuta. La interrupción regresa vía una instrucción IRET (Regreso de interrupción), que saca de la pila el IP, CS y las banderas y regresa el control a la instrucción que sigue al INT.

### Interrupciones externas e internas

Una interrupción *externa* es provocada por un dispositivo que es externo al procesador. Las dos líneas que pueden señalar interrupciones externas son la línea de interrupción no enmascarable

(NMI) y la línea de petición de interrupción (INTR). La línea NMI reporta la memoria y errores de paridad de E/S. El procesador siempre actúa sobre esta interrupción, aún si emite un CLI para limpiar la bandera de interrupción en un intento por deshabilitar las interrupciones externas. La línea INTR reporta las peticiones desde los dispositivos externos, en realidad, las interrupciones 05H a la OFH, para el cronómetro, el teclado, los puertos seriales, el disco duro, las unidades de disco flexible y los puertos paralelos.

Una interrupción *interna* ocurre como resultado de la ejecución de una instrucción INT o una operación de división que cause desbordamiento, ejecución en modo de un paso o una petición para una interrupción externa, tal como E/S de disco. Los programas por lo común utilizan interrupciones internas, que no son enmascarables, para accesar los procedimientos del BIOS y del DOS.

## INTERRUPCIONES DEL BIOS

Esta sección cubre las interrupciones del BIOS de la OOH a la 1BH. Existen otras operaciones que sólo pueden ser ejecutadas por el BIOS, y que no son tratadas aquí.

INT OOH: División entre cero. Llamada por un intento de dividir entre cero. Muestra un mensaje y por lo regular se cae el sistema. Los desarrolladores de programas están familiarizados con este error porque el borrado de un registro de segmento puede causarlo de manera accidental.

INT 01H: Un solo paso. Usado por DEBUG y otros depuradores para permitir avanzar por paso a través de la ejecución de un programa.

INT 02H: Interrupción no enmascarable. Usada para condiciones graves de hardware, tal como errores de paridad, que siempre están habilitados. Por tanto un programa que emite una instrucción CLI (limpiar interrupciones) no afecta estas condiciones.

INT 03H: Punto de ruptura. Usado por depuración de programas para detener la ejecución. Los comandos Go y Proceed colocan esta interrupción en el punto de detención apropiado en el programa; DEBUG deshace el modo de un solo paso y permite al programa ejecutarse de forma normal como hasta la INT 03H, en donde DEBUG restablece el modo de paso sencillo.

INT 04H: Desbordamiento. Puede ser causado por una operación aritmética, aunque por lo regular no realiza acción alguna.

INT 05H: Imprime pantalla. Hace que el contenido de la pantalla se imprima. Emita la INT 05H para activar la interrupción internamente, y presione las teclas Ctrl + PrtSC para activarla externamente. La operación permite interrupciones y guarda la posición del cursor. Ningún registro es afectado. La dirección 50:00 en el área de datos del BIOS contiene el estado de la operación.

INT 08H: Sistema del cronómetro. Una interrupción de hardware que actualiza la hora del sistema y (si es necesario) la fecha. Un chip temporizador programable genera una interrupción cada 54.9254 milisegundos, casi 18.2 veces por segundo.

INT 09H: Interrupción del teclado. Provocada por presionar o soltar una tecla en el teclado; descrita con detalle en el capítulo 11.

INT OBH, INT OCH: Control de dispositivo serial. Controla los puertos COM 1 y COM2, respectivamente.

**INT ODH, INT OFH:** Control de dispositivo paralelo. Controla los puertos LPT1 y LPT2, respectivamente.

**INT OEH:** Control de disco flexible. Señala actividad de disco flexible, como la terminación de una operación de E/S.

**INT 10H:** Despliegue en video. Acepta el número de funciones en el AH para el modo de pantalla, colocación del cursor, recorrido y despliegue; descrito en detalle en el capítulo 10.

**ENT 11H:** Determinación del equipo. Determina los dispositivos opcionales en el sistema y regresa el valor en la localidad 40:10H del BIOS al AX. (A la hora de encender el equipo, el sistema ejecuta esta operación y almacena el AX en la localidad 40:10H; véase la sección anterior, "Área de datos del BIOS", para mayores detalles.)

**INT 12H:** Determinación del tamaño de la memoria. En el AX, regresa el tamaño de la memoria de la tarjeta del sistema, en términos de kilobytes contiguos, tal que: memoria de 640K es 0280H, como se determinó durante el encendido.

**INT 13H:** Entrada/salida de disco. Acepta varias funciones en el AH para el estado del disco, sectores leídos, sectores escritos, verificación, formato y obtener diagnóstico; cubierto en el capítulo 19.

**INT 14H:** Las comunicaciones de Entrada/Salida. Proporciona una fila de bytes de E/S (esto es, un bit a la vez) al puerto de comunicación RS232. El DX debe contener el número del adaptador RS232 (0-3 para COMÍ, 2, 3 y 4, respectivamente). Varias funciones son establecidas por medio del registro AH.

***Función 00H:*** Inicialización del puerto de comunicaciones. Establece los parámetros siguientes en el AL de acuerdo con el número del bit:

| Vel. en baudios | Paridad | Bit de parada | Longitud de palabra |
|---|---|---|---|
| 7-5 | 4-3 | 2 | 1-0 |
| 000 = 110 | 00 = ninguna | 0 = 1 | 10 = 7 bits |
| 001 = 150 | 01 = impar | 1 = 2 | 1 1 = 8 bits |
| 010 = 300 | 10 = ninguna | | |
| 011 = 600 | 11 = par | | |
| 100 = 1,200 | | | |
| 101 = 2,400 | | | |
| 110 = 4,800 | | | |
| 111 = 9,600 | | | |

En el AX, la operación regresa el estado del puerto de comunicaciones. (Véase la función 03H para detalles.) A continuación está un ejemplo que establece COMÍ a 1,200 baudios, sin paridad, un bit de parada y longitud de 8 bit:

```
MOV  AH,00H          /Petición para inicializar el puerto
MOV  AL, 10000011B   ;Parámteros
MOV  DX,00           /Puerto serial COMÍ
INT  14H             /Llama al BIOS
```

***Función 01H:*** Carácter de transmisión. Cargue el AL con el carácter que la rutina transmite y el DX con el número de puerto. Al regreso, la operación coloca en el AH el estado del puerto. (Véase la función 03H.) Si la operación es habilitada para transmitir el byte, también establece el bit 7 del AH, aunque el propósito normal de este bit es reportar un error de tiempo terminado. Asegúrese de ejecutar la función OOH antes de utilizar este servicio.

***Función 02H:*** Carácter recibido. Cargue el número de puerto en el DX. La operación acepta un carácter desde la línea de comunicaciones y lo envía al AL. También establece el AH con el estado del puerto (véase la función 03H) para bits de error 7, 4, 3, 2 y 1. Por lo tanto un número diferente de cero en el AX indica un error de entrada. Asegúrese de ejecutar la función OOH antes de utilizar este servicio.

***Función 03H:*** Regresa estado del puerto de comunicaciones. Cargue el número del puerto en el DX. La operación regresa el estado de la línea en el AH y el estado del módem en el AL:

| AH (ESTADO DE LA LÍNEA) | AL (ESTADO DEL MÓDEM) |
|---|---|
| 7 Fuera de tiempo | 7 Línea recibida señal detectada |
| 6 Transmitir registro de corrimiento vacío | 6 Indicadores |
| 5 Trasmitir registro válido vacío | 5 Conjunto de datos listo |
| 4 Detección de ruptura | 4 Limpiar para enviar |
| 3 Error de enmarcado | 3 Delta recibe línea de señal de detección |
| 2 Error de paridad | 2 Detector anillo del borde posterior |
| 1 Error de sobrecorrida | 1 Conjunto de datos Delta listo |
| 0 Dato listo | 0 Limpiar Delta para enviar |

Otras funciones de la INT 14H son 04H (inicialización extendida) y 05H (control de puerto ampliado de comunicación).

INT 15H: Servicios del sistema. Esta muy elaborada operación proporciona un gran número de funciones en el AH, como las siguientes:

| | |
|---|---|
| 21H | Autoprueba de encendido |
| 43H | Lee el estado del sistema |
| 84H | Da soporte a palanca de juegos |
| 88H | Determina el tamaño de la memoria extendida |
| 89H | Conmuta el procesador al modo protegido |
| C2H | Interfaz del ratón |

Por ejemplo, con la función 88H en el AH, la operación regresa en el AX el número de kilobytes de memoria extendida. (Por ejemplo, 0580H significa 1408K bytes.) Como la operación sale sin reestablecer las interrupciones, úsela así:

```
MOV AH,88H      ;Petición de memoria extendida
INT 15H         ; desde el BIOS
STI             ,-Restablece las interrupciones
```

INT 16H: Entrada desde el teclado. Acepta varias funciones en el AH para entrada básica desde el teclado; cubierta en el capítulo 10.

**INT 17H:** Salida a la impresora. Proporciona varias funciones para la impresión vía el BIOS, estudiada en el capítulo 20.

**INT 18H:** Entrada al BASIC de ROM. Llamada al BIOS si el sistema inicia sin disco que contenga los programas de sistema del DOS.

**INT 19H:** Cargador de arranque. Si un dispositivo de disco (o de disco flexible) está disponible con los programas de sistema del DOS, lee la pista 0, sector 1, en la localidad de arranque en 7C00H y transfiere el control a esta localidad. Si no existe unidad de disco, transfiere la entrada al BASIC de ROM por medio de la INT 18H. Es posible usar esta operación como una interrupción de software; no limpia la pantalla o inicializa datos en el BIOS de ROM.

**INT 1AH:** Lee y pone la hora. Lee o pone la hora del día de acuerdo con el código de la función en el AH:

- 00H = Lee el reloj del sistema. Regresa la parte alta del contador en el CX y la parte baja en el DX. Si la hora pasó las 24 horas desde la última lectura, la operación coloca en el AL un valor diferente de cero.
- 01H = Pone la hora del reloj del sistema. Carga la parte alta del contador en el CX y la parte baja en el DX.
- 02H-07H. Esta función maneja la hora y fecha para los servicios del reloj de tiempo real.

Para determinar cuánto se tarda en ejecutar una rutina, podría poner en cero el reloj y después leerlo al terminar el procesamiento.

**INT 1BH: Tomar control en una interrupción de teclado.** Cuando son presionadas las teclas Ctrl+Break se provoca que el BIOS en ROM transfiera el control a su dirección de interrupción, en donde la bandera esté en uno.

## PUNTOS CLAVE

- La ROM reside empezando en la localidad FFFFOH. Al encender la computadora se provoca un "arranque en frió". El procesador ingresa un estado de restablecimiento, pone en cero todas las localidades de memoria, realiza una verificación de paridad de la memoria y establece el registro CS en FFFF[0]H y el IP en cero. Por lo tanto, la primer instrucción a ejecutar está en FFFF:0 o FFFFO, el punto de entrada al BIOS.
- Al arranque, el BIOS verifica los diferentes puertos para identificar e inicializar dispositivos que están conectados. Entonces el BIOS establece una tabla de servicios de interrupción, empezando en la localidad 0 de memoria, que contiene las direcciones para las interrupciones que aparezcan. Dos operaciones que realiza el BIOS son la determinación del equipo y del tamaño de la memoria. Si un disco con el DOS está presente, el BIOS accesa el primer sector del disco que contiene el cargador del arranque. Este programa carga los archivos del DOS: IO.SYS, MSDOS.SYS y COMMAND.COM desde el disco a la memoria.
- El BIOS mantiene su propia área de datos en memoria baja, empezando en la dirección del segmento 40[0]H. Las áreas relevantes de datos incluyen aquellas del puerto serial, el puerto paralelo, equipo del sistema, teclado, unidad de disco flexible, control de video, disco duro y reloj de tiempo real.

- El operando en la instrucción de interrupción, tal como INT 21H identifica el tipo de petición. Para cada uno de los 256 tipos posibles, el sistema mantiene una dirección de cuatro bytes en la tabla de servicios de interrupción en las localidades 0000H a 3FFH. Por lo tanto, los bytes 0-3 contienen la dirección para la interrupción 0, los bytes 4-7 para la interrupción 1, y así sucesivamente.
- Las interrupciones del BIOS van desde la OOH hasta la 1FH e incluyen división entre cero, impresión de pantalla, cronómetro, control de video, control de disco flexible, E/S de video, determinación de equipo y del tamaño de la memoria, E/S de disco, E/S de comunicaciones, entrada desde el teclado, salida a la impresora y cargador de arranque.

## PREGUNTAS

**25-1.** Distinguir entre una interrupción interna y una externa.

**25-2.** Distinguir entre una línea NMI y una línea INTR.

**25**-3. (a) ¿Cuál es la localidad de la memoria del punto de entrada al BIOS? (b) Al encender la computadora, ¿cómo es que el sistema se dirige a sí mismo a su dirección?

**25-4.** Al arranque, el BIOS realiza las interrupciones 11H, 12H y 19H. ¿Cuál es el objetivo?

**25-5.** ¿Cuál es ¡a localidad inicial del área de datos del BIOS?

**25-6.** Los siguientes números binarios fueron observados en el área de datos del BIOS. Para cada parte, identifique el campo y explique el significado del número.

(a) 10-11H: 10000010 00100101 (b) 17H: 11100001

(c) 18H: 00000011 (d) 96H: 00001100

**25**-7. Los siguientes números binarios fueron observados en el área de datos del BIOS. Para cada parte, identifique el campo y explique el significado del número.

(a) 00-03H: F8 03 F8 02 (b) 08-0BH: 78 03 00 00

(c) 13-14H: 80 02 (d) 15-16H: 00 08

(e) 4A-4BH: 50 00 (0 60-61H: 0E 0D

(g) 84H: 18

**25-8.** Identifique las siguientes interrupciones del BIOS: (a) División entre cero; (b) impresión de la pantalla; (c) interrupción del teclado; (d) despliegue en video; (e) E/S en disco; (f) entrada desde el teclado; (g) salida de impresora; (h) obtiene estado del equipo; (i) determinación del tamaño de la memoria; (j) E/S de comunicaciones.

## CAPITULO 26

# Interrupciones del DOS

OBJETIVO

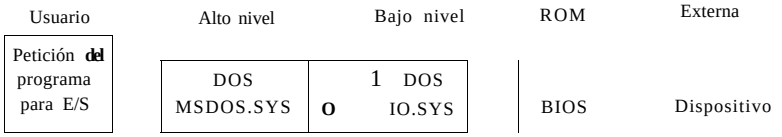
Describir las diferentes funciones de interrupción del DOS.

### INTRODUCCIÓN

Los dos módulos del DOS, 10.SYS y MSDOS.SYS, facilitan el uso del BIOS. Ya que proporcio-1 nan muchas de las pruebas adicionales necesarias, las operaciones del DOS por lo general son más j fáciles de usar que sus contrapartes del BIOS y por lo común son independientes de la máquina. |

IO.SYS es una interfaz de nivel bajo con el BIOS que facilita la lectura de datos desde j dispositivos externos hacia la memoria y la escritura de datos desde la memoria hacia dispositivos i externos. i

MSDOS.SYS contiene un administrador de archivos y proporciona varios servicios. Por) ejemplo, cuando un programa usuario solicita la INT 21H, la operación envía información al I MSDOS.SYS por medio del contenido de los registros. Para completar la petición, MSDOS.SYS J puede traducir la información a una o más llamadas a IO.SYS, el cual a su vez llama al BIOS. Las j siguientes son las relaciones implicadas: ¡



## INTERRUPCIONES DEL DOS

Las interrupciones desde la 20H hasta la 3FH están reservadas para operaciones del DOS, como se describe en las secciones siguientes.

**INT 20H:** Termina programa. Finaliza la ejecución de un programa .COM, restaura las direcciones para Ctrl + Break y errores críticos, limpia los búfer de registros y regresa el control al DOS. Esta función por lo regular sería colocada en el procedimiento principal y al salir de él, CS contendría la dirección del PSP. La terminación preferida es por medio de la función 4CH de la **INT 21H.**

**INT 21H:** Petición de función al DOS. La principal operación del DOS necesita una función en el AH y se describe con detalle más adelante.

**INT 22H:** Dirección de terminación. Copia la dirección de esta interrupción en el PSP del programa (en el desplazamiento OAH) cuando el DOS carga un programa para ejecución. A la terminación del programa, el DOS transfiere el control a la dirección de la interrupción. Sus programas no deben emitir esta interrupción.

**INT 23H:** Dirección del Ctrl + Break. Diseñada para transferir el control a una rutina del DOS (por medio del PSP desplazamiento OEH) cuando usted presiona Ctrl+Break o Ctrl+C. La rutina finaliza la ejecución de un programa o de un archivo de procesamiento por lotes. Un programa también puede cambiar esta dirección para que su propia rutina realice una acción especial sin terminar el programa. Sus programas no deben emitir esta interrupción.

**INT 24H:** Manejador de error crítico. Usada por el DOS para transferir el control (por medio del PSP desplazamiento 12H) cuando reconoce un error crítico (a veces en una operación de disco o de la impresora). Sus programas no deben emitir esta interrupción.

**INT 25H:** Lectura absoluta de disco. Lee el contenido de uno o más sectores de disco; estudiada en el capítulo 17, pero sustituida por la función 440DH código secundario 61H, de la INT21H.

**INT 26H:** Escritura absoluta en disco. Escribe información desde la memoria a uno o más sectores de disco; tratada en el capítulo 17, pero sustituida por la función 440DH, código secundario 41H de la INT21H.

**INT 27H:** Termina pero permanece residente (residente en memoria). Hace que un programa .COM al salir permanezca residente en memoria; sustituida por la función 31H de la INT 21H.

**INT 2FH:** Interrupción de multiplexión. Implica la comunicación entre programas, como la comunicación del estado de un spooler de la impresora, la presencia de un controlador de dispositivo o un comando del DOS tal como ASSIGN o APPEND. El capítulo 24 describe la función 4A01H, que verifica si hay espacio disponible en el área alta de memoria.

**INT 33H:** Manejador del ratón. Proporciona servicios para el manejo del ratón. (Véase el capítulo 21.)

## SERVICIO DE LA INT 21H DEL DOS

A continuación están las funciones del DOS solicitadas por la INT 21H, que requieren de una función codificada en el registro AH:

00H Termina programa. Básicamente la misma que la INT 20H y también reemplazada por la función 4CH de la INT 21H.
01H Entrada desde el teclado con repetición en la pantalla. (Véase el capítulo 11.)
02H Despliega un carácter. (Véase el capítulo 9.)
03H Entrada de comunicación. Lee un carácter desde el puerto serial y lo envía al AL. Éste es un servicio primitivo, y es preferida la INT 14H del BIOS.
04H Salida de comunicación. El DL contiene el carácter a transmitir. Es preferida la INT 14H del BIOS.
05H Salida a la impresora. (Véase el capítulo 20.)
06H Teclado y pantalla directos. (Véase el capítulo 11.)
07H Teclado directo sin repetición en la pantalla. (Véase el capítulo 11.)
08H Entrada desde el teclado sin repetición en la pantalla. (Véase el capítulo 11.)
09H Despliega cadena de caracteres. (Véase el capítulo 9.)
OAH Entrada a un búfer de teclado. (Véase el capítulo 11.)
OBH Verificar estado del teclado. (Véase el capítulo 11.)
OCH Limpiar el búfer del teclado y llamar a una entrada. (Véase el capítulo 11.)
ODH Restablecer unidad de disco. (Véase el capítulo 18.)
OEH Selecciona unidad de disco por omisión. (Véase el capítulo 18.)
OFH Abre un FCB de un archivo. (Véase el capítulo 17.)
10H Cierra un FCB de un archivo. (Véase el capítulo 17.)
11H Busca la primer entrada que coincida en un disco. Es obsoleta y reemplazada por la función 4EH.
12H Busca la siguiente entrada que coincida en un disco. Obsoleta y reemplazada por la función 4FH.
13H Borra un FCB de un archivo. Obsoleta y reemplazada por la función 41H.
14H Lee un registro secuencial de un FCB. (Véase el capítulo 17.)
15H Escribe un registro secuencial de un FCB. (Véase el capítulo 17.)
16H Crea un FCB de un archivo. (Véase el capítulo 17.)
17H Renombra un FCB de un archivo. Obsoleto y reemplazado por la función 56H.
19H Determina la unidad de disco por omisión. (Véase el capítulo 18.)
1AH Designa el área de transferencia de disco. (Véase el capítulo 17.)
1BH Obtiene información de la unidad por omisión. (Véase el capítulo 18.)
1CH Obtiene información de una unidad específica. (Véase el capítulo 18.)
1FH Obtiene bloque de parámetros de la unidad por omisión. (Véase el capítulo 18.)
21H Lee de forma directa un registro del FCB. (Véase el capítulo 17.)
22H Escribe de forma directa un registro del FCB. (Véase el capítulo 17.)
23H Obtiene el tamaño del FCB de un archivo. Obsoleta y reemplazada por la función 42H.
24HL Establece el campo de registro directo de un FCB. (Véase el capítulo 17.)
25H Establece la dirección de la tabla de interrupciones. (Véase el capítulo 24.) El ejemplo que sigue ilustra el uso de esta función. Cuando el usuario presiona las teclas Ctrl+Break o Ctrl+C, el procedimiento normal para el programa es terminar y regresar al DOS. Puede necesitar que su programa proporcione su propia rutina para manejar esta situación. El ejemplo utiliza la función 25H de la INT 21H, para establecer la dirección para Ctrl+Break en la tabla de interrupciones (INT 23H) a su propia rutina, C10BRK. La rutina podría reinicializar el programa o cualquier cosa que sea necesaria. El código es el siguiente:

```
        MOV     AH,25H      ;Petición establ. en tabla de interrup.
        MOV     AL,23H        para la interrupción 23H
        LEA     DX,C10BRK   ;Nueva dirección
        INT     21H         /Llama al DOS

C10BRK                      /Rutina para Ctrl+Break

        IRET                /Regreso de la interrupción
```

26H Crear un nuevo prefijo de segmento de programa. Reemplazada por la función 4B00H.

27H Lee directamente un bloque de disco. (Véase el capítulo 17.)

28H Escribe directamente un bloque de disco. (Véase el capítulo 17.)

29H Análisis gramatical del nombre de un archivo. (Véase el capítulo 18.)

2AH Obtiene la fecha del sistema. Regresa estos valores binarios:
AL = día de la semana (Domingo = 0)
CX = año (1980-2099)
DH = mes (01-12)
DL = día (01 -31)

2BH Fecha del sistema. Establece los siguientes valores binarios:
CX = año (1980, 2099)
DH = mes (01-12)
DL = día (01-31)
Al regresar, el AL indica válido (OOH) o no válido (FFH).

2CH Obtiene la hora del sistema. Regresa estos valores binarios:
CH = horas, en formato de 24 horas (00-23, medianoche = 00)
CL = minutos (00-59)
DH = segundos (00-59)
DL = centésimas de segundo (00-99)

2DH Pone la hora del sistema. Establece los siguientes valores binarios:
CH = horas, en formato de 24 horas (00-23, medianoche = 00)
CL = minutos (00-59)
DH = segundos (00-59)
DL = centésimas de segundo (00-99)
Al regresar, el AL indica válido (OOH) o no válido (FFH).

2EH Establece/reestablece la verificación de disco. (Véase capítulo 18.)

2FH Obtiene la dirección de la actual área de transferencia a disco (DTA). (Véase capítulo 17 y la función 1AH para establecer la dirección.)

30H Obtiene el número de la versión del DOS. Regresa estos valores:
AL = número principal de versión, como 7 para la versión 7.11
AH = número secundario de versión, como B hex (11) para la versión 7.11
BH = número de fabricante o bandera de la versión. Si la bandera de la versión es 08H, el DOS se ejecuta en ROM.

BL:CX = cero o número de serie, en 24 bits, del usuario (dependiente del fabricante). Véase también la función 3306H.

31H Termina pero permanece residente. (Véase capítulo 24.)

32H Obtiene bloque de parámetros de la unidad (DPB). (Véase capítulo 18.)

3300H Obtiene estado de Ctrl+C. Si la bandera de Ctrl+C está apagada (0), el DOS verifica por Ctrl+C sólo mientras maneja funciones de E/S de caracteres, 01H-0CH. Si la bandera está activada (1), el DOS también verifica mientras maneja otras funciones. Para obtener el estado, ponga la subfunción 00H en el AL. El valor regresado en el DL es 00H = verificación deshabilitada o 01H = verificación habilitada.

3300H Obtiene estado de Ctrl+C. Si la bandera de Ctrl+C está apagada (0), el DOS verifica por Ctrl+C sólo mientras maneja funciones de E/S de caracteres, 01H-0CH. Si la bandera está activada (1), el DOS también verifica mientras maneja otras funciones. Para establecer el estado, ponga la subfunción 01H en el AL y establezca el estado en el DL como 00H = deshabilita verificación o 01H = habilita verificación.

3305H Obtiene unidad de arranque (disponible desde DOS 5). La operación regresa en el DL la unidad (1 = A, etc.) usada para cargar el DOS.

3306H Obtiene la versión del DOS (disponible desde DOS 5). La operación regresa:

BL = número principal de versión, como 7 para la versión 7.11

BH = número secundario de versión, como B hex (11) para la versión 7.11

DL = número de revisión en los bits 2-0

DH = bandera de versión del DOS (indica si el DOS está corriendo en memoria convencional, en área de memoria alta, o en ROM)

Aunque el comando SETVER del DOS puede falsear el número de versión del DOS, la función 3306H envía la versión verdadera.

34H Obtiene la dirección de la bandera de DOS ocupado (inDOS). (Véase el capítulo 24.)

35H Obtiene la dirección de la tabla de interrupción. (Véase el capítulo 24.)

36H Obtiene el espacio libre en disco. (Véase el capítulo 18.)

38H Obtiene/establece información dependiente del país. Da soporte a varias funciones concernientes a información específica de varios países, como el símbolo y formato monetario del país, separadores de millares y lugares decimales y separadores de fecha y hora. Carga el DX para la operación:

DX = FFFFH: Establece el código del país que el DOS usa hasta que se indique otra cosa.

DX = cualquier otro número: Obtiene el código actual del país en uso.

39H Crea subdirectorio (MKDIR). (Véase el capítulo 18.)

3AH Elimina subdirectorio (RMDIR). (Véase el capítulo 18.)

3BH Cambia de directorio (CHDIR). (Véase el capítulo 18.)

3CH Crea un archivo con manejador. (Véase el capítulo 17.)

3DH Abre archivo con manejador. (Véase el capítulo 17.)

3EH Cierra archivo con manejador. (Véase el capítulo 17.)

3FH Lee archivo/dispositivo. (Véase capítulos 9 y 17.)

40H Escribe archivo/dispositivo con manejador. (Véase capítulos 9, 17 y 20.)

41H Borra archivo desde el directorio. (Véase el capítulo 18.)

42H Mueve el apuntador del archivo. (Véase el capítulo 17.)

| | |
|---|---|
| 43H | Examina/cambia el atributo del archivo. (Véase el capítulo 18.) |
| 44H | Control de E/S para dispositivos. Da soporte a un extenso conjunto de subfunciones para examinar dispositivos y leer y escribir datos, listadas en las funciones siguientes: |
| 4400H | Obtiene información de dispositivo. (Véase el capítulo 18.) |
| 4401H | Establece información de dispositivo. (Véase el capítulo 18.) |
| 4404H | Lee datos de control desde la unidad. (Véase el capítulo 18.) |
| 4405H | Escribe datos de control a la unidad. (Véase el capítulo 18.) |
| 4406H | Examina el estado de la entrada. (Véase el capítulo 18.) |
| 4407H | Examina estado de la salida. (Véase el capítulo 18.) |
| 4408H | Determina si el medio es removible para el dispositivo. (Véase el capítulo 18.) |
| 440DH, | Código secundario 41H escribe sector en disco. (Véase el capítulo 18.) |
| 440DH, | Código secundario 61H lee sector en disco. (Véase el capítulo 18.) |
| 440DH, | Código secundario 42H formatea pista. (Véase el capítulo 18.) |
| 440DH, | Código secundario 46H establece identificación del medio. (Véase el capítulo 18.) |
| 440DH, | Código secundario 60H obtiene parámetros del dispositivo. (Véase el capítulo 18.) |
| 440DH, | Código secundario 66H obtiene identificación del medio. (Véase el capítulo 18.) |
| 440DH, | Código secundario 68H sensible al tipo de medio. (Véase el capítulo 18.) |
| 45H | Duplica un manejador de archivo. (Véase el capítulo 18.) |
| 46H | Fuerza la duplicación de un manejador. (Véase el capítulo 18.) |
| 47H | Obtiene directorio actual. (Véase el capítulo 18.) |
| 48H | Asigna bloque de memoria. (Véase el capítulo 24.) |
| 49H | Libera bloque de memoria asignado. (Véase el capítulo 24.) |
| 4AH | Establece tamaño del bloque de memoria asignado. (Véase el capítulo 24.) |
| 4BH | Carga/ejecuta un programa. (Véase el capítulo 24.) |
| 4CH | Termina un programa. (Véase el capítulo 4.). Ésta es la forma estándar de terminar un programa. |
| 4DH | Recupera el código de regreso de un subproceso. (Véase el capítulo 24.) |
| 4EH | Encuentra primera entrada del directorio que coincida. (Véase el capítulo 18.) |
| 4FH | Encuentra siguiente entrada del directorio que coincida. (Véase el capítulo 18.) |
| 50H | Establece la dirección de un prefijo de segmento de programa (PSP). Cargue el BX con el desplazamiento del PSP para el programa actual. Ningún valor es regresado. |
| 51H | Obtiene la dirección de un prefijo de segmento de programa (PSP). Regresa la dirección de desplazamiento del PSP para el programa actual. (Véase el capítulo 24.) |
| 52H | Obtiene la dirección de la lista interna del DOS. (Véase el capítulo 24.) |
| 54H | Obtiene estado de verificación. (Véase el capítulo 18.) |
| 56H | Renombra un archivo. (Véase el capítulo 18.) |
| 57H | Obtiene/pone la fecha y hora de un archivo. (Véase el capítulo 18.) |
| 5800H | Obtiene la estrategia de asignación de memoria. (Véase el capítulo 24.) |
| 5801H | Establece la estrategia de asignación de memoria. (Véase el capítulo 24.) |
| 5802H | Obtiene enlace con la memoria superior. (Véase el capítulo 24.) |
| 5803H | Establece enlace con la memoria superior. (Véase el capítulo 24.) |
| 59H | Obtiene código de error extendido. (Véase el capítulo 18.) |
| 5AH | Crea un archivo temporal. (Véase el capítulo 18.) |
| 5BH | Crea un archivo nuevo. (Véase el capítulo 18.) |

5CH Bloquea/desbloquea el acceso a archivo. Usado en entornos de redes y de multitareas.

5DH Establece error extendido. Cargue el DX con la dirección de desplazamiento de una tabla de información de errores. La tabla será recuperada por la siguiente ejecución de la función 59H (obtener código de error extendido: para detalles véase la función 59H en el capítulo 18).

5EH Servicios de redes de área local. Una subfunción en el AL especifica el servicio:
00H Obtiene el nombre de la máquina
Q2H Establece configuración de la impresora
03H Obtiene configuración de la impresora

5FH Servicios de redes de área local. Una subfunción en el AL especifica el servicio:
02H Obtiene una entrada de la lista de asignación
03H Crea la conexión con la red
04H Cancela la conexión con la red

62H Obtiene la dirección del PSP. (Véase la función 51H para una operación idéntica.)

65H Obtiene información ampliada del país. Da soporte a varias subfunciones concernientes a información específica de varios países.

66H Obtiene/establece página global de código.

67H Establece el número máximo de manejadores. (Véase el capítulo 24.)

68H Archivo commit. (Véase el capítulo 18.)

6CH Extensión de abrir archivo. Combina las funciones 3CH (crear archivo), 3DH (abrir archivo) y 5BH (crear archivo único) . (Véase el capítulo 18.)

## PUNTOS CLAVE

- Interrupciones desde la 20H hasta la 3FH están reservadas para operaciones con el DOS.
- La INT 21H maneja operaciones como entrada desde el teclado, salida a la pantalla, salida a la impresora, restablecer el disco, abrir/cerrar archivos, borrar archivo, leer/escribir registros secuenciales, leer/escribir registros directos, terminar y permanecer residente (residente en memoria), crear subdirectorios y terminar un programa.

## PREGUNTAS

26-1. ¿Qué interrupciones están reservadas para el DOS?

26-2. Identifique las funciones para los siguientes servicios de la INT 21H del DOS: (a) entrada de comunicaciones; (b) obtener la hora del sistema; (c) obtener la versión del DOS; (d) terminar pero permanecer residente (residente en memoria);(e) obtener la dirección de la tabla de interrupciones; (f) crear un subdirectorio; (g) obtener la cantidad de espacio libre en el disco; (h) obtener la dirección del PSP.

26-3. Identifique las siguiente funciones de la INT 21H: (a) 05H; (b) OAH; (c) OFH; (d) 16H; (e) 35H; (f) 3CH; (g) 3DH; (h) 3FH; (i) 40H.

# CAPÍTULO 27

# Operadores y directivas

OBJETIVO

Describir con detalle los operadores y directivas del lenguaje ensamblador.

## INTRODUCCIÓN

Las diferentes características del lenguaje ensamblador al principio tienden a ser un poco confusas. Pero una vez que ya se ha familiarizado con las características más sencillas y comunes descritas en los capítulos precedentes, usted debe encontrar las descripciones en este capítulo más fáciles de entender y como una referencia a mano. Aquí, describimos los diferentes tipos de especificadores, operadores y directivas. El manual del lenguaje ensamblador contiene unas cuantas características más de uso marginalmente útil.

## ESPECIFICADORES DE TIPO

Los especificadores de tipo proporcionan el tamaño de una variable de datos o la distancia relativa de una etiqueta de instrucción. Los especificadores de tipo que dan el tamaño de una variable de datos son BYTE, WORD, DWORD, FWORD, QWORD y TBYTE. Aquellas que dan la distancia de una etiqueta de instrucción son NEAR, FAR y PROC. Una dirección cercana, la cual sólo es un desplazamiento, se supone que es en el segmento actual; una dirección lejana, que consiste en una dirección segmento:desplazamiento, puede ser usada para acceso a otro segmento.

Los operadores PTR y THIS, así como las directivas COM, EXTRN, LABEL y PROC, utilizan especificadores de tipo.

## OPERADORES

Un operador proporciona una facilidad para cambiar o analizar operandos durante un ensamblado. Los operadores están divididos en varias categorías:

- Operadores de cálculo: Aritméticos, índice, lógicos, desplazamiento y nombre de estructura de campo.
- Operadores de macro: Varios tipos, tratados en el capítulo 22.
- Operadores de registro: MASK y WIDTH, tratados más adelante en este capítulo, con la directiva RECORD.
- Operadores relaciónales: EQ, GE, GT, LE, LT y NE.
- Operadores de segmento: OFFSET, SEG y pasar por alto el segmento.
- Operadores de tipo (o atributo): HIGH, HIGHWORD, LENGTH, LOW, LOWWORD, PTR, SHORT, SIZE, THIS y TYPE.

Como el conocimiento de estas categorías no es necesario, sólo trataremos los operadores en orden alfabético.

### Operadores aritméticos

Los operadores aritméticos incluyen los conocidos signos aritméticos y realizan aritmética durante un ensamble. En la mayoría de los casos, podría realizar los cálculos usted mismo, aunque la ventaja de usar estos operadores es que cada vez que cambie el programa y lo reensamble, el ensamblador de manera automática recalcula los valores de los operadores aritméticos. A continuación está una lista de los operadores, junto con un ejemplo de su uso y el efecto obtenido:

| SIGNO | TIPO | EJEMPLO | EFECTO |
|---|---|---|---|
| + | Adición | FLD1+25 | Suma 25 a la dirección de FLD1 |
| + | Positivo | +FLD1 | Trata FLD1 como positivo |
| - | Resta | FLD2-FLD1 | Calcula la diferencia entre dos direcciones de desplazamiento |
| | Negación | -FLD1 | Invierte el signo |
| * | Multiplicación | núm*3 | Multiplica el número por 3 |
| / | División | núm/3 | Divide el número entre 3 |
| MOD | Residuo | núml MOD núm2 | Envía el residuo de núml/núm2 |

Salvo para la adición (+) y resta (-), todos los operadores deben ser constantes enteras. Los siguientes ejemplos relacionados de expresiones enteras son ilustrativos:

```
valorl = 12 * 4          ;48
valorl = valorl / 6      ;48 / 6 = 8
valorl = valorl - 3      ;(-8) - (3) = -11
```

### Operadores HIGH y HIGHWORD

El operador HIGH regresa el byte alto (de la izquierda) de una expresión y HIGHWORD (desde MASM 6.0) regresa la palabra alta de una expresión. (Véase también el operador LOW.) Aquí está un ejemplo:

```
EQUVAL: EQU  1234H

        MOV  CL, HIGH EQUVAL     ; Carga 12H en CL
```

### Operadores INDEX

Para una referencia directa a memoria, un operando de una instrucción especifica el nombre de una variable definida, como es mostrado por COUNTER en la instrucción ADD CX,COUNTER. Durante la ejecución, el procesador localiza la variable especificada en la memoria por combinación del valor del desplazamiento de la variable con la dirección del segmento de datos en el DS.

Para direccionamiento indirecto de la memoria, un operando hace referencia a un registro base o índice, constante, variables de desplazamiento y variables. El operador de índice, que utiliza corchetes, actúa como un signo más (+). Un uso común de indexamiento es para hacer referencia a datos en tablas. Puede usar las operaciones siguientes para referenciar memoria indexada:

- [Constante], i.e, un número o nombre inmediato entre corchetes. Por ejemplo, cargue la quinta entrada de TABLEA en el CL (note que TABLEAfO] es la primer entrada):

```
TABLEA  DB   25 DUP(?)       ;Tabla definida

        MOV  CL, TABLEA[4]   ,-Obtiene la quinta entrada de TABLEA
```

- Registro base BX como [BX] en asociación con el registro de segmento DS y el registro base BP como [BP] en asociación con el registro de segmento SS. Por ejemplo, utilice la dirección de desplazamiento en el BX (combinada con la dirección de segmento en el registro DS) y mueva el elemento referenciado al DX:

```
MOV DX,[BX]      /Registro base DS:BX
```

- Registro índice DI como [DI] y registro índice SI como [SI], ambos en asociación con el registro de segmento DS. Por ejemplo, combine la dirección en el DS con la dirección de desplazamiento en el SI y mueva el elemento referenciado al AX:

```
MOV AX, [SI]  /Registro índice DS:SI
```

- Registros índice combinados. Por ejemplo, mueva el contenido del AX a la dirección determinada por la suma de la dirección DS, el desplazamiento BX, el desplazamiento SI y la constante 4:

```
MOV [BX+SI+4], AX  /Base + índice + constante
```

El ejemplo anterior también podría codificarlo como [BX+SI]+4. Puede combinar estos operandos en cualquier orden, pero no combinar dos registros base [BX+BP] o dos registros índice [DI + SI]. Sólo los registros índice deben estar entre corchetes.

### Operador LENGTH

El operador LENGTH regresa el número de entradas definidas por un operador DUP. La siguiente instrucción MOV regresa la longitud 10 al DX:

```
TABLEA  DW   10 DUP(?)

        MOV  DX,LENGTH TABLEA
```

Si el operando referenciado no contiene una entrada DUP, el operador regresa el número 01. (Véase también los operadores SIZE y TYPE.)

### Operadores lógicos

Los operadores lógicos realizan las operaciones lógicas en los bits de una expresión:

| OPERADOR | USADO COMO | EFECTO |
|---|---|---|
| AND | expl AND exp2 | Conjunción de los bits |
| OR | expl OR exp2 | Disyunción de los bits |
| XOR | expl XOR exp2 | Disyunción exlusiva de los bits |
| NOT | NOT expresión! | Invierte los bits |

A continuación están unos ejemplos:

```
MOV  AL,00111100B AND 01010101B     ;00010100B

MOV  BL,NOT 01010101B               ;10101010B
```

### Operadores LOW y LOWWORD

El operador LOW regresa el byte bajo (de la derecha) de una expresión y LOWWORD (desde MASM 6.0) regresa la palabra baja de una expresión. (También véase el operador HIGH.) Aquí está un ejemplo:

```
EQUVAL  EQU  1234H

        MOV  CL.LOW EQUVAL   ;Carga 34H en CL
```

### Operador OFFSET

El operador OFFSET regresa la dirección de desplazamiento (esto es, la dirección relativa dentro del segmento de datos o del segmento de código) de una variable o de una etiqueta. El formato general es

```
OFFSET variable o etiqueta
```

El siguiente MOV regresa la dirección de desplazamiento de TABLEA:

```
MOV DX, OFFSET TABLEA
```

Note que LEA no requiere de OFFSET para regresar el mismo valor:

```
LEA DX,TABLEA
```

### Operador MASK

Véase "Directiva RECORD" en la sección titulada "Directivas".

### Operador PTR

El operador PTR puede ser usado en variables de datos y en etiquetas de instrucción. Utiliza los especificadores de tipo BYTE, WORD, FWORD, DWORD, QWORD y TBYTE para especificar un tamaño en un operando ambiguo o para pasar por encima del tipo definido (DB, DW, DF, DD, DF o DT) para variables. También usa los especificadores de tipo NEAR, FAR y PROC para pasar por alto la distancia implicada de etiquetas. El formato general es

```
tipo PTR expresión
```

El tipo es el nuevo atributo, tal como BYTE. La expresión es una variable o constante. A continuación están unos ejemplos del operador PTR (tenga cuidado por FLDW, donde los bytes están en secuencia inversa de bytes):

```
FLDB DB   22H
     DB   35H
FLDW DW   2672H                  /Almacenado como 7226


     MOV  AH, BYTE PTR FLDW      ,-Mueve el primer byte (72)
     ADD  BL,BYTE PTR FLDW+1     ;Suma el segundo byte (26)
     MOV  BYTE PTR FLDW, 05      ,-Mueve 05 al primer byte
     MOV  AX,WORD PTR FLDB       ,-Mueve dos bytes (2335) al AX
     CALL FAR PTR [BX]           ,-Llama a un procedimiento lejano
```

Una característica **que** realiza una función semejante a PTR es la directiva LABEL, descrita posteriormente.

### Operador SEG

El operador SEG regresa dirección del segmento en el que una variable o etiqueta especificada es colocada. Los programas que combinan segmentos ensamblados por separados son los más adecuados para usar este operador. El formato general es

```
SEG variable o etiqueta
```

La siguiente instrucción MOV regresa la dirección del segmento en el que los nombres referenciados están definidos:

```
MOV  DX,SEG FLDW   /Dirección del segmento de datos
MOV  DX,SEG A20    /Dirección del segmento de código
```

### Operador para pasar por alto el segmento

Este operador, codificado como dos puntos (:), calcula la dirección de una etiqueta o de una variable relativo a un segmento en particular. El formato general es

```
segmento:expresión
```

El segmento nombrado puede ser cualquiera de los registros de segmento o un nombre de segmento o de grupo. La expresión puede ser una constante, una expresión o una expresión SEG. Los siguientes ejemplos pasan por alto el registro de segmento DS por omisión:

```
MOV  BH,ES:10H     ;Accesa desde ES más el desplazamiento 10H
MOV  CX,SS:[BX]    ,-Accesa desde SS más el desplazamiento en BX
```

Una instrucción puede tener un operador de pasar por alto el segmento aplicado a un solo operando.

### Operador SHL y SHR

Los operadores SHL y SHR hacen un corrimiento de una expresión durante un ensamblado. Los formatos generales son:

```
expresión  SHL  contador
expresión  SHR  contador
```

En el ejemplo siguiente, el operador SHR hace un corrimiento de la constante de bit tres bits hacia la derecha:

```
MOV BL,01011101B SHR 3    ;Carga 00001011B
```

Más probablemente, la expresión referenciaría a un nombre simbólico en lugar de a un valor constante.

### Operador SHORT

El objetivo del operador SHORT es modificar el atributo NEAR del destino de un JMP que está entre +127 y -128. El formato es

```
JMP SHORT etiqueta
```

El ensamblador reduce el código de máquina de un operando de dos bytes a uno. Esta característica es útil para saltos cercanos que van hacia adelante, ya que de otra manera el ensamblador en un inicio no sabe la distancia de la dirección del salto y puede suponer que es de dos bytes para un salto lejano.

### Operador SIZE

El operador SIZE regresa el producto de LENGTH veces TYPE y es útil sólo si la variable referenciada contiene la entrada DUP. El formato general es

```
SIZE variable
```

Para un ejemplo, véase "Operador TYPE"

### Operador THIS

El operador THIS crea un operando con segmento y desplazamiento que son iguales a aquéllos de la posición actual del contador. El formato general es

```
THIS tipo
```

El especificador de tipo puede ser BYTE, WORD, DWORD, FWORD, QWORD o TBYTE para variables y NEAR, FAR o PROC para etiquetas. Por lo común usted utiliza THIS con la directiva EQU, o signo igual (=). El ejemplo siguiente define FLDA:

```
FLDA  EQU   THIS  BYTE
```

El segmento es el mismo como si usted usó la directiva LABEL

```
FLDE   LABEL  BYTE
```

### Operador TYPE

El operador TYPE regresa el número de bytes, de acuerdo a la definición de la variable referenciada. Sin embargo, la operación siempre regresa 1 para una variable de cadena y 0 para una constante

| DEFINICIÓN | NÚMERO DE BYTES PARA UNA VARIABLE NUMÉRICA |
|---|---|
| DB | 1 |
| DW | 2 |
| DD | 4 |
| DF | 6 |
| DQ | 8 |
| DT | 10 |
| STRUC | Número de bytes definido para STRUC |
| NEAR | etiqueta FFFFH |
| FAR | etiqueta FFFEH |

El formato general de TYPE es

```
TYPE variable o etiqueta
```

Los ejemplos siguientes ilustran los operadores TYPE, LENGTH y SIZE:

```
FLDB    DB    ?    /Define un byte
TABLEA  DW    20 DUP(?)    /Define 20 palabras

        MOV   AX,TYPE FLDB        /AX = 0001H
        MOV   AX,TYPE TABLEA      /AX = 0002H
        MOV   CX,LENGTH TABLEA    /CX = 000AH (10)
        MOV   DX,SIZE TABLEA      /DX = 0014H (20)
```

Como TABLEA está definida como DW, TYPE regresa 0002H, LENGTH regresa 000AH con base en la entrada DUP y SIZE regresa type veces la longitud, o 14H (20).

### Operador WIDTH

Véase "Directiva RECORD" en la sección siguiente.

## DIRECTIVAS

Esta sección describe la mayoría de las directivas de lenguaje ensamblador. El capítulo 4 trata en detalle las directivas para la definición de datos (DB, DW, etcétera) y el capítulo 22 cubre las directivas para las macro instrucciones, de modo que no serán repetidas aquí. Las directivas están divididas en varias categorías:

- Etiquetas de código: ALIGN, EVEN, LABEL y PROC.
- Ensamblado condicional: IF, ELSE y otras, estudiadas en el capítulo 21.
- Errores condicionales: .ERR, .ERR1 y otras.
- Asignación de datos: ALIGN, EQU, EVEN, LABEL y ORG. DB, DW, DD, DF, DQ y DT, estudiadas en el capítulo 4.
- Control de listado: .CREF, .LIST, .PAGE, SUBTTL (SUBTITLE), TITLE, XCREF y .XLIST, estudiadas en este capítulo. .LALL, .LFCOND, .SALL, .SFCOND, .TFCONDy .XALL, estudiadas en el capítulo 22.
- Macros: ENDM, EXITM, LOCAL, MACRO y PURGE, estudiadas en el capítulo 21.
- Varias: COMMENT, INCLUDE, INCLUDELIB, ÑAME, &OUT y .RADIX.
- Procesador: .8086, .286, .286P, .386, .386P, .8087, .287, .387, etcétera.
- Bloques repetidos: IRP, IRPC y REPT, estudiados en el capítulo 22.
- Alcance: COMM, EXTRN y PUBLIC.
- Segmento: .ALPHA, ASSUME, .DOSSEG, END, ENDS, GROUP, SEGMENT y .SEQ.
- Segmento simplificado: .CODE, .CONST, .DATA, .DATA?, DOSSEG, .EXIT, .FARDATA, .FARDATA?, .MODEL y .STACK.
- Estructura/registro: ENDS, RECORD, STRUCT, TYPEDEF, UNION.

Ya que el conocimiento de estas categorías no es necesario, estudiaremos las directivas (distintas de las relacionadas con macros) en orden alfabético.

### Directiva ALIGN

MASM 5.0 introdujo la directiva ALIGN para forzar al ensamblador alinee el siguiente elemento de datos o instrucción de acuerdo a un valor dado. El formato general es

```
ALIGN numero
```

El número debe ser una potencia dé 2, tal como 2, 4, 8 o 16. Para el enunciado ALIGN 4, el ensamblador avanza su contador de posición a la dirección siguiente que sea divisible entre 4. Si el contador de posición ya está en la posición necesaria, no avanza. El ensamblador llena los bytes

no usados con ceros para datos e instrucciones NOP. Note que ALIGN 2 tiene el mismo efecto que EVEN.

La alineación no se aprovecha en el procesador 8088, el cual accesa sólo un byte a la vez, pero puede aumentar la velocidad en procesadores más avanzados.

### Directiva .ALPHA

La directiva .ALPHA, colocada en o cerca del inicio de un programa, le indica al ensamblador que acomode los segmentos en orden alfabético. Pasa por encima de la opción de ensamblador /S. (Véase también la directiva .SEQ.)

### Directiva ASSUME

ASSUME le indica al ensamblador que asocie nombres de segmento con los registros de segmento CS, DS, ES y SS. El formato general **es**

```
ASSUME  reg-seg:nom-reg[...]
```

Entradas válidas de registro de segmento son CS, DS, ES y SS, más FS y GS en los procesadores 80386 y posteriores. Nombres válidos de segmentos son aquellos de registros de segmentos, NOTHING, GROUPs y una expresión SEG. Un enunciado ASSUME puede asignar hasta cuatro registros de segmento, en cualquier orden. Las directivas simplificadas de segmento de manera automática generan un ASSUME.

En el enunciado ASSUME siguiente, CODESG, DATASG y STACK son los nombres que el programa ha usado para definir a los segmentos:

```
ASSUME  CS:CODESG,DS:DATASG,SS:STACK,ES:DATASG
```

La omisión de una referencia de segmento es lo mismo que codificar NOTHING. Usar la palabra clave NOTHING también cancela cualquier ASSUME anterior para un registro de segmento especificado:

```
ASSUME  ES:NOTHING
```

Suponga que no asigna el registro ES ni usa NOTHING para cancelarlo. Entonces, para referenciar un elemento de dato en el segmento de datos, un operando de instrucción puede usar el operador para pasar por alto el segmento (:) para referenciar el registro ES, que debe contener una dirección válida:

```
MOV AX,ES:[BX]       /Utilice dirección inde>fada
MOV AX,ES:FLDW       /Mueva el contenido de FLDW
```

### Directiva .CODE

Esta directiva simplificada de segmento define el segmento de código. Su formato general es

```
.CODE  [nombre]
```

Todo código ejecutable debe ser colocado en este segmento. Para modelos TINY, SMALL y COMPACT, por omisión el nombre del segmento es TEXT. Los modelos de memoria MÉDIUM y LARGE permiten múltiples segmentos de código, los cuales se distinguen por medio del nombre del operando. (Véase también la directiva .MODEL.)

### Directiva COMM

Al definir una variable como COMM le da los atributos PUBLIC y EXTRN. De esta manera, no tendría que definir la variable como PUBLIC en un módulo y EXTRN en otro. El formato general es:

```
COMM [NEAR/FAR] etiqueta:tamaño[:contador]
```

- COMM es codificada dentro de un segmento de datos.
- Los atributos NEAR o FAR pueden ser codificados o permitir los valores por omisión para uno u otro, dependiendo del modelo de memoria.
- Etiqueta es el nombre de la variable. Note que la variable no puede tener un valor inicial.
- Tamaño puede ser cualquiera de los especificadores de tipo BYTE, WORD, DWORD, QWORD y TBYTE, o un especificador del número de bytes.
- Contador indica el número de elementos para la variable. Por omisión es 1.

El ejemplo siguiente define FLDCOM con **el** atributo COMM:

```
COMM NEAR FLDCOM:WORD
```

### Directiva COMMENT

Esta directiva es útil para múltiples líneas de comentarios. El formato general es

```
COMMENT delimitador [comentarios]
          [comentarios]
delimitador [comentarios]
```

El delimitador es el primer carácter diferente de espacio en blanco que sigue a COMMENT, tal como % o + . El comentario termina en la línea en la que aparece un segundo delimitador. En el ejemplo siguiente se utiliza un signo más como un delimitador:

```
COMMENT + Esta rutina busca
          en la cadena de entrada
          por caracteres
        + no válidos.
```

### Directiva .CONST

Esta directiva simplificada de segmento define un segmento de datos (o constante) con la clase 'const'. (Véase también la directiva .MODEL.)

### Directiva .CREF

Esta directiva (que se tiene por omisión) le indica a¡ ensamblador que genere una tabla de referencias cruzadas. Sería usada a continuación una directiva .XCREF que causa la supresión de la tabla.

### Directivas .DATA y .DATA?

Estas directivas simplificadas de segmento definen segmentos de datos. .DATA define un segmento para inicializar datos cercanos; .DATA? define un segmento para datos cercanos no

inicializados, por lo común usada cuando se enlaza con lenguajes de alto nivel. Para un programa en ensamblador autónomo, también puede definir datos cercanos no inicializados en un segmento .DATA. (Véase, además, las directivas .FARDATA y .MODEL.)

### Directiva DOSSEG

Existen varias maneras de controlar la secuencia en la que el ensamblador acomoda los segmentos. (Algunas versiones lo hacen alfabéticamente.) Puede codificar las directivas .SEQ o .ALPHA al inicio de un programa, o puede ingresar las opciones del ensamblador /S o /A en el momento del ensamblado. La directiva DOSSEG (.DOSSEG desde MASM 6.0) le indica al ensamblador ignorar todas las demás peticiones y adoptar la secuencia de segmentos DOS —básicamente, código, datos y pila. Codifique esta directiva en o cerca del inicio del programa, principalmente para facilitar el uso **de** CODEVIEW para programas autónomos.

### Directiva END

La directiva END es colocada al final de un programa fuente. El formato general es

```
END  [dirección-inicial]
```

La dirección-inicial opcional indica la localidad en el segmento de código (por lo regular la primer instrucción) en donde la ejecución empieza. El cargador del sistema utiliza esta dirección para inicializar el registro CS. Si su programa consiste en sólo un módulo, defina una dirección inicial. Si consiste en varios módulos, sólo uno (por lo regular el primero) tiene la dirección-inicial.

### Directiva ENDP

Esta directiva indica el final de un procedimiento, definido por PROC. El formato general es

```
etiqueta  ENDP
```

La etiqueta es la misma **como** la **que** define el procedimiento.

### Directiva ENDS

Esta directiva indica el final de un segmento (definido como SEGMENT) o de una estructura. Su formato general es

```
etiqueta  ENDS
```

La etiqueta es la misma como la que define el segmento o estructura.

### Directiva EQU

La directiva EQU es usada para redefinir un nombre de dato o una variable con otro nombre de dato, variable o valor inmediato. La directiva debe ser definida en un programa antes de ser referenciada. Los formatos para datos numéricos y de cadena de caracteres difieren:

```
Equivalencia numérica:      nombre EQU expresión

Equivalencia de cadena:     nombre EQU <cadena>
```

El ensamblador reemplaza cada ocurrencia del nombre con el operando. Ya que EQU es utilizada para reemplazo simple, no ocupa espacio de almacenamiento adicional en el programa objeto generado.

Ejemplos del uso de EQU con datos numéricos son:

```
COUNTER DW    0
SUMA    EQU   CONTADOR      /Otro nombre para CONTADOR
DIEZ    EQU   10            /Valor numérico

        INC   SUMA          ;Incrementa CONTADOR
        ADD   SUM,DIEZ      ;Suma 10 a CONTADOR
```

Ejemplos del uso de EQU con datos de cadena de caracteres son:

```
PRODMSG  EQU   <'Enter product number'>
BYPTR    EQU   <BYTE PTR>

MSG           PRODMSG          ,-Reemplaza con cadena de caracteres

        MOV   SAVE,BYPTR [BX]     /Reemplaza con cadena de caracteres
```

Los paréntesis angulares hacen más fácil la indicación de un operando de cadena de caracteres.

### Directiva .ERR

Estas directivas de error condicional pueden ser usadas para ayudar a probar por errores durante un ensamblado:

| DIRECTIVA | ERROR FORZADO |
|---|---|
| .ERR | Cuando es encontrado |
| .ERR1 | Durante la pasada uno de un ensamblador |
| .ERR2 | Durante la pasada dos de un ensamblador |
| .ERRE | Por una expresión verdadera (0) |
| .ERRNZ | Por una expresión falsa (no 0) |
| .ERRDEF | Por símbolo definido |
| .ERRNDEF | Por símbolo no definido |
| .ERRB | Por cadena en blanco |
| .ERRNB | Por cadena no en blanco |
| .ERRIDN[I] | Por cadenas idénticas |
| .ERRDIF[I] | Por cadenas diferentes |

Podría usar las directivas precedentes en macros y en enunciados condicionales en ensamblador. En los enunciados condicionales de siguientes ensambladores, el ensamblador muestra un mensaje si la condición no es verdadera:

```
IF      condición

ELSE    .ERR
        #OUT  [mensaje]
ENDIF
```

Desde MASM 6.0 ya no es necesario referirse a la pasada 1 (.ERR1) o a la pasada 2 (.ERR2) de un ensamblado.

### Directiva EVEN

EVEN le indica al ensamblador avanzar su contador de localidad si es necesario, de modo que el siguiente dato o etiqueta definido esté alineado con una frontera par de almacenamiento. Esta característica hace el procesamiento más eficiente en procesadores que accesan 16 o 32 bits a un tiempo. (Véase también la directiva ALIGN.)

En el ejemplo siguiente, BYTELOCN es un campo de un byte en una frontera par. El contador de localidad del ensamblador inicia en 0017. EVEN causa que el ensamblador avance el contador de localidad un byte a 0018:

```
0OÍS    BYTELOCN  DB ?
0017              EVEN          (avanza el contador de localidad)
[0017             NOP]
0018    WORDLOCN  DW ?
```

### Directiva EXTRN

La directiva EXTRN (o EXTERN desde MASM 6.0) informa al ensamblador y al enlazador acerca de las variables de datos y de las etiquetas que el actual ensamblado hace referencia, pero que otro módulo (enlazado al actual) define. El formato general es

```
EXTRN nombre:tipo [, ... ]
```

La entrada nombre es un elemento definido en otro ensamblado y declarado en él como PUBLIC. El especificador tipo puede referirse a cualquiera de lo siguiente:

- Elementos de datos: ABS (una constante), BYTE, WORD, DWORD, FWORD, QWORD, TBYTE. Codifique el EXTRN en el segmento en el que los elementos aparecen.
- Distancia: NEAR o FAR. Codifique NEAR en el segmento en el que el elemento ocurre y codifique FAR en cualquier otro lugar.

En el ejemplo siguiente, el programa que llama define CONVAL como PUBLIC y como un DW. El subprograma llamado identifica CONVAL (en otro segmento) como EXTRN y FAR. El código es como sigue:

Programa que llama:

```
DSEGl   SEGMENT

        PUBLIC  CONVAL


CONVAL  DW ?


DSEGl   ENDS
```

Subprograma llamado:

```
        EXTRN  CONVAL : FAR
DSEG2   SEGMENT

        MOV    AX, CONVAL

DSEG2   ENDS
```

Véase el capítulo 23 para ejemplos de EXTRN.

### Directivas .FARDATA y .FARDATA?

Estas directivas simplificadas de segmento definen segmentos de datos. .FARDATA define un segmento para datos lejanos inicializados, y .FARDATA? define un segmento para datos lejanos no incializados. Para un programa ensamblado autónomo, también puede definir datos lejanos no inicializados en un segmento .FARDATA. (Véase también las directivas .DATA y .MODEL.)

### Directiva GROUP

Un programa puede contener varios segmentos del mismo tipo (código, datos y pila). El objetivo de la directiva GROUP es juntarlos bajo un mismo nombre, de modo que residan dentro de un segmento, por lo regular un segmento de datos. El formato general es

```
nombre GROUP nom-seg [, nom-seg], ...
```

El siguiente GROUP combina SEGl y SEG2 en el mismo módulo ensamblado:

```
GROUPX  GROUP    SEGl, SEG2
SEGl    SEGMENT  PARA 'Data'
        ASSUME   DS:GROUPX


SEGl    ENDS
SEG2    SEGMENT  PARA 'Data'
        ASSUME   DS:GROUPX


SEG2    ENDS
```

El efecto de usar GROUP es semejante al dar a los segmentos el mismo nombre y el atributo PUBLIC.

### Directiva INCLUDE

Puede tener secciones de código ensamblado o macroinstrucciones que varios programas utilicen. Si es así, puede almacenar éstas en archivos de disco separados disponibles para usarse por

cualquier programa. Considere una rutina que convierta código ASCII a binario que está almacenada en la unidad D en el archivo con nombre CONVERT.LIB. Para accesar el archivo, inserte un enunciado INCLUDE tal como

```
INCLUDE D:CONVERT.LIB
```

en la posición en el código fuente en donde normalmente codificaría la rutina de conversión a ASCII. El ensamblador entonces localiza el archivo en disco e incluye el enunciado en su propio programa. (Si el ensamblador no puede encontrar el archivo, emite un mensaje de error e ignora el INCLUDE.)

Para cada línea incluida, el ensamblador imprime una C en la columna 30 del archivo .LST y empieza el código fuente en la columna 33.

El capítulo 22 da un ejemplo práctico de INCLUDE y explica cómo usar la directiva sólo para la pasada uno de un ensamblado.

### Directiva LABEL

La directiva LABEL le permite redefinir el atributo de una variable de datos o de una etiqueta de instrucción. El formato general es

```
nombre LABEL especificador-de-típo
```

Para etiquetas, puede usar LABEL para redefinir código ejecutable como NEAR, FAR o PROC, tal como para un punto secundario de entrada en un procedimiento. Para variables puede usar los especificadores de tipo BYTE, WORD, DWORD, FWORD, QWORD o TBYTE, o un nombre de estructura, para redefinir elementos de datos y los nombres de las estructuras, respectivamente. Por ejemplo, LABEL le permite definir un campo a la vez como DB y como DW. Lo siguiente ilustra el uso de los tipos BYTE y WORD:

```
REDEFB  LABEL  BYTE
FIELDW  DW     2532H
REDEFW  LABEL  WORD
FIELDB  DB     25H
        DB     32H

        MOV    AL,REDEFB   ,-Mueve el primer byte
        MOV    BX,REDEFW   ,-Mueve 2 bytes
```

La primer instrucción MOV sólo mueve el primer byte de FIELDW. El segundo MOV mueve los dos bytes empezando en FIELDB. El operador PTR realiza una función similar.

### Directiva .LIST

La directiva .LIST (que es por omisión) causa que el ensamblador liste el programa fuente. Puede usar la directiva .XLIST en cualquier parte en un programa fuente ensamblador para discontinuar el listado. Una situación común es en donde los enunciados son comunes a otros programas y no necesita listarlo otra vez. .LIST reasume el listado. Codifique ambas directivas sin operandos.

### Directiva .MODEL

Esta directiva simplificada de segmento crea segmentos por omisión y los enunciados ASSUME y GROUP necesarios. **Su** formato general es

```
.MODEL  modelo-memoria
```

Los modelos de memoria son

| | |
|---|---|
| TINY | Desde MASM 6.0, usado para programas .COM. |
| SMALL | Todos los datos en un segmento y todo el código en un segmento. |
| MÉDIUM | Todos los datos **en** un segmento, pero el código en más de un segmento. |
| COMPACT | Datos en más de un segmento, pero código en un segmento. |
| LARGE | Datos y código, ambos en más de un segmento, pero los arreglos no pueden exceder 64K. |
| HUGE | Datos y código, ambos en más de un segmento, y los arreglos pueden exceder 64K. |

La directiva .STACK define la pila, .CODE define el segmento de código y cualquiera de .DATA, .DATA?, .FARDATA y .FARDATA? pueden definir segmentos de datos. A continuación está un ejemplo:

```
.MODEL   SMAL
.STACK   120
.DATA
         [elemento de dato]
.CODE
         [instrucciones]
END
```

### Directiva ORG

El ensamblador utiliza un *contador de localidades* para llevar la cuenta de su posición relativa en **un** segmento de datos o de código. Considere un segmento de datos con las definiciones siguientes:

| DESPLAZAMIENTO | NOMBRE | OPERACIÓN | OPERANDO | CONTADOR DE LOCALIDADES |
|---|---|---|---|---|
| 00 | FLDA | DW | 2542H | 02 |
| 02 | FLDB | DB | 36H | 03 |
| 03 | FLDC | DW | 212EH | 05 |
| 05 | FLDD | DD | 00000705H | 09 |

En un inicio, el contador de localidades es puesto en 00. Como FLDA es de dos bytes, el contador de localidades se incrementa en 02 para la localidad del siguiente elemento. Como FLDB es un

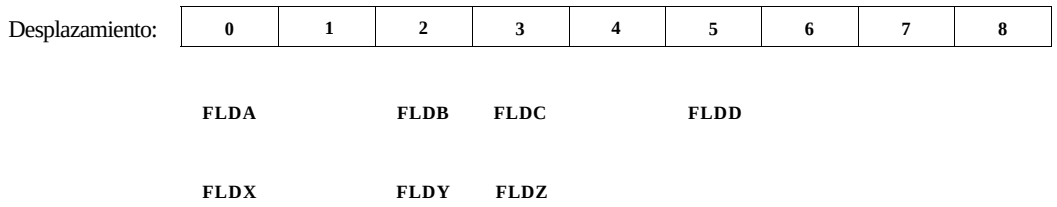
byte, el contador de localidades se incrementa a 03, y así sucesivamente. Puede usar la directiva ORG para cambiar el contenido del contador de localidades y, de acuerdo con eso, la localidad de los siguientes elementos definidos. El formato general es

```
ORG expresión
```

La expresión debe formar un número absoluto de dos bytes y no debe ser un nombre simbólico. Suponga que los siguientes elementos están definidos después de FLDD en la definición anterior:

```
DESPLAZAMIENTO  NOMBRE   OPERACIÓN  OPERANDO     CONTADOR DE LOCALIDADES
                         ORG        0                 00
00              FLDX     DB         ?                 01
01              FLDY     DW         ?                 02
03              FLDZ     DB         ?                 04
                         ORG        $ + 5             09
```

El primer ORG restablece el contador de localidades a 00. Las variables que siguen —FLDX, FLDY y FLDZ— redefinen estas localidades de memoria como FLDA, FLDB y FLDC, respectivamente:




Un operando que contiene un signo de pesos ($), como en el segundo ORG, se refiere al valor actual en el contador de localidades. Por tanto, el operando $+5 pone el contador de localidades a 04 + 5, o 09, que es el mismo como después de la definición de FLDD.

Una referencia a FLDC es a un campo de un byte en el desplazamiento 03, y una referencia a FLDZ es a un campo de un byte en el desplazamiento 03:

```
MOV  AX,FLDC      ;Una palabra
MOV  AL,FLDZ      ;Un byte
```

Puede usar ORG para redefinir localidades de memoria en la manera precedente. Pero asegúrese que usted restablecerá el contador de localidades al valor correcto y que tiene en cuenta para todas las localidades de memoria rede finidas. También, las variables redefinidas no deben contener constandes definidas —éstas se traslaparían encima de las originales. ORG no puede aparecer dentro de una definición STRUC.

### Directiva %OUT

Esta directiva le indica al ensamblador que dirija un mensaje al dispositivo de salida estándar (por lo común la pantalla). (Desde el MASM 6.0, el nombre es ECHO.) El formato general es

```
#OUT mensaje
```

La sección "Directivas .ERR" dan un ejemplo.

### Directiva PAGE

La directiva PAGE al inicio de un programa fuente especifica el número máximo de líneas a lista en una página y el número máximo de caracteres en una línea. Su formato general es

```
PAGE [[longitud] ,ancho]
```

El ejemplo siguiente establece 60 líneas por página y 132 caracteres por línea:

```
PAGE 60,132
```

El número de líneas por página puede ser en el rango de 10 a 255, y el número de caracteres por línea desde 60 hasta 132. Por omisión de un enunciado PAGE hace que el ensamblador suponga PAGE 50,80. Para forzar que una página salte a una línea específica, tal como al final de un segmento, codifique PAGE sin operando.

### Directiva PROC

Un procedimietno es un bloque de código que inicia con la directiva PROC y termina con ENDP. Un uso común es para una subrutina dentro de un segmento de código. Aunque técnicamente, usted puede ingresar a un procedimiento en una línea o por una instrucción JMP, la práctica común es utilizar CALL para entrar y RET para salir. El operando CALL puede ser un especificador de tipo NEAR o FAR y RET supone el mismo tipo.

Un procedimiento que está en el mismo segmento que el programa que llama es un procedimiento NEAR (cercano) y es accesado por un desplazamiento:

```
nombre-proc PROC [NEAR]
```

Un operando omitido por omisión es NEAR. Si un procedimietno llamado es externo al segmento que hace la llamada, debe ser declarado como PUBLIC y debe usar CALL para introducirlo.

Para un programa .EXE, el PROC principal que es el punto de entrada para la ejecución debe ser FAR. También un procedimietno llamado bajo un valor diferente de ASSUME CS debe tener el atributo FAR:

```
                PUBLIC  nombre-proc

nombre-proc     PROC    FAR
```

Una etiqueta lejana puede estar en otro segmento, el cual CALL accesa por una dirección y desplazamiento de segmento.

### Directiva de procesador

Estas directivas definen los procesadores que el ensamblador va a reconocer. La colocación normal de las directivas de procesador es al inicio de un programa fuente, aunque podría codificarlas dentro de un programa en un punto en donde quiera habilitar o deshabilitar un procesador. Una referencia al 8086 también supone **el** 8088, y el .486 fue introducido por MASM 6.0.

- .8086 habilita el 8086 y el coprocesador 8087 **(el** modo por omisión).
- .186, .286, .386 y .486 permiten la configuración de instrucciones e incluyen los procesadores nombrados y su coprocesador asociado. Esto es, la directiva permite instrucciones de procesadores anteriores. (Por ejemplo, .386 habilita .387, .286, .186 y .8086.)

- .186P, .286P, .386P y .486P habilita todos los conjuntos de instrucciones citados más las instrucciones privilegiadas del procesador.

### Directiva PUBLIC

El objetivo de la directiva PUBLIC es para informar al ensamblador y al enlazador que los símbolos identificados en un ensamblado serán referenciados por otros módulos enlazados con el actual. El formato general es

```
PUBLIC símbolo [, . . . ]
```

El símbolo puede ser una etiqueta, un número (hasta de dos bytes) o una variable. Para ejemplos, véase la sección "Directiva EXTRN" en el capítulo 23.

### Directiva RECORD

La directiva RECORD le permite definir patrones de bits. Un objetivo es definir indicadores de conmutación como un bit o como de multibits. El formato general es

```
nom-reg RECORD nom-campo:ancho[=exp]  [,  ...]
```

El nombre del registro y los nombres de campo pueden ser cualesquiera identificadores únicos. A continuación de cada nombre de campo está dos puntos (:) y un ancho —el número de bits. El rango de ancho es de 1 a 16 bits:

| NÚMERO DE BITS DEFINIDOS | TAMAÑO POR OMISIÓN |
|---|---|
| 1-8 | 8 |
| 9-16 | 16 |

Cualquier longitud hasta de 8 se convierte en 8 bits, y longitudes de 9 a 16 se convierten en 16 bits, con el contenido ajustado si es necesario. El ejemplo siguiente define RECORD:

```
BITREC  RECORD  BIT1:3,BIT2:7,BIT3:S
```

BIT1 define los primeros 3 bits de BITREC, BIT2 define los siguientes 7 y BIT3 define los últimos 6. El total es de 16 bits, o una palabra. Puede inicializar valores en RECORD como sigue:

```
BITREC2  RECORD  BIT1:3=10IB,BIT2:7=0110110B,BIT3:G=011010B
```

Suponga que una definición de RECORD está al inicio del segmento de datos. Dentro del segmento de datos, debe estar otro enunciado que asigne espacio para el registro. Defina un nombre válido único, el nombre del registro y un operando que consiste en paréntesis angulares (los símbolos menor y mayor que):

```
DEFBITS  BITREC  o
```

La asignación para DEFBITS genera el código objeto AD9AH (almacenado como 9AAD) en el segmento de datos. Los paréntesis angulares también pueden contener entradas que redefinan a BITREC.

El programa de la figura 27-1 define BITREC como RECORD, pero sin valores iniciales en el registro de campos. En este caso, un enunciado de asignación en el segmento de datos inicializa cada campo como se muestra dentro de los paréntesis angulares.

```
                        TITLE     P27RECOR (COM)   Prueba de la directiva RECORD
0000                    CODESG    SEGMENT PARA 'Code'
                                  ASSUME  CS:CODESG,DS:CODESG,SS:CODESG
0100                              ORG     100H
0100 EB 02              BEGIN:    JMP     SHORT  MAIN

                        BITREC    RECORD BIT1:3,BIT2:7,BIT3:6     ,-Define registro
0102 AD9A               DEFBITS   BITREC <101B, 0110110B, 011010B>  ,-Inic. registro

0104                    MAIN      PROC    NEAR
0104                    A10:                                  Ancho:
0104 B7 10                        MOV     BH,WIDTH  BITREC      de registro (16)
0106 BO 07                        MOV     AL, WIDTH BIT2        de campo (07)
0108                    B10:                                  Contador de corrimiento
0108 Bl OD                        MOV     CL,BIT1               OD
010A Bl 06                        MOV     CL,BIT2               06
010C Bl 00                        MOV     CL,BIT3               0 0 hex
010E                    CIO:                                  Máscara:
010E B8 E000                      MOV     AX,MASK  BIT1         E000
0111 BB 1FC0                      MOV     BX,MASK  BIT2         1FC0
0114 B9 003F                      MOV     CX,MASK  BIT3         003F hex
0117                    DIO:                                  BIT2 aislado:
0117 Al 0102 R                    MOV     AX,DEFBITS            obtiene registro
011A 25 1FC0                      AND     AX,MASK  BIT2         limpia BIT1 y 3
011D Bl 06                        MOV     CL,BIT2  '            obtiene despl. 06
011F D3 E8                        SHR     AX,CL                 corri. a la derecha
0121                    E10:                                  BIT1 aislado:
0121 Al 0102 R                    MOV     AX,DEFBITS            Obtiene registro
0124 Bl OD                        MOV     CL,BIT1               obtiene corri. 13
0126 D3 E8                        SHR     AX, CL                despl. a la derecha
0128 B8 4C00                      MOV     AX,4C00H              Sale al DOS
012B CD 21                        INT     21H
012D                    MAIN      ENDP
35 012D                 CODESG    ENDS
36                                END     BEGIN

Estructuras y registros:
          N a ra e                Width    # fields
                                  Shift     Width   Mask    Initial
BITREC                          . ,  0010   0003
  BIT1                          . .  000D   0003    E000     0000
  BIT2                          . ,  0006   0007    1FC0     0000
  BIT3                          . .  0000   0006    003F     0000
Segmenta and Groups:
          Ñ a m e                 Length   Align   Combine  Class
CODESG                             012D    PARA    NONE     CODE'
Symbols:
          Ñ a m e                  Type     Valué    Attr
A10                             . . L NEAR  0104    CODESG
B10                               L NEAR    0108    CODESG
BEGIN                             L NEAR    0100    CODESG

BIT2                               0006
BIT3                            , ,  0000
CÍO                             . , L NEAR  010E    CODESG
DIO                             . . L NEAR  0117    CODESG
DEFBITS                         , , L WORD  0102    CODESG
E10                               L NEAR    0121    CODESG
MAIN                            , , N PROC  0104    CODESG   Lengtl
```

Figura 27-1 Uso de la directiva RECORD

Los operadores específicos de registros son WIDTH, cuenta de desplazamiento y MASK. El uso de estos operadores le permite cambiar una definición de RECORD sin tener que cambiar las instrucciones que hacen referencia a él.

Operador WIDTH. El operador WIDTH regresa un ancho como el número de bits en un RECORD o en un campo RECORD. Por ejemplo, en la figura 27-1, después de A10 están dos ejemplos de WIDTH. El primer MOV regresa el ancho de todo el RECORD BITREC (16 bits); el segundo MOV regresa el ancho del registro de campo BIT2 (7 bits). En ambos casos, el ensamblador ha generado un operando inmediato para WIDTH.

Contador de desplazamiento. Una referencia directa a un campo RECORD, tal como

```
MOV CL,BIT2
```

no se refiere al contenido de BIT2. (En realidad, sería muy difícil.) En lugar de eso, el ensamblador genera un operando inmediato que contiene un *contador de corrimiento* para ayudarle a aislar el campo. El valor inmediato representa el número de bits que tendría que desplazar BIT2 para ajustado a la derecha. En la figura 2-1, los tres ejemplos que siguen a B10 regresan el contador de corrimiento para BITl, BIT2 y BIT3.

Operador MASK. El operador MASK regresa una máscara (filtro) de bits en uno que representa el campo especificado y, de hecho, define las posiciones de bits que el campo ocupa. Por ejemplo, el MASK para cada uno de los campos definidos en BITREC es

| CAMPO | BINARIO | HEXADECIMAL |
|---|---|---|
| BITl | 1110000000000000 | EOOO |
| BIT2 | 0001111111000000 | 1FC0 |
| BIT3 | 0000000000111111 | 003F |

En la figura 27-1, las tres instrucciones que siguen a CÍO regresan los valores de MASK para BITl, BIT2 y BIT3. Las instrucciones que siguen a DIO y E10 aislan BIT2 y BITl, respectivamente, de BITREC. DIO obtiene el registro en el registro AX y utiliza un MASK de BIT2 para realizar un AND:

```
Registro:        101 0110110  011010
AND MASK BIT2:   000 1111111  000000
Resultado:       000 0110110  000000
```

El efecto es limpiar todos los bits excepto aquellos de BIT2. Las dos instrucciones siguientes hacen que el AX recorra seis bits de modo que BIT2 esté alineado a la derecha:

```
0000000000110110  (0036H)
```

El ejemplo siguiente a E10 obtiene el registro en el AX y ya que BITl es el campo de más a la izquierda, la rutina sólo utiliza el factor de corrimiento para recorrer a la izquierda 13 bits:

```
0000000000000101  (0005H)
```

### Directiva SEGMENT

Un módulo ensamblado consiste en uno o más segmentos, parte de un segmento o aun partes de varios segmentos. El formato general es

```
nom-seg   SEGMENT [alinear]  [combinar]  Pelase']

nom-seg   ENDS
```

Todos los operandos son opcionales. Las siguientes subsecciones describen las entradas para alinear, combinar y clase.

**Alinear.** El operando alinear indica la frontera inicial para un segmento:

| | |
|---|---|
| BYTE | Siguiente dirección |
| WORD | Siguiente dirección par (divisible entre 2) |
| DWORD | Siguiente dirección de palabra doble (divisible entre 4) |
| PARA | Siguiente párrafo (divisible entre 10H) |
| PAGE | Siguiente dirección de página (divisible entre 100H) |

PARA es usado comúnmente para todos los tipos de segmentos. BYTE y WORD pueden ser usados para segmentos que serán combinados dentro de otro segmento, por lo común un segmento de datos. DWORD es normalmente usada con procesadores 80386 y posteriores.

**Combinar.** Los operandos combinar NONE, PUBLIC, STACK y COMMON indican la manera en que el enlazador maneja un segmento:

- NONE (por omisión): El segmento será separado lógicamente de otros segmentos, aunque pueden estar adyacentes físicamente. El segmento se supone tiene su propia dirección base.
- PUBLIC: LINK carga segmentos PUBLIC del mismo nombre y clase adyacentes uno con otro. Una dirección base se supone para tales segmentos PUBLIC.
- STACK: LINK trata STACK lo mismo que PUBLIC. Debe haber definido al menos un STACK en un programa .EXE enlazado. Si existe más de una pila, el SP es puesto al inicio de la primera pila.
- COMMON: Si segmentos COMMON tienen el mismo nombre y clase, el enlazador les da la misma dirección base. Durante la ejecución, el segundo segmento se traslapa con el primero. El segmetno más grande determina el tamaño del área común.
- AT dirección-párrafo: El párrafo debe estar definido previamente. La entrada facilita la definición de etiquetas y variables en desplazamientos fijos dentro de áreas fijas de memoria, tal como la tabla de interrupciones en memoria baja o el área de datos del BIOS en 40[0]H. Por ejemplo, el código en ROM define la posición del búfer del despliegue de video como

```
VIDEO RAM SEGMENT AT 0B800H
```

El ensamblador crea un segmento mudo (ficticio) que proporciona, de hecho, una imagen de las localidades de memoria.

'clase'. La entrada clase puede ayudar al enlazador a asociar segmentos con diferentes nombres, identificar segmentos y controlar su orden. Clase puede contener cuaquier nombre válido, escrito entre apóstrofos. El enlazador utiliza el nombre de los segmentos relacionados que tienen el mismo nombre y clase. Ejemplos típicos son 'Data' y 'Code'. Si define una clase como 'Code', el enlazador espera que el segmento contenga código de instrucción. También, el depurador CODEVIEW espera esa clase para el segmento de código.

El enlazador combina los dos segmentos siguientes con el mismo nombre (CSEG) y clase ('Code') en un segmento físico bajo el mismo registro de segmento:

```
Ensamblador  CSEG   SEGMENT PARA PUBLIC 'Code'
módulo 1             ASSUME  CS:CSEG

             CSEG   ENDS

Ensamblador  CSEG   SEGMENT PARA PUBLIC 'Code'
módulo 2             ASSUME  CS:CSEG

             CSEG   ENDS
```

Ya que puede querer controlar el orden de los segmentos dentro de un programa, es útil entender cómo el enlazador maneja el proceso. El orden original de los nombres de los segmentos proporciona la secuencia básica, que puede pasar por alto por medio del atributo PUBLIC y nombres de clase. El ejemplo siguiente enlaza dos módulos objeto (ambos módulos contienen un segmento llamado DSEGl con el atributo PUBLIC e idénticos nombres de clase):

Antes de enlazar los módulos .OBJ:

```
módulo 1    SSEG   SEGMENT   PARA STACK
módulo 1    DSEGl  SEGMENT   PARA PUBLIC 'Data
módulo 1    DSEG2  SEGMENT   PARA
módulo 1    CSEG   SEGMENT   PARA 'Code'
módulo 2    DSEGl  SEGMENT   PARA PUBLIC 'Data
módulo 2    DSEG2  SEGMENT   PARA
módulo 2    CSEG   SEGMENT   PARA 'Code'
```

Después de enlazar los módulos ,OBJ en un módulo .EXE:

```
módulo 1       CSEG   SEGMENT   PARA 'Code'
módulo 2       CSEG   SEGMENT   PARA 'Code'
módulo 1 + 2   DSEG1  SEGMENT   PARA PUBLIC 'Data'
módulo 1       DSEG2  SEGMENT   PARA
módulo 2       DSEG2  SEGMENT   PARA
módulo 1       SSEG   SEGMENT   PARA STACK
```

Puede anidar segmentos, siempre que un segmento anidado esté contenido por completo dentro del otro. En el ejemplo siguiente, SEG2 está contenido por completo dentro de SEG1:

```
____SEGl   SEGMENT
                          SEG1 inicio
r- SEG2    SEGMENT        SEG2 área
L SEG2     ENDS           SEGl resume
____SEGl   ENDS
```

Las directivas .ALPHA, .SEQ y DOSSEG y las opciones de ensamblador /A y /S también; pueden controlar el orden de los segmentos. (Para combinar segmentos en grupos, véase la directiva GROUP.)

**Directiva .SEQ**

Esta directiva (que es por omisión), colocada cerca del inicio de un programa, le inidica al ensamblador que deje segmentos en su secuencia original. Pasa por alto la opción del ensamblador /A. (Véase también la directiva .ALPHA.)

**Directiva .STACK**

Esta directiva simplificada de segmetno define la pila. Su formato general es

```
.STACK [tamaño]
```

El tamaño por omisión de la pila es de 1,024 bytes, que puede pasar por alto. (Véase también la directiva .MODEL.)

**Directiva STRUC**

La directiva STRUC (STRUCT desde MASM 6.0) facilita la definiciñon de campos relacionados dentro de una estructura. Su formato general es

```
nom-estruct  STRUC

        [campos definidos]

nom-estruct  ENDS
```

Una estructura empieza con su nombre y la directiva STRUC y termina con el nombre y la directiva ENDS. El ensamblador almacena los campos definidos uno después del otro desde el inicio de la estructura. Entradas válidas son definiciones DB, DW, DD, DQ y DT con nombres de campo opcionales.

En la figura 27-2, STRUC define una lista de parámetros llamada PARLIST para usarla con la función OAH de la INT 21H, para ingresar un nombre vía el teclado. Un enunciado subsecuente asigna almacenamiento para la estructura, haciéndola direccionable dentro del programa:

```
PARAMS  PARLIST  o
```

Los paréntesis angulares (símbolo menor y mayor que) en el operando están vacíos en este ejemplo, pero puede usarlos para redefinir (pasar por alto) datos dentro de una estructura.

Las instrucciones pueden hacer referencia a una estructura de forma directa por su nombre. Para referenciar campos dentro de una estructura, las instrucciones deben cualificarlas usando el nombre asignado de la estructura (PARAMS, en el ejemplo), seguido por un punto que lo conecta con el nombre del campo, como por ejemplo,

```
MOV AL, PARAMS.ACTLEN
```

También puede usar el enunciado para asignar (PARAMS en la figura 27-2) para redefinir el contenido de campos dentro de una estructura.

### Directiva SUBTTL

La directiva SUBTTL (SUBTITLE desde MASM 6.0) hace que un subtítulo de hasta 60 caracteres se imprima en la línea 3 de cada página de un listado fuente en ensamblador. Puede codificar SUBTTL cualquier número de veces. El formato general es

```
SUBTTL  texto
```

### Directiva TEXTEQU

El formato general para esta directiva (introducida por MASM 6.0) es

```
TEXTEQU  [elemento-texto]
```

El operando elemento-texto puede ser una cadena literal, una constante precedida por el signo % o una cadena que una macrofunción ha regresado.

### Directiva TITLE

La directiva TITLE hace que un título de hasta 60 caracteres se imprima en la línea 2 de cada página de un listado fuente en ensamblador. Puede codificar TITLE una vez, al inicio. El formato general es

```
TITLE  texto
```

```
                        TITLE   P27DSTRU (COM)   Definición de una estructura
0000                    CODESG  SEGMENT PARA 'Code'
                                ASSUME CS:CODESG,DS:CODESG,SS:CODESG
0100                            ORG     100H
0100 EB 2C              BEGIN:  JMP     SHORT MAIN

                        PARLIST STRUC                   ;-Lista de parámetros
0000 19                 MAXLEN  DB      25
0001 00                 ACTLEN  DB      ?
0002 0019 [20]          NAMEIN  DB      25 DUP(' ')
001B                    PARLIST ENDS

0102 19                 PARAMS  PARLIST o               ;Asignación de almacenamient
0103 00
0104 0019 [20]
011D 57 68 61 74 20 69  PROMPT  DB      'What is the name?'
     73 20 74 68 65 20
     6E 61 6D 65 3F

012E                    MAIN    PROC    NEAR
012E  B4 40                     MOV     AH,40H          ;Petición de despliegue
0130  BB 0001                   MOV     BX,01
0133  B9 0011                   MOV     CX,17           ;Longitud de la petición
0136  8D 16 011D R              LEA     DX,PROMPT       ;-Dirección de la indicación
013A  CD 21                     INT     21H
013C  B4 0A                     MOV     AH,0AH          ;Acepta entrada
013E  8D 16 0102 R              LEA     DX,PARAMS       desde el teclado
0142  CD 21                     INT     21H
0144  A0 0103 R                 MOV     AL,PARAMS.ACTLEN
                                                        ;Longitud de la entrada
0147  B8 4C00                   MOV     AX,4C00H        ;Sale al DOS
014A  CD 21                     INT     21H
014C                    MAIN    ENDP
014C                    CODESG  ENDS
42                              END     BEGIN


Estructuras y registros:
                Name            Width   # fields
                                Shift   Width   Mask    Initial
PARLIST . . . . . .             001B    0003
  MAXLEN . . . . . .            0000
  ACTLEN . . . . . .            0001
  NAMEIN . . . . . .            0002
Segments and Groups:
                Name            Length  Align   Combine Class
CODESG . . . . . .              014C    PARA    NONE    'CODE'
Symbols:
                Name            Type    Value   Attr
BEGIN . . . . . .               L NEAR  0100    CODESG
MAIN . . . . . .                N PROC  012E    CODESG  Length = 001E
PARAMS . . . . . .              L 0102  CODESG
PROMPT . . . . . .              L BYTE  011D    CODESG
```

Figura 27-2 Uso de la estructura

### Directiva .XCREF

Esta directiva (llamada .NOCREF desde MASM 6.0) le indica al ensamblador que suprima la tabla de referencias cruzadas. El formato general es

```
.XCREF [nombre [,nombre] ...]
```

La omisión del operando provoca la supresión de todas las entradas en la tabla. También puede suprimir la referencia cruzada de elementos en particular. A continuación están unos ejemplos de .XCREF y de .CREF.

```
XCREF                   /Suprime la referencia cruzada

 .CREF                  /Restablece la referencia cruzada

 .XREF FIELDA,FIELDB    /Suprime la referencia cruzada de FIELDA y FIELDB.
```

### Directiva .XLIST

Puede utilizar la directiva .XLIST (llamada .NOLIST desde MASM 6.0) en cualquier lugar en un programa fuente para discontinuar la impresión de un programa ensamblado. Una situación común sería en donde los enunciados son comunes a otros programas y no necesita otro listado. La directiva .LIST (por omisión) reasume el listado. Codifique ambas directivas sin operandos.

## CAPÍTULO 28

# El conjunto de instrucciones de la PC

OBJETIVO

Explicar el código de máquina y proporcionar una descripción del conjunto de instrucciones de la PC.

### INTRODUCCIÓN

Este capítulo explica el código de máquina y proporciona una lista de las instrucciones simbólicas con una explicación de su objetivo.

Muchas instrucciones tienen un objetivo específico, de modo que una instrucción de lenguaje de máquina de un byte es adecuada. Veamos los ejemplos siguientes:

| CÓDIGO DE MÁQUINA | INSTRUCCIÓN SIMBÓLICA | COMENTARIO |
|---|---|---|
| 40 | INC AX | Incrementa AX |
| 50 | PUSH AX | Guarda en la pila AX |
| C3 | RET (cercano) | Regreso corto desde un procedimiento |
| CB | RET (lejano) | Regreso lejano desde un procedimiento |
| PD | STD | Pone en uno la bandera de dirección |

Ninguna de estas instrucciones hace referencia directa a memoria. Instrucciones que especifican un operando inmediato, un registro de ocho bits, dos registros o una referencia a memoria son más complicadas y requieren de dos o más bytes de código de máquina.

| Registros general, base e índice | | | | |
|---|---|---|---|---|
| Bits | w = 0 | w = 1 | Bits para - registros de segmento | |
| 000 | AL | AX/EAX | 000 | ES |
| 001 | CL | CX/ECX | 001 | CS |
| 010 | DL | DX/EDX | 010 | SS |
| 011 | BL | BX/EBX | 011 | DS |
| 100 | AH | SP | 100 | FS |
| 101 | CH | BP | 101 | GS |
| 110 | DH | SI | | |
| 111 | BH | DI | | |

Figura 28-1 Notación de registro

El código de máquina tiene una provisión especial para indicar un registro en particular y otra para hacer referencia a memoria por medio de un byte de modo de direccionamiento.

## NOTACIÓN DE REGISTROS

Las instrucciones que hacen referencia a un registro pueden contener tres bits que indican el registro particular y el bit-w que señala si el ancho es de un byte (0) o de una palabra (1). Además, sólo ciertas instrucciones pueden accesar los registros de segmentos. La figura 28-1 muestra las notaciones completas de registros. Por ejemplo, el número 000 significa AH, si el bit w es 0 y AX si es 1.

Aquí está el código simbólico y de máquina para una instrucción MOV con un operando inmediato de un byte:

```
MOV  AH,00      10110 100 00000000
                    | | | |
                    w reg = AH
```

En este caso, el primer byte del código de máquina indica un ancho de un byte (w = 0) y se refiere al registro AH (100). Veamos una instrucción MOV que contiene un operando inmediato de una palabra, junto con su código de máquina generado:

```
MOV  AX,00      10111 000 00000000 00000000
                    | 111
                    w reg = AX
```

El primer byte del código de máquina indica un ancho de una palabra (w = 1) y se refiere al registro AX (000). En otras instrucciones, w y reg pueden ocupar posiciones diferentes.

## BYTE DEL MODO DE DIRECCIONAMIENTO

El byte de *modo,* cuando está presente, ocupa el segundo byte del código de máquina y consiste en los siguientes tres elementos:

mod Un modo de dos bits, donde los números 00, 01 y 10 se refieren a localidades de memoria

reg Una referencia, de tres bits, a un registro

r/m Una referencia de tres bits a un registro o a memoria, en donde r especifica qué registro y m indica una dirección de memoria

Además, el primer byte del código de máquina puede contener un bit que indica la dirección de flujo. A continuación está un ejemplo de la suma del AX al BX:

```
ADD   BX,AX      00000011  11 011 000
                        dw mod reg r/m
```

En el ejemplo, d = 1 significa que mod (11) y reg (011) describen el primer operando y r/m (000) el segundo operando. Ya que w = 1, el ancho es de una palabra. Por lo tanto, la instrucción es para sumar el AX (000) al BX (011).

El segundo byte del código objeto indica la mayoría de los modos de direccionamiento de memoria. La sección siguiente examina el modo de direccionamiento con mayor detalle.

### Bits de modo

Los dos *bits de modo* distinguen entre direccionamiento de registro y de memoria. Enseguida se explica su objetivo:

00 los bits r/m dan la opción exacta de direccionamiento; no existe byte de desplazamiento.

01 los bits r/m dan la opción exacta de direccionamiento; hay un byte de desplazamiento.

10 los bits r/m dan la opción exacta de direccionamiento; hay dos bytes de desplazamiento.

11 r/m especifica un registro. El bit w (en el byte del código de operación) determina si es una referencia a un registro de 8, 16 o 32 bits.

### Bits reg

Los tres bits reg, en asociación con el bit w, determinan el registro real de 8 o 16 bits.

### Bits r/m

Los tres *bits r/m (registro/memoria),* en asociación con los bits mod, determinan el modo de direccionamiento, como se muestra en la figura 28-2.

| r/m | mod=00 | mod=01 o 10 | mod=ll w=0 | mod=ll w=l |
|---|---|---|---|---|
| 000 | BX+SI | DS:[BX+SI+disp] | AL | AX |
| 001 | BX+DI | DS:[BX+DI+disp] | CL | CX |
| 010 | BP+SI | SS:[BP+SI+disp] | DL | DX |
| 011 | BP+DI | SS:[BP+DI+disp] | BL | BX |
| 100 | SI | DS:[Sl+disp] | AH | SP |
| 101 | DI | DS:[Dl+disp] | CH | BP |
| 110 | Direct | SS: [BP+disp] | DH | SI |
| 111 | BX | DS:[BX+disp] | BH | DI |

Figura 28-2 Los bits r/m

## INSTRUCCIONES DE DOS BYTES

El ejemplo siguiente de una instrucción de dos bytes suma el BX al AX:

```
ADD  AX,BX  0000 0011   11 000 011
                 |I   M III III
                dw  mod reg r/m
```

d = 1 reg más w describen el primer operando y mod más r/m describen el segundo operando
w = 1 El ancho es de una palabra
mod = 11 El segundo operando es un registro
reg = 000 El primer operando es el registro AX
r/m = 011 El segundo operando es el registro BX

El ejemplo siguiente multiplica el AL por el BL:

```
MUL  BL 11110110   11 100 011
                |   || ||| |||
               w  mod reg r/m
```

El procesador supone que el multiplicando está en el AL, si es de un byte, en el AX si es de una palabra y en el EAX si es de una palabra doble. El ancho (w= 0) es un byte, mod (11) hace referencia a un registro y el registro (r/m = 011) es el BL (011). Reg = 100 aquí carece de significado.

## INSTRUCCIONES DE TRES BYTES

El siguiente MOV genera tres bytes de código de máquina:

```
MOV  palabra mem,  AX     10100011. dddddddd dddddddd
                                    I
                                   dw
```

Un movimiento desde el acumulador (AX o AL) sólo necesita conocer si la operación es de un byte o de una palabra. En este ejemplo, w = 1 significa una palabra y se entiende que es el registro de 16 bits AX. (El uso de AL en el segundo operando causaría que el bit w fuera cero.) Los bytes 2 y 3 contienen el desplazamiento a la localidad de memoria. El uso del registro acumulador con frecuencia es más eficiente (a causa de la longitud más corta de la instrucción requerida y su ejecución más rápida) que el uso de otros registros.

## INSTRUCCIONES DE CUATRO BYTES

El ejemplo siguiente de una instrucción de cuatro bytes multiplica el AL por una localidad de memoria:

```
MUL  byte-mem   11110110   00 100 110 x--x x--x
                        |   || |I| ||| 
                       w  mod reg r/m
```

Para esta instrucción, aunque reg es 100, el multiplicando se supone estará en el AL. Mod = OC indica una referencia a memoria y r/m = 110 significa una referencia directa a memoria. La instrucción de máquina también contiene dos bytes subsecuentes que proporcionan el desplazamiento a la localidad de memoria.

El ejemplo siguiente ilustra la instrucción LEA, que especifica una dirección de una palabra:

```
LEA  DX,mem   10001101  00 010 110 x--x x—x
                          I I  III  III
                    LEA   mod reg r/m
```

Reg = 010 designa el registro DX. Mod = 00 y r/m = 110 indican una referencia directa a una dirección de memoria. Los dos bytes subsecuentes proporcionan el desplazamiento de esta localidad.

## CONJUNTO DE INSTRUCCIONES

Esta sección estudia el conjunto de instrucciones en orden alfabético, aunque instrucciones muy estrechamente relacionadas por conveniencia están agrupadas juntas. Además de la discusión anterior del byte de modo y del bit w (ancho), las abreviaturas siguientes son importantes:

| | |
|---|---|
| direc | Dirección de la localidad de memoria |
| direc-alta | Byte de más a la derecha de una dirección |
| direc-baja | Byte de más a la izquierda de una dirección |
| datos | Operando inmediato (si w = 0, es de 8 bits, si w = 1 es de 16) |
| dato-alto | Byte de más a la derecha de un operando inmediato |
| dato-bajo | Byte de más a la izquierda de un operando inmediato |
| despl | Desplazamiento (valor del desplazamiento) |
| reg | Referencia a un registro |

El procesador 80286 y posteriores dan soporte a varias instrucciones especiales no tratadas aquí: ARPL, BOUND, CLTS, ENTER, LAR, LEAVE, LGDT, LIDT, LLDT, LMSW, LSL, LTR, SGDT, SIDT, SLDT, SMSW, STR, VERR y VERW. Instrucciones sólo para el 80486 y posteriores son BSWAP, XADD, CMPXCHG, INVD, WBINVD e INVLPG, tampoco tratadas aquí.

### AAA: Ajusta a ASCII después de sumar

*Operación.* Corrige la suma de dos bytes ASCII en el AL. Si los cuatro bits de más a la derecha del AL tienen un número mayor que 9, o si la bandera AF está en uno, AAA suma 1 al AH y pone en uno las banderas AF y CF. De otra forma, las banderas AF y CF son puestas en cero. AAA siempre pone en cero los cuatro bits de más a la derecha.
*Banderas.* Afecta AF y CF. (OF, PF, SF y ZF están indefinidas.)
*Código fuente.* AAA (sin operando)
*Código objeto,* **o o n o m**

### AAD: Ajusta a ASCII antes de dividir

*Operación.* Corrige para la división de valores ASCII. Utilice AAD antes de dividir entre un número BCD desempacado en el AX. (Quita los 3 ASCII.) AAD corrige el dividendo a un número

binario en el AL para una subsecuente división binaria. Multiplica el AH por 10, suma el producto al AL y pone en cero el AH.
*Banderas.* Afecta PF, SF y ZF. (AF, CF y OF están indefinidas.)
*Código fuente.* ADD (sin operando)
*Código objeto.* `|11010101|00001010|`

**AAM: Ajusta ASCII después de multiplicar**

*Operación.* Corrige el producto generado por la multiplicación de dos números BCD desempacados. AAM divide el AL entre 10 y almacena el cociente en el AH y el residuo en el AL.
*Banderas.* Afecta PF, SF y ZF. (AF, CF y OF están indefinidos.)
*Código fuente.* AAM (sin operando)
*Código objeto.* `|11010100|oooooioio|`

**AAS: Ajusta ASCII después de restar**

*Operación.* Corrige la diferencia de dos bytes ASCII en el AL. Si los cuatro bits de más a la derecha tienen un valor mayor que 9, o si la bandera CF es 1, AAS resta 6 del AL, resta 1 del AH y pone en uno las banderas AF y CF. De otra forma, las banderas AF y CF son puestas en cero. AAS siempre pone en cero los cuatro bits de más a la izquierda del AL.
*Banderas.* Afecta AF y CF. (OF, PF, SF y ZF están indefinidos.)
*Código fuente:* AAS (sin operando)
*Código objeto,* `oo111111`

**ADC: Suma con acarreo**

*Operación.* Por lo común es usado en suma de múltiples palabras binarias para acarrear un bit 1 al siguiente paso de la aritmética. ADC suma el contenido de la bandera CF (0/1) al primer operando, y después suma el segundo operando al primero, al igual que ADD. (Véase también SBB.)
*Banderas.* Afecta AF, CF, OF, PF, SF y ZF.
*Código fuente.* ADC {registro/memoria}, {registro/memoria/inmediato}
*Código objeto.* Tres formatos:

- Reg/mem con registro: `|oooioodw|modreg/m|`
- Inmed a acumulador: `|oooioiow|–dato--|dato si w = i|`
- Inmed a reg/mem: `|iooooosw|modoior/m|–dato–|dato si sw=oi|`

**ADD: Suma números binarios**

*Operación.* Suma números binarios desde la memoria, registro o inmediato a un registro, o suma números en un registro o inmediato a memoria. Los valores pueden ser de un byte, palabra o palabra doble (80386 y posteriores).
*Banderas.* Afecta a AF, CF, OF, PF, SF y ZF
*Código fuente.* ADD {registro/memoria}, {registro/memoria/inmediato}
*Código objeto.* Tres formatos:

- Reg/mem con registro: `|oooooodw|modreg/m|`
- Inmed a acumulador: `|ooooooiow|---dato--|dato si w=i|`
- Inmed a reg/mem: `|iooooosw|modooor/m|–dato–|dato si sw=oi|`

### AND: Conjunción lógica

*Operación.* Realiza una conjunción lógica (AND) sobre los bits de cada operando. Ambos operandos son bytes, palabras o palabras dobles (80386 y posteriores), que AND compara bit a bit. Si ambos bits son 1, el bit 1 en el primer operando es puesto en 1; de otra forma el bit es puesto en cero. (Véase también OR, XOR y TEST.)
*Banderas.* Afecta CF. (0), OF (0), PF, SF Y ZF ( AF está indefinida.)
*Código fuente.* AND {registro/memoria}, {registro/memoria/inmediato}
*Código objeto.* Tres formatos:

- Reg/mem con registro: `|ooiooodw¡modreg/m|`
- Inmed a acumulador: `|ooiooiow|–dato--|dato si w=i|`
- Inmed a reg/mem: `|iooooosw|mod ioor/mj–dato–|dato si sw=i|`

### BSF y BSR: Bit de rastreo (80386 y posteriores)

*Operación.* Busca una cadena de bit para el primer bit en 1. BSF busca de derecha a izquierda y BSR busca de izquierda a derecha. El segundo registro de operando (16 o 32 bits) contiene la cadena que será examinada. La operación regresa la posición del bit (si existe) en el registro del primer operando.
*Banderas.* Afecta ZF.
*CÓdigO fuente.* `BSF/BSR registro, {registro/memoria}`
*CÓdigO Objeto.* `BSF: |OOOOllll|lOllll00|modreg/m|`
`BSR: |00001111|10111101|raodreg/m|`

### BT/BTC /BTR/BTS: Prueba bit (80386 y posteriores)

*Operación.* Copia un bit específico en la bandera CF. El primer operando contiene la cadena de bits que serán examinadas y el segundo indica su posición. BTC complementa el bit inviniendo su valor en el primer operando. BTR reestablece el bit poniéndolo en cero. BTS pone el bit en 1. Referencias son valores de 16 y 32 bits.
*Banderas.* Afecta CF.
*Código fuente.* BT/BTC/BTR/BTS {registro/memoria}, {registro/inmediato}
*Código objeto.* Dos formatos:

- Inmed a reg: `|ooooim 110111010 |mod***r/m|`
- Reg/mem a reg: `|ooooнn 110***010 |modreg/m!`

`(***significa BT = 100, BTC = 111, BTR = 110, BTS = 101)`

### CALL: Llama a un procedimiento

*Operación.* Llama a un procedimiento cercano o lejano. El ensamblador genera un CALL cercano si el procedimiento llamado es NEAR y un CALL lejano si el procedimiento es FAR. Para cercano, CALL guarda en la pila el IP (la dirección de la siguiente instrucción). Después carga el IP con el desplazamiento de la dirección de destino. Para lejano, CALL guarda en la pila el CS y carga un apuntador intersegmento sobre la pila. Después guarda el IP en la pila y carga el IP con el desplazamiento de la dirección destino. Un RET cercano o lejano subsecuente invierte estos pasos para regresar.
*Banderas.* No las afecta.
*Código fuente.* CALL {registro/memoria}

*Código objeto.* Cuatro formatos:

- Directo dentro del segmento: `¡11101000|desp-bajo |despl-alto|`
- Indirecto dentro del segmento: `|11111111|mod0l0r/ml`
- Indirecto entre segmentos: `|11111111|raod011r/m|`
- Directo entre segmentos: `|10011010|desp-bajo|desp-alto|seg-bajo|seg-alto|`

**CBW: Convierte byte a palabra**

*Operación.* Extiende un número de un byte con signo a una palabra duplicando el bit de signo (bit 7) del AL a los bits en el AH. (Véase también CWD, CWDE y CDQ.)
*Banderas.* No las afecta.
*Código fuente.* CBW (sin operando)
*Código objeto.* `10011000`

**CDQ: Convierte palabra doble en palabra cuádruple (80386 y posteriores)**

*Operación.* Extiende un número con signo de 32 bits a uno de 64 bits duplicando el signo (bit 31) de EAX por medio del EDX. (Véase también CBW, CWD y CWDE.)
*Banderas.* No las afecta.
*Código fuente.* CDQ (sin operando)
*Código objeto.* `10011001`

**CLC: Limpia la bandera de acarreo**

*Operación.* Limpia (pone en cero) la bandera CF de modo que por ejemplo, un ADC no sume un bit de 1. (Véase también STC.)
*Banderas.* CF (se pone en cero).
*Código fuente.* CLC (sin operando)
*Código objeto,* `miiooo`

**CLD: Limpia la bandera de dirección**

*Operación.* Pone en cero la bandera DF, para hacer que operaciones como MOVS procesen de izquierda a derecha. (Véase también STD.)
*Banderas.* DF (se pone en cero)
*Código fuente.* CLD (sin operando)
*Código objeto,* `miiioo`

**CLI: Limpia la bandera de interrupción**

*Operación.* Pone en cero la bandera IF, para deshabilitar la interrupción externa enmascarable. (Véase también STI.)
*Banderas.* IF (se pone en cero).
*Código fuente.* CLI (sin operando)
*Código objeto,* `miioio`

**CMC: Complemento de la bandera de acarreo**

*Operación.* Complementa la bandera CF: invierte el bit de CF; 0 se convierte en 1 y 1 se convierte en 0.

*Banderas.* CF (se invierte)
*Código fuente.* CMC (sin operando)
*Código objeto,* **niioioi**

### CMP: Comparar

*Operación.* Compara el contenido de dos campos de datos. CMP resta internamente el segundo operando del primero y pone en uno o en cero las banderas, pero no almacena el resultado. Ambos operandos son byte, palabra o palabra doble (80386 y posteriores). CMP puede comparar registro, memoria o inmediato con un registro o comparar registro o inmediato con memoria. (Véase también CMPS.)
*Banderas.* Afecta AF, CF, OF, PF, SF y ZF.
*Código fuente.* CMP {registro/memoria}, {registro/memoria/inmediato}
*Código objeto.* Tres formatos:

- Reg/mem con reg: |**oomodw|modreg/m|**
- Inmed a acumulador: |**ooimow|—datos--|datos si w=i|**
- Inmed a reg/mem: |**iooooosw|modiiir/m|—dato———|datos si sw=o|**

### CMPS/CMPSB/CMPSW/CMPSD: Compara cadenas

*Operación.* Compara cadenas de cualquier longitud en la memoria. Por lo regular, un prefijo REPn precede estas instrucciones, junto con un valor máximo en el CX. CMPSB compara bytes, CMPSW compara palabras y CMPSD (80386 y posteriores) compara palabras dobles. Los registros DS:SI direccionan el primer operando y los registros ES:DI direccionan el segundo. Si la bandera DF es 0, la operación compara de izquierda a derecha e incrementa el SI y el DI; si el DF es 1, compara de derecha a izquierda y disminuye el SI y el DI. REPn disminuye el CX en uno en cada repetición. La operación termina cuando el valor comparado es encontrado (REPNE), cuando no es encontrado (REPE) o cuando el CX es disminuido hasta cero; el DI y el SI son avanzados pasando el byte que causó la terminación. La última comparación pone en uno/cero las banderas, no el contenido del CX.
*Banderas.* Afecta AF, CF, OF, PF, SF y ZF.
*Código fuente.* [REPnn]CMPSB/CMPSW/CMPSD (sin operando)
*Código objeto.* **íoiooiiw**

### CWD: Convierte palabra en palabra doble

*Operación.* Extiende un número con signo de una palabra a una palabra doble en el DX:AX duplicando el signo (bit 15) del AX por medio del DX, por lo regular para generar un dividendo de 32 bits. (Véase también CBW, CWDE y CDQ.)
*Banderas.* No las afecta.
*Código fuente.* CWD (sin operando)
*Código objeto.* **10011001**

### CWDE: Convierte una palabra a una palabra doble ampliada (80386 y posteriores)

*Operación.* Extiende un número con signo de una palabra a una palabra doble en la EAX duplicando el signo (bit 15) del AX, por lo regular para generar un dividendo de 32 bits. (Véase también CBW, CWD y CDQ.)

*Banderas.* No las afecta.
*Código fuente.* CWDE (sin operando)
*Código objeto,* **i o o 11 o o o**

**DAA: Ajusta decimal después de sumar**

*Operación.* Corrige el resultado de la suma de dos elementos empacados BCD en el AL. Si los cuatro bits de más a la derecha tienen un valor mayor que 9, o si la bandera AF es uno, DAA suma 6 al AL y pone en uno la AF. Si el AL contiene un valor mayor que 99H o si la bandera CF es uno, DAA suma 60H al AL y pone en uno el CF. De otra forma, el AF y el CF son puestos en cero. (Véase también DAS.)
*Banderas.* Afecta AF, CF, PF, SF y ZF. (OF está indefinido.)
*Código fuente.* DAA (sin operando)
*Código objeto,* **o o i o o m**

**DAS: Ajusta decimal después de restar**

*Operación.* Corrige el resultado de restar dos elementos BCD empacados en el AL. Si los cuatro bits de más a la derecha tienen un valor mayor que 9, o si la bandera AF está en uno, DAS resta 60H del AL y pone en uno la bandera CF. De otra forma, el AF y el CF se ponen en cero. (Véase también DAA.)
*Banderas.* Afecta AF, CF, PF, SF y ZF. (OF está indefinido.)
*Código fuente.* DAS (sin operando)
***CÓdigO Objeto.*** **OOlOllll (sin operando)**

**DEC: Disminuye en uno**

*Operación.* Disminuye 1 de un byte, palabra o palabra doble (80386 y posteriores) en un registro o memoria. (Véase también INC.)
*Banderas.* Afecta AF, OF, PF, SF y ZF.
*Código fuente.* DEC {registro/memoria}
*Código objeto.* Dos formatos:

- Registro: **| OlOOlreg |**
- Reg/memoria: i i i i i i i i w i m o d o o i r / m i

**DIV: Divide sin signo**

*Operación.* Divide un dividendo sin signo entre un divisor sin signo. DIV trata un bit 1 de más a la izquierda como un bit de dato, no como un bit de signo. La división entre cero provoca una interrupción de división entre cero. (Véase también IDIV.) Aquí están las operaciones para byte, palabra y palabra doble:

| Tamaño | Dividendo (operando 1) | Divisor (operando 2) | Cociente | Residuo | Ejemplo |
|---|---|---|---|---|---|
| 16-bit | AX | 8-bit reg/memoria | AL | AH | DIV BH |
| 32-bit | DX:AX | 16-bit reg/memoria | AX | DX | DIVCX |
| 64-bit | EDX:EAX | 32-bit reg/memoria | EAX | EDX | DIV ECX |

*Banderas.* Afecta AF, CF, OF, PF, SF y ZF. (Todas no están definidas.)
*Código fuente.* DIV {registro/memoria}
*CÓdigO Objeto.* `|llllOllw|modllOr/m|`

**ESC: Escape**

*Operación.* Facilita el uso de coprocesadores tales como el 80x87 realiza operaciones especiales. ESC proporciona al coprocesador una instrucción y un operando para ejecución. Note que la versión 6.1, MASM ya no da más soporte a ESC; en lugar de eso, genera el código objeto completo para las instrucciones de coprocesador.
*Banderas.* No las afecta.
*Código fuente.* ESC inmediato, {registro/memoria}
*Código objeto.* `|iioiixxx|modxxxr/m|` (los bits x no son importantes)

**HLT: Entra a un estado de suspensión (detención)**

*Operación.* Provoca que el procesador ingrese a un estado de detención mientras espera por una interrupción. HLT termina con los registros CS e IP apuntando a la instrucción que sigue al HLT. Cuando ocurre una interrupción, el procesador guarda en la pila el CS y el IP y ejecuta la rutina de interrupción. Al regresar, una instrucción IRET saca de la pila, y el proceso continúa después del HLT original.
*Banderas.* No las afecta.
*Código fuente.* HLT (sin operando)
*Código objeto,* **niioioo**

**IDIV: División (entera) con signo**

*Operación.* Divide un dividendo con signo entre un divisor con signo. IDIV trata el bit de más a la izquierda como el signo (0 = positivo, 1 = negativo). La división entre cero provoca una interrupción de división entre cero. (Véase CBW y CWD para ampliar la longitud del dividendo con signo, y véase también DIV.) A continuación están las operaciones de división para byte, palabra y palabra doble:

| Tamaño | Dividendo (operando 1) | Divisor (operando 2) | Cociente | Residuo | Ejemplo |
|---|---|---|---|---|---|
| **16-bit** | **AX** | **8-bit** reg/**memoria** | **AL** | **AH** | **IDIV BH** |
| 32-**bit** | **DX:AX** | **16-bit** reg/**memoria** | **AX** | **DX** | **IDIV CX** |
| **64-bit** | **EDX:EAX** | 32-**bit** reg/**memoria** | **EAX** | **EDX** | **IDIV ECX** |

*Banderas.* Afecta AF, CF, OF, PF, SF y ZF.
*Código fuente.* IDIV {registro/memoria}
*CÓdigO Objeto.* `|1111011w|MODlllr/m|`

**IML1L: Multiplicación (entera) con signo**

*Operación.* Multiplica un multiplicando con signo por un multiplicador con signo. IMUL trata el bit de más a la izquierda como el signo (0 = positivo, 1 = negativo). (Véase también MUL.) A continuación están las operaciones de multiplicación por byte, palabra y palabra doble:

| Tamaño | Multiplicando (operando 1) | Multiplicador (operando 2) | Producto | Ejemplo |
|---|---|---|---|---|
| 8-bit | AL | 8-bit registro/memoria | AX | IMUL BL |
| 16-bit | AX | 16-bit registro/memoria | DX:AX | IMUL BX |
| 32-bit | EAX | 32-bit registro/memoria | EDX:EAX | IMUL ECX |

*Banderas.* Afecta CF y OF. (AF, PF, SF y ZF están indefinidos.)
*Código fuente.* IMUL {registro/memoria} (todos los procesadores)
*Código objeto.* |iinoiiw|modioir/m| (primer formato)

Otros tres formatos están disponibles para procesadores avanzados:

- IMUL registro,inmediato (80286 y posteriores)
- IMUL registro,registro,inmediato (80286 y posteriores)
- IMUL registro, {registro/memoria} (80386 y posteriores)

### IN: Entrada de un byte o palabra

*Operación.* Transfiere desde un puerto de entrada un byte al AL o una palabra al AX. Codifique el puerto como un operando numérico fijo (como IN AX,#puerto) o como una variable en el DX (como IN AX;DX). Utilice el DX si el número del puerto es mayor que 256. El procesador 80286 y posteriores también dan soporte a una instrucción INS (Entrada de cadena). (Véase también OUT.)
*Código fuente.* IN {AL/AX},{númpuerto/DX}
*Banderas.* No las afecta
*Código objeto.* Dos formatos:

- Puerto variable: 1111011ow|
- Puerto fijo: |niooiow|--puerto--1

### ENC: Incrementa en 1

*Operación.* Incrementa en uno un byte, palabra o palabra doble (80386 y posteriores) en un registro o memoria, codificado, por ejemplo, como INC CX. (Véase también DEC.)
*Código de fuente.* Afecta AF, OF, PF, SF y ZF.
*Banderas.* INC {registro/memoria}
*Código objeto.* Dos formatos:

- Registro: **|01000reg|**
- Reg/memoria: |iiiiiiiw|modooor/m|

### ENT: Interrupción

*Operación.* Interrumpe el procesamiento y transfiere el control a uno de las 256 direcciones (vector) de interrupción empezando en el segmento 0, desplazamiento 0. INT realiza lo siguiente: (1) guarda en la pila las banderas y restablece las banderas IF y TF; (2) guarda en la pila el CS y coloca la palabra de orden alto de la dirección de interrupción en el CS; y (3) guarda en la pila el

IP y llena el IP con la palabra de orden bajo de la dirección de interrupciones. Para el 80396 y posteriores, INT guarda en la pila un IP de 16 bits para segmentos de 16 bits y un IP de 32 bits para segmentos de 32 bits. IRET regresa de una rutina de interrupción.
*Banderas.* Pone en cero IF y TF.
*Código fuente.* INT número
*CÓdigO Objeto.* `|ll00110v[--tipo--|  (si v = 0 tipo es 3)`

**LNTO: Interrumpe un desbordamiento**

*Operación.* Provoca una interrupción (generalmente inofensiva) si ha ocurrido un desbordamiento (la OF está en uno) y realiza una INT 04H. La dirección de interrupción está en la localidad 10H de la tabla de servicios de interrupción. (Véase también INT.)
*Banderas.* Afecta a IF y TF.
*Código fuente.* INTO (sin operando)
*Código objeto.* **íioomo**

**IRET/IRETD: Regresa de una interrupción**

*Operación.* Proporciona un regreso lejano de una rutina de interrupción. IRET realiza el siguiente procedimiento: (1) saca de la pila la palabra de la parte superior de la pila al IP, incrementa en dos el SP y saca de la parte superior de la pila al CS; (2) incrementa en 2 el SP y saca de la pila al registro de bandera. Este proceso deshace los pasos de la interrupción original y realiza un regreso. Para el 80386 y posteriores, utilice IRETD (palabra doble) para sacar de la pila un IP de 32 bits. (Véase también RET.)
*Banderas.* Afecta a todas.
*Código fuente.* IRET
*CÓdigO Objeto.* `llOOllll (sin operando)`

**JA/JNBE: Salta si es superior o si no es inferior/igual**

*Operación.* Utilizada después de una prueba de datos sin signo. Si la bandera CF es 0 (no hay acarreo) y la bandera ZF es cero (una condición de no cero), la instrucción suma el desplazamiento del operando al IP y realiza un salto. El salto debe ser corto (-128 a 127 bytes, salvo para el 80386 y posteriores, en los cuales puede ser cercano (dentro de 32K).
*Banderas.* No las afecta.
*Código fuente.* JA/JNBE etiqueta
*Código objeto.* `|oiiioin|--despl--1`

**JAE/JNB: Salta si es superior/igual o salta si no es inferior**

*Operación.* Usado después de una prueba de datos sin signo. Si la bandera CF es 0 (no hay acarreo), la instrucción suma al operando el desplazamiento del IP y realiza un salto. El salto debe ser corto (-128 a 127 bytes), salvo para el 80386 y posteriores, en los cuales puede ser cercano (dentro de 32K).
*Banderas.* No las afecta.
*Código fuente.* JAE/JNB etiqueta
*CÓdigO Objeto.* `|OlllOOll|--despl-1z`

**JB/JNAE: Salta si es inferior o salta si no es superior/igual**

*Operación.* Usado después de una prueba de datos sin signo. Si la bandera CF es 1 (hay acarreo), la instrucción suma al operando el desplazamiento del IP y realiza un salto. El salto debe ser corto (-128 a 127 bytes), salvo para el 80386 y posteriores, en los cuales puede ser cercano (dentro de 32K).
*Banderas.* No las afecta.
*Código fuente.* JB/JNAE etiqueta
*CÓdigO Objeto.* `| OlllOOlO | - -despl- ·`1

**JBE/JNA: Salta si es inferior/igual o salta si no es superior**

*Operación.* Usado después de una prueba de datos sin signo. Si la bandera CF es 1 (hay acarreo) o la bandera AF es 1, la instrucción suma al operando el desplazamiento del IP y realiza un salto. El salto debe ser corto (-128 a 127 bytes), salvo para el 80386 y posteriores, en los cuales puede ser cercano (dentro de 32K).
*Banderas.* No las afecta.
*Código fuente.* JBE/JNA etiqueta
*CÓdigO objeto.* `| OlllOllO | - -despl- - |`

**JC: Salta si hay acarreo**

*Operación.* Véase JB/JNAE (operaciones idénticas).

**JCXZ/JECXZ: Salta si CX/ECX es cero**

*Operación.* Salta a una dirección específica si el CX o el ECX (80386 y posteriores) contiene cero. Esta operación podría ser útil al inicio de un ciclo, aunque limitada a un salto corto.
*Banderas.* No las afecta.
*Código fuente.* JCXZ/JECXZ etiqueta
*CÓdigO objeto.* `| lllOOOll | - -despl- - |`

**JE/JZ: Salta si es igual o salta si es cero**

*Operación.* Usado después de una prueba de datos con o sin signo. Si la bandera ZF es 1 (condición de cero), la instrucción suma al operando el desplazamiento del IP y realiza un salto. El salto debe ser corto (-128 a 127 bytes), salvo para el 80386 y posteriores, en los cuales puede ser cercano (dentro de 32K).
*Banderas.* No las afecta.
*Código fuente.* JE/JZ etiqueta
*CÓdigO objeto.* `| OlllOlOO | - -despl- - |`

**JG/JNLE: Salta si es mayor o salta si no es menor/igual**

*Operación.* Usado después de una prueba de datos con signo. Si la bandera ZF es 0 (condición de no cero) y la bandera SF.es igual a la OF (ambas 0 o ambas 1), la instrucción suma al operando el desplazamiento del IP y realiza un salto. El salto debe ser corto (-128 a 127 bytes), salvo para el 80386 y posteriores, en los cuales puede ser cercano (dentro de 32K).
*Banderas.* No las afecta.
*Código fuente.* JG/JNLE etiqueta
*CÓdigO objeto.* `| Olllllll | --despl- - |`

**JGE/JNL: Salta si es mayor/igual o salta si no es menor**

*Operación.* Usado después de una prueba de datos con signo. Si la bandera SF es igual a la **OF1** (ambas 0 o ambas 1), la instrucción suma al operando el desplazamiento del IP y realiza un salto. El salto debe ser corto (-128 a 127 bytes), salvo para el 80386 y posteriores, en los cuales puede; ser cercano (dentro de 32K).
*Banderas.* No las afecta.
*Código fuente.* JGE/JNL etiqueta
*CÓdigO Objeto.* `|ollllol|--despl--|`

**JL/JNGE: Salta si es menor o salta si no es mayor/igual**

*Operación.* Usado después de una prueba de datos con signo. Si la bandera SF no es igual a la OF, la instrucción suma al operando el desplazamiento del IP y realiza un salto. El salto debe ser corto (-128 a 127 bytes), salvo para el 80386 y posteriores, en los cuales puede ser cercano (dentro de 32K).
*Banderas.* No las afecta.
*Código fuente.* JL/JNGE etiqueta
*CÓdigO Objeto.* `|ollll0|--despl--|`

**JLE/JNG: Salta si es menor/igual o salta si no es mayor**

*Operación.* Usado después de una prueba de datos con signo. Si la bandera ZF es 1 (condición de! cero) o si la bandera SF no es igual a la OF, la instrucción suma al operando el desplazamiento delj IP y realiza un salto. El salto debe ser corto (-128 a 127 bytes), salvo para el 80386 y posteriores, en los cuales puede ser cercano (dentro de 32K).
*Banderas.* No las afecta.
*Código fuente.* JLE/JNG etiqueta
*Código objeto.* \ **o m i n o** `|--despl--1`

**JMP: Salto incondicional**

*Operación.* Salta a una dirección designada bajo cualquier condición. Una dirección de JMP puedej ser corta (-128 a +127 bytes), cercana (dentro de 32K) o lejana (a otro segmento). Un JMP corto: reemplaza el IP con eí desplazamiento de la dirección de destino. Un salto lejano (como JMP FARi PTR etiqueta) reemplaza el CS:IP con una nueva dirección de segmento.
*Banderas.* No las afecta. **j**
*Código fuente.* JMP {registro/memoria} **j**
*Código objeto.* Cinco formatos: ]

- Directo dentro de un seg corto: `11101011|--despl--|`
- Directo dentro de segmento: `11101001|despl-bajo|despl-alto|` j
- Indirecto dentro del segmento: `11111111|modlOOr/m|` ]
- Indirecto entre segmento: `11111111|modl01r/m|` 1
- Directo entre segmento: `11101010|despl-bajojdespl-altojseg-bajo seg alto|`

**JNC: Salta si no hay acarreo**

*Operación.* Véase JAE/JNB (operaciones idénticas).

### JNE/JNZ: Salta si no es igual o salta si no es cero

*Operación.* Usado después de una prueba de datos con signo. Si la bandera ZF es 0 (condición de no cero), la instrucción suma al operando el desplazamiento del IP y realiza un salto. El salto debe ser corto (-128 a 127 bytes), salvo para el 80386 y posteriores, en los cuales puede ser cercano (dentro de 32K).
*Banderas.* No las afecta.
*Código fuente.* JNE/JNZ etiqueta
*CÓdigO Objeto.* **| Olll0l0l | - - despl- - |**

### JNO: Salta si no hay desbordamiento

*Operación.* Salta si una operación provoca un desbordamiento. Si la bandera OF es 0, la instrucción suma al operando el desplazamiento del IP y realiza un salto. El salto debe ser corto (-128 a 127 byte), salvo para el 80386 y posteriores, en los cuales puede ser cercano. (Véase también JO.)
*Banderas.* No las afecta.
*Código fuente.* JNO etiqueta
*CÓdigO Objeto.* **| Olll000l | - - despl- - |**

### JNP/JPO: Salta si no hay paridad o salta si la paridad es impar

*Operación.* Salta si una operación no provoca paridad (o impar) —esto es, una operación que toma un número impar de bits en los ocho bits de orden bajo—. Si la bandera PF es cero (paridad impar), la instrucción suma al operando el desplazamiento del IP y realiza un salto. El salto debe ser corto (-128 a 127 bytes), salvo para el 80386 y posteriores, en los cuales puede ser cercano. (Véase también JP/JPE.)
*Banderas.* No las afecta.
*Código fuente.* JNP/JPO etiqueta
*CÓdigO Objeto.* \ **Olllloll | - - despl- - |**

### JNS: Salta si no hay signo

*Operación.* Salta si una operación pone el signo en positivo. Si la bandera SF es 0 (positivo), JNS suma al operando el desplazamiento del IP y realiza un salto. El salto debe ser corto (-128 a 127 bytes), salvo para el 80386 y posteriores, en los cuales puede ser cercano. (Véase también JS.)
*Banderas.* No las afecta.
*Código fuente.* JNS etiqueta
*Código objeto.* | o m i o o i | **- - despl- -** 1

### JO: Salta si hay desbordamiento

*Operación.* Salta si una operación causa un desbordamiento. Si la bandera OF es 1 (desbordamiento), JO suma al operando el desplazamiento del IP y realiza un salto. El salto debe ser corto (-128 a 127 bytes), salvo para el 80386 y posteriores, en los cuales puede ser cercano. (Véase también JNO.)
*Banderas.* No las afecta.
*Código fuente.* JO etiqueta
*Código objeto.* \ o m o o o o | **- - despl- -** 1

**JP/JPE: Salta si hay paridad o salta si la paridad es par**

*Operación.* Salta si una operación causa una paridad par —esto es, una operación que pone un número par de bits en uno en los ocho bits de orden bajo—. Si la bandera PF es 1 (paridad par), la instrucción suma al operando el desplazamiento del IP y realiza un salto. El salto debe ser corto (-128 a 127 bytes), salvo para el 80386 y posteriores, en los cuales puede ser cercano. (Véase también JNP/JPO.)
*Banderas.* No las afecta.
*Código fuente.* JP/JPE etiqueta
*CÓdigO Objeto.* `|ollllol0|--despl--|`

**JS: Salta si hay signo**

*Operación.* Salta si una operación pone el signo en negativo. Si la bandera SF es 1 (negativo), JS suma al operando el desplazamiento del IP y realiza un salto. El salto debe ser corto (-128 a 127 bytes), salvo para el 80386 y posteriores, en los cuales puede ser cercano. (Véase también JNS.)
*Banderas.* No las afecta.
*Código fuente.* JS etiqueta
*CÓdigO objeto.* `|olllloOO|--despl--|`

**LAHF: Carga el AH con las banderas**

*Operación.* Carga los ocho bits de más a la derecha del registro de banderas y los envía al AH. (Véase también SAHF.)
*Banderas.* No las afecta.
*Código fuente.* LAHF (sin operando)
*Código objeto,* **i o o 1 1 1 1 1**

**LDS/LES/LFS/LGS/LSS: Carga el registro de segmento**

*Operación.* Inicializa una dirección lejana y desplazamiento de un elemento de datos de modo que las instrucciones que siguen puedan accesarla. El primer operando hace referencia a cualquiera de los registros general, índice o apuntador. El segundo operando hace referencia a cuatro bytes en memoria que contiene un desplazamiento y una dirección de segmento. La operación carga la dirección de segmento en el registro del segmento y la dirección del desplazamiento en el registro del primer operando. Por ejemplo, LDS significa cargar el registro del segmento de datos. LFS, LGS y LSS les da soporte por el 80386 y posteriores.
*Banderas.* No las afecta
*Código fuente.* LDS/LES/LFS/LGS/LSS registro,memoria
*Código objeto,*

```
LDS  I1100Ü101Imod reg r/m|
LES  I11000100Imod reg r/m¡
LFS   00001111I10110100Imod reg r/ml
LGS  I00001111I10110101¡mod reg r/ml
LSS  I00001111I10110010Imod reg r/ml
```

**LEA: Carga dirección efectiva**

*Operación.* Carga una dirección (desplazamiento) cercana en un registro.
*Banderas.* No las afecta.
*Código fuente.* LEA registro,memoria
*Código objeto,* **i o o o 1 1 o i**

### LES/LFS/LGS: Carga un registro de segmento extra

*Operación.* Véase LDS.

### LOCK: Bloquea el bus

*Operación.* Previene que el 80x87 u otros coprocesadores cambien datos al mismo tiempo que el procesador. LOCK es un prefijo de un byte que puede codificar inmediatamente antes de cualquier instrucción. La operación envía al coprocesador para prevenirlo de usar los datos hasta que la siguiente instrucción sea completada.
*Banderas.* No las afecta.
*Código fuente.* LOCK instrucción
*Código objeto.* **1111 0 0 0 0**

### LODS/LODSB/LODSW/LODSD: Carga una cadena de un byte, una palabra o una palabra doble

*Operación.* Carga el registro acumulador con un valor de la memoria. Aunque LODS es una operación de cadena, no requiere de un prefijo REP. Los registros DS:SI direccionan un byte (si LODSB), una palabra (si LODSW) o palabra doble (si LODSD) y la cargan desde la memoria a AL, AX o EAX, respectivamente. Si la bandera DF es 0, la operación suma 1 (si es byte), 2 (si es palabra) o 4 (si es palabra doble) al SI; de otra forma resta 1, 2 o 4.
*Banderas.* No las afecta.
*Código fuente.* LODSB/LODSW/LODSD (sin operando)
*Código objeto.* **íoionow**

### LOOP/LOOPW/LOOPD: Repite hasta completar

*Operación.* Controla la ejecución de una rutina un número específico de veces. El CX debe contener un contador antes de iniciar el ciclo. LOOP aparece al final del ciclo y disminuye el CX en 1. Si el CX no es cero, LOOP transfiere a su dirección del operando, que apunta al inicio del ciclo (añade el desplazamiento en el IP); de otra forma LOOP pasa por la instrucción siguiente. El desplazamiento debe ser un salto corto.

Para el 80386 y posteriores, LOOP utiliza el CX en el modo de 16 bits y el ECX en el modo de 32 bits. LOOPW puede especificar el CX de 16 bits, y LOOPD puede especificar el ECX de 32 bits.
*Banderas.* No las afecta.
*Código fuente.* LOOP etiqueta
*CÓdigO objeto.* **|HIOOOIO|--despl--|**

### LOOPE/LOOPZ: Repite mientras sea igual o cero

*Operación.* Controla la-ejecución repetitiva de una rutina. LOOPE y LOOPZ son semejantes a LOOP, salvo que transfieren a la dirección del operando si el CX no es cero y la bandera ZF es 1 (condición de cero). (Véase también LOOPNE/LOOPNZ.)

Para el 80386 y posteriores, LOOPE y LOOPZ utilizan el CX en modo de 16 bits y el ECX en modo de 32 bits. LOOPEW y LOOPZW pueden especificar el CX de 16 bits y LOOPED y LOOPZD pueden especificar el ECX de 32 bits.

*Banderas.* No las afecta.
*Código fuente.* LOOPE/LOOPZ etiqueta
*CÓdigO Objeto.* `\lll0000l|--despl--|`

**LOOPNE/LOOPNZ: Repite mientras no sea igual o cero**

*Operación.* Controla la ejecución repetitiva de una rutina. LOOPNE y LOOPNZ son semejantes a LOOP, salvo que transfieren a la dirección del operando si el CX no es cero y la bandera ZF es cero (condición de no cero). (Véase también LOOPE/LOOPZ.)

Para el 80386 y posteriores, LOOPNE y LOOPNZ utilizan el CX en el modo de 16 bits y el ECX en el modo de 32 bits. LOOPNEW y LOOPNZW pueden especificar el CX de 16 bits, y LOOPNED y LOOPNZD pueden especificar el ECX de 32 bits.
*Banderas.* No las afecta.
*Código fuente.* LOOPNE/LOOPNZ etiqueta
*CÓdigO Objeto.* `|lll00000|--despl--|`

**LSS: Carga el registro de segmento de la pila**

*Operación.* Véase LDS.

**MOV: Mueve datos**

*Operación.* Transfiere datos entre dos registros o entre un registro y memoria y transfiere datos inmediatos a un registro o memoria. Los datos referenciados definen el número de bytes (1, 2 o 4) movidos; los operandos deben coincidir en tamaño. MOV no puede transferir entre dos localidades de memoria (utilice MOVS), desde datos inmediatos a un registro de segmento o desde un registro de segmento a un registro de segmento. (Véase también MOVSX/MOVZX.)
*Banderas.* No las afecta.
*Código fuente.* MOV {registro/memoria}, {registro/memoria/inmediato}
*Código objeto.* Siete formatos:

- Reg/mem a/desde reg: `100 0010W|modregr/m`
- Inmed a reg/mem: `ll000llwjmod000r/mj____datos--|datos`
- Inmed a registro: `1011wreg|____datos--|datos si w=l|`
- Mem a acumulador: `101000 0w|dir-baja|dir-alta|`
- Acumulador a mem: `1010001w|dir-baja|dir-alta|`
- Reg/mem a seg reg: `10001110|mod0sgr/mj (seg = seg reg)`
- Seg reg a reg/mem: `10001100|mod0sgr/m¡ (seg = seg reg)`

**MOVS/MOVSB/MOVSW/MOVSD: Mueve cadena de caracteres**

*Operación.* Mueve datos entre localidades de memoria. Por lo común usada con el prefijo REP y una longitud en el CX, MOVSB mueve bytes, MOVSW mueve palabras y MOVSD (80386 y posteriores) mueve palabras dobles. El primer operando es direccionado por el ES:DI, el segundo operando por el DS:SI. Si la bandera DF es 0, la operación mueve datos de izquierda a derecha en el destino del primer operando e incrementa el DI y el SI en 1, 2 o 4. Si el DF es 1, la operación mueve datos de derecha a izquierda y disminuye el DI y el SI. REP disminuye en 1 el CX en cada

repetición. La operación termina cuando el CX es disminuido a cero; el DI y el SI son avanzados pasando el último byte movido.
*Banderas.* No las afecta.
*Código fuente.* [REP] MOVSB/MOVSW/MOVSD (sin operando)
*Código objeto,* **IOIOOIOW**

### MOVSX/MOVSZX: Mueve con signo ampliado o cero ampliado (80386 y posteriores)

*Operación.* Copia un operando origen de 8 o 16 bits a un operando destino de 16 o 32 bits. MOVSX llena con el bit de signo los bits de más a la izquierda y MOVZX llena con cero los bits.
*Banderas.* No las afecta.
*Código fuente.* MOVSX/MOVZX {registro/memoria}, {registro/memoria/inmediato}
*CÓdigO Objeto.* `MOVSX: I OOOOllll 1lOllllllwlmodregr/m`
`MOVZX: |000011111 lOUOllwI modregr/m|`

### MUL: Multiplica sin signo

*Operación.* Multiplica un multiplicando sin signo por un multiplicador sin signo. MUL trata un bit en uno de más a la izquierda como un bit de dato, no como un signo negativo. (Véase también IMUL.) A continuación están las operaciones de multiplicación para byte, palabra y palabra doble:

| Tamaño | Multiplicando (operando 1) | Multiplicador (operando 2) | Producto | Ejemplo |
|---|---|---|---|---|
| 8-bit | AL | 8-bit registro/memoria | AX | MUL BL |
| 16-bit | AX | 16-bit registro/memoria | DX:AX | MUL BX |
| 32-bit | EAX | 32-bit registro/memoria | EDX:EAX | MUL ECX |

*Banderas.* Afecta CF y OF. (AF, PF, SF y ZF están indefinidos.)
*Código fuente.* MUL {registro/memoria}
*CÓdigO objeto.* **|1111011w|modl00r**/m|

### NEG: Niega

*Operación.* Invierte un número binario de positivo a negativo y de negativo a positivo. NEG proporciona el complemento a dos del operando especificado restando el operando de cero y sumando uno. Los operandos pueden ser un byte, palabra o palabra doble (80386 y posteriores) en un registro o en memoria. (Véase también NOT.)
*Banderas.* Afecta AF, CF, OF, PF, SF y ZF.
*Código fuente.* NEG {registro/memoria}
*CÓdigO Objeto.* **| 1111011w|MOD011r**/m|

### NOP: No operación

*Operación.* Usada para borrar o insertar código de máquina o para retrasar la ejecución para fines de tiempo. NOP sólo realiza una operación nula ejecutando XCHG AX,AX.

*Banderas.* No las afecta.
*Código fuente.* NOP (sin operando)
*Código objeto.* **10010000**

### NOT: Negación lógica (NOT)

*Operación.* Cambia los bit en cero por bits en uno y viceversa. El operando es un byte, palabra o palabra doble (80386 y posteriores) en un registro o en memoria. (Véase también NEG.)
*Banderas.* No las afecta.
*Código fuente.* NOT {registro/memoria}
*Código Objeto.* `Illll0llwlmod 010 r/ml`

### OR: Disyunción lógica (OR)

*Operación.* Realiza una operación de disyunción sobre los bits de dos operandos. Ambos operandos son bytes, palabras o palabras dobles (80386 y posteriores), que OR compara bit a bit. Si cualquiera es uno, el bit del primer operando se pone en uno; de otra forma el bit queda sin cambio. (Véase también AND y XOR.)
*Banderas.* Afecta CF (0), OF (0), PF, SF y ZF. (AF está indefinida.)
*Código fuente.* OR {registro/memoria}, {registro/memoria/inmediato}
*Código objeto.* Tres formatos:

- Reg/mem con registro: | **oooo iodw** `|modregr/m|`
- Inmed a acumulador: `I` **oooon** `0wi__datos--|datos si w=l|`
- Inmed a reg/mem: `i iooooooswimodooir/m¡--datos--ídatos si w=i i`

### OUT: Salida de un byte o palabra

*Operación.* Transfiere un byte desde el AL o una palabra desde el AX a un puerto de salida. El puerto es un operando numérico fijo (como OUT #puerto,AX) o una variable en el DX (como OUT DX,AX). Utilice el DX si el número del puerto es mayor que 256. El procesador 80286 y posteriores también dan soporte a una instrucción OUTS (Salida de cadena de caracteres). (Véase también IN.)
*Banderas.* No las afecta.
*Código fuente.* OUT {númpuerto/DX},{AL/AX}
*Código objeto.* Puerto variable: **1111** o **11 iw |**
Puerto fijo: | **1110011w** `|--puerto--]`

### POP: Saca una palabra de la pila

*Operación.* Saca de la pila una palabra o una palabra doble (80386 y posteriores) previamente guardada, y la envía a un destino especificado —una localidad de memoria, un registro general o un segmento de registro (salvo el CS; utilice RET para esto). El SP apunta a la palabra actual en el tope de la pila; POP la transfiere al destino especificado e incrementa el SP en 2 o 4. (Véase también PUSH.)
*Banderas.* No las afecta.
*Código fuente.* POP {registro/memoria}
*Código objeto.* Tres formatos:

- Registro: |01011reg|
- Segmento reg: |000sg|||| **(sg implica segmento reg)**
- Reg/memoria: 110001111 **|mod ooo r/m|**

POPA: **Saca de la pila todos los registros generales (80286** y **posteriores)**

*Operación.* Saca de la pila ocho palabras y las envía a DI, SI, BP, SP, BX, DX, CX y AX, en ese orden e incrementa el SP en 16. Por lo común, un PUSHA ha guardado los registros. Para el 80386 y posteriores, POPAD maneja palabras dobles e incrementa el SP en 32. El valor del SP es descartado en lugar de ser guardado.
*Banderas.* No las afecta.
*Código fuente.* POPA/POPAD (sin operando)
*Código objeto,* **ono oooi**

**POPF: Saca de la pila las banderas**

*Operación.* Saca de la pila la palabra de la parte superior y la envía al registro de banderas e incrementa el SP en 2. Por lo común un PUSHF ha guardado en la pila las banderas. Para el 80386 y posteriores, POPFD maneja palabras dobles e incrementa el SP en 4.
*Banderas.* Afecta todas.
*Código fuente.* POPF/POPFD (sin operando)
*Código objeto.* 10011101

**PUSH: Guarda en la pila**

*Operación.* Guarda en la pila una palabra o una palabra doble (80386 y posteriores) para uso posterior. El registro SP apunta a la palabra (doble) actual en el tope de la pila. PUSH disminuye en 2 o en 4 el SP y transfiere una palabra (doble) desde el operando especificado al nuevo tope de la pila. El origen puede ser un registro general, un registro de segmento o memoria. (Véase también POP y PUSHF.)
*Banderas.* No las afecta.
*Código fuente.* PUSH {registro/memoria} (todos los procesadores)
PUSH inmediato (80286 y posteriores)
*Código objeto.* Tres formatos

- Registro: |01010reg|
- Segmento reg: |000sg||0| **(sg implica segmento reg)**
- Reg/memoria: **| í i i i i m |modiior/m|**

**PUSHA: Guarda en la pila todos los registros generales (80286** y **posteriores)**

*Operación.* Guarda en la pila AX, CX, DX, BX, SP, BP, SI y DI, en ese orden, y disminuye en 16 el SP. Para el 80386 y posteriores, PUSHAD maneja palabras dobles y disminuye el SP en 32. Por lo común, los registros se sacan más tarde de la pila.
*Banderas.* No las afecta.
*Código fuente.* PUSHA/PUSHAD (sin operando)
*Código objeto,* **o** 11 **ooooo**

**PUSHF: Guarda en la pila las banderas**

*Operación.* Guarda en la pila el contenido del registro de banderas para uso posterior. PUSHF disminuye el SP en 2 y transfiere las banderas al nuevo tope de la pila. Para el 80386 y posteriores, PUSHFD maneja palabras dobles y disminuye el SP en 4. (Véase también POPF y PUSH.)
*Banderas.* No las afecta.
*Código fuente.* PUSHF (sin operando)
*Código objeto.* **10011100**

**RCL/RCR: Rota a la izquierda por** y **a la derecha por el acarreo**

*Operación.* Rota los bits a través de la bandera CF. La operación rota los bits a la izquierda o a la derecha en un byte, palabra o palabra doble (80386 y posteriores) en un registro o memoria. El operando puede ser una constante inmediata o una referencia al CL. En el 8088/86, la constante sólo puede ser uno; una rotación mayor debe estar en el CL. En procesadores posteriores, la constante puede ser hasta 31. Para RCL, el bit de más a la izquierda entra a la CF, y el bit de la CF entra al bit 0 del destino; todos los demás bits se rotan a la izquierda. Para RCR, el bit 0 entra a la CF, y el bit de la CF entra al bit de más a la izquierda del destino; todos los demás bits se rotan a la derecha. (Véase también ROL y ROR.)
*Banderas.* Afecta CF y OF.
*Código fuente.* RCL/RCR {registro/memoria}, {CL/inmediato}
*Código objeto.* RCL: **1110100cw|mod010r/m| (si c = 0, el corrimiento es 1;**
RCR: **!H0l00cw!mod011r/m| (si c = 1, el corrimiento está en CL)**

**REP: Repetir cadena de caracteres**

*Operación.* Repite una operación de cadena un número específico de veces. REP es un prefijo de repetición opcional codificado antes de las instrucciones de cadena MOVS y STOS (y antes de INS y de OUTS). Cargue el CX con un contador antes de la ejecución. Para cada ejecución de la instrucción de cadena, REP disminuye en uno CX y repite la operación hasta que el CX es cero. (Véase también REPE/REPZ/REPNE/REPNZ.)
*Banderas.* Véase las instrucciones de cadena asociadas.
*Código fuente.* REP instrucción-de-cadena
*Código objeto.* **11110010**

**REPE/REPZ/REPNE/REPNZ: Repite cadena condicionalmente**

*Operación.* Repite una operación de cadena un número específico de veces o hasta que una condición se cumple. REPE, REPZ, REPNE y REPNZ son prefijos de repetición condicional antes de las instrucciones de cadena SCAS y CMPS. Cargue el CX con un contador antes de la ejecución. Para REPE/REPZ (repite mientras es igual/cero), la operación se repite mientras la ZF es 1 (condición de igual/cero) y el CX no es igual a cero. Para REPNE/REPNZ (repite mientras no es igual/cero), la operación se repite mientras la ZF es 0 (condición de diferente/no-cero) y el CX no es igual a cero. Mientras las condiciones son verdaderas, la operación disminuye en uno el CX y ejecuta las instrucciones de cadena.

*Banderas.* Véase las instrucciones de cadena asociadas.
*Código fuente.* **REPE/REPZ/REPNE/REPNZ instrucción-de-cadena**
*Código objeto,* REPNE/REPNZ: 11110010
REPE/REPZ: 11110011

### RET/RETN/RETF: Regresa de un procedimiento

*Operación.* Regresa de un procedimiento al que se entró previamente por un CALL cercano o lejano. El ensamblador genera un RET cercano si está dentro de un procedimiento etiquetado como NEAR y un RET lejano si está dentro de un procedimiento etiquetado como FAR. Para cercano, RET mueve la palabra en el tope de la pila al IP e incrementa el SP en dos. Para lejano, RET mueve las palabras en el tope de la pila al IP y CS e incrementa el SP en 4. Cualquier operando numérico (un valor de la operación pop codificado como RET 4) es sumado al SP.

RETN y RETF fueron introducidas por MASM 5.0. Puede codificar un regreso cercano o lejano de manera explícita y puede codificar el procedimiento sin las directivas PROC o ENDP. Utilice las etiquetas CALL NEAR/FAR PTR para llamar al procedimiento.
*Banderas.* No las afecta.
*Código fuente.* RET/RETN/RETF [valor del pop]
*Código objeto.* Cuatro formatos:

- Dentro de un segmento: 1 1 0 0 0 0 1 1 1
- Dentro de un segmento con un valor de pop: | n o o o o i o | d a t o - b a j o | d a t o - a l t o |
- Entre segmento: ¡ i i o o i o n |
- Entre segmento con valor de pop: | n o o i o i o | d a t o - b a j o | d a t o - a l t o |

### ROL/ROR: Rota a la izquierda y rota a la derecha

*Operación.* Rota los bits a la izquierda o a la derecha en un byte, palabra o una palabra doble (80386 y posteriores) en un registro o memoria. El operando puede ser una constante inmediata o una referencia al CL. En el 8088/86 la constante sólo puede ser uno; una rotación mayor debe estar en el CL. En procesadores posteriores, la constante puede ser hasta 31. Para ROL, el bit de más a la izquierda entra al bit 0 del destino; todos los demás se rotan a la izquierda. Para ROR, el bit 0 entra al bit de más a la izquierda del destino; todos los demás se rotan a la derecha. (Véase también RCL y RCR.) El bit rotado también entra a la CF.
*Banderas.* Afecta a CF y OF.
*Código fuente.* ROL/ROR {registro/memoria}, {CL/inmediato}
***CÓdigO** objeto.* ROL: | 1 1 0 1 0 0 c w ¡ m o d 0 0 0 r / m | **(si c=0 cuenta=l;**
ROR: | **H O l O O c w | m o d O O l r / m |** **si c=l cuenta está en CL)**

### SAHF: Almacena el contenido de AH en las banderas

*Operación.* Almacena los bits del AH en los bits de más a la derecha del registro de banderas. (Véase también LAHF.)
*Banderas.* Afecta AF, CF, PF, SF y ZF.
*Código fuente.* SAHF (sin operando)
*Código objeto,* **i o o 1 1 1 1 o**

**SAL/SAR: Corrimiento algebraico a la izquierda o corrimiento algebraico a la derecha**

*Operación.* Recorre los bits a la izquierda o a la derecha en un byte, palabra o palabra doble en un registro o en la memoria. El operando puede ser una constante inmediata o una referencia al CL. En el 8088/86, la constante sólo puede ser uno; un corrimiento mayor debe estar en el CL. En procesadores posteriores, la constante puede ser hasta 31.

SAL recorre los bits a la izquierda un número especificado de veces y llena con ceros las posiciones vacantes a la derecha. SAL actúa exactamente igual que SHL. SAR es un corrimiento aritmético que considera el signo del campo referenciado. SAR recorre los bits a la derecha un número especificado y llena con el bit de signo (con 0 o 1) a la izquierda. Todos los bits recorridos fuera del elemento se pierden.

*Banderas.* Afecta CF, OF, PF, SF y ZF. (AF está indefinida.)

*Código fuente.* SAL/SAR {registro/memoria}, {CL/inmediato}

*CÓdigO Objeto.* SAL: |110100cw|modl00r/m| (si c = 0 cuenta = 1;
SAR: |HOl00cw|modlllr/m| si c = 1 cuenta está en CL)

**SBB: Resta con préstamo**

*Operación.* Por lo regular utilizada en resta binaria de múltiples palabras para acarrear el bit uno de desbordamiento al siguiente paso de la aritmética. SBB resta primero el contenido de la CF del primer operando y después el segundo operando del primero, al igual que SUB. (Véase también ADC.)

*Banderas.* Afecta AF, CF, OF, PF, SF y ZF.

*Código fuente.* SBB {registro/memoria}, {registro/memoria/inmediato}

*Código objeto.* Tres formatos:

- Reg/mem con registro: | **ooonodw** | modregr/m |
- Inmed del acumulador: |**oooniow**|**--datos--1datos si w = i**|
- Inmed de reg/mem: |iooooosw|modoiir/m|--datos--|zdatos si swsoizl

**SCAS/SCASB/SCASW/SCASD: Busca en cadena de caracteres**

*Operación.* Busca una cadena de caracteres en memoria por un valor especificado. Para SCASB cargue el valor en el AL, para SCASW cargúelo en el AX y para SCASD cargúelo en el EAX. La pareja ES:DI hace referencia a la cadena en memoria que será buscada. Las operaciones por lo común son usadas con un prefijo REPE/REPNE, junto con un contador en el CX. Si la bandera DF es cero, la operación busca en memoria de izquierda a derecha e incrementa el DI. Si la bandera DF es uno, la operación busca en memoria de derecha a izquierda y disminuye el DI. REPn disminuye el CX en cada repetición. La operación termina en una condición igual (REPNE) o en una no igual (REPE) o cuando el CX es disminuido a cero. La *última* comparación pone en cero o en uno las banderas, no el contenido del CX.

*Banderas.* Afecta AF, CF, OF, PF, SF y ZF.

*Código fuente.* [REPnn]SCASB/SCASW/SCASD (sin operando)

*Código objeto.* **íoioniw**

**SETnn: Establece un byte condicionalmente (80386 y posteriores)**

*Operación.* Establece un byte especificado con base en una condición. Este es un grupo de 30 instrucciones, incluyendo SET(N)E, SET(N)L, SET(N)C y SET(N)S, que de forma paralela hace salto condicional.

Si una condición probada es verdadera, la operación establece el byte operando a uno, de otra forma lo pone en cero. Un ejemplo es:

```
CMP   AX,BX  /Compara el contenido del Ax con el de BX
SETE  CL     /si es igual, pone el CL en 1, en caso contrario en 0
```

*Banderas.* No las afecta.
*Código fuente.* SETnn {registro/memoria}
*CÓdigO Objeto.* \ OOOOllll 1 1001cond|mod000r/m|

(cond varía de acuerdo a la condición que se prueba)

SHL/SHR: **Corrimiento lógico a la izquierda o a la derecha**

*Operación.* Recorre los bits a la izquierda o a la derecha en un byte, palabra o palabra doble en un registro o en la memoria. El operando puede ser una constante inmediata o una referencia al CL. En el 8088/86, la constante sólo puede ser uno; un corrimiento mayor debe estar en el CL. En procesadores posteriores, la constante puede ser hasta 31. SHL y SHR son corrimientos lógicos que tratan el bit de signo como un bit de datos.

SHL recorre los bits a la izquierda un número especificado de veces y llena con ceros las posiciones vacantes a la derecha. SHL actúa exactamente igual que SAL. SHR recorre los bits a la derecha un número especificado y llena con ceros a la izquierda. Todos los bits recorridos fuera del elemento se pierden.

*Banderas.* Afecta CF, OF, PF, SF y ZF. (AF está indefinida.)
*Código fuente.* SHL/SHR {registro/memoria}, {CL/inmediato}
*CÓdigO Objeto.* SHL: 1110100cw|modl00r/m| (si c = 0 cuenta = 1/
SHR: | HOlOOcw | modlOlr/m | si c = 1 cuenta está en CL)

SHLD/SHRD: **Corrimiento en doble precisión (80386 y posteriores)**

*Operación.* Recorre múltiples bits en un operando. Las instrucciones requieren tres operandos. El primer operando es un registro de 16 o 32 bits o una localidad de memoria con el valor que será corrido. El segundo es un registro (del mismo tamaño que el primer operando) con los bits que serán corridos en el primer operando. El tercer operando es el CL o una constante inmediata con el valor de corrimiento.

*Banderas.* Afecta CF, OF, PF, SF y ZF. (AF está indefinida.)
*Código fuente.* SHLD/SHRD {registro/memoria},registro, {CL/inmediato}
*CÓdigO Objeto.* | OOOOllll | ÍOIOOIOO |modregr/m|

**STC: Pone en uno la bandera de acarreo**

*Operación.* Pone en uno la bandera CF. (Véase CLS para poner en cero CF.)
*Banderas.* Pone en uno CF.
*Código fuente.* STC (sin operando)
*Código objeto,* m i i o o i

**STD: Pone en uno la bandera de dirección**

*Operación.* Pone en uno la bandera DF para hacer que las operaciones de cadena de caracteres, como MOVS, procesen de derecha a izquierda. (Véase CLD para poner en cero DF.)

*Banderas.* Pone en uno DF.
*Código fuente.* STD (sin operando)
*Código objeto,* m i i i o i

**STI: Pone en uno la bandera de interrupción**

*Operación.* Pone en uno la bandera IF para permitir interrupciones externas enmascarables después de la ejecución de la siguiente instrucción. (Véase CLI para poner en cero IF.)
*Banderas.* Pone en uno IF.
*Código fuente.* STI (sin operando)
*Código objeto.* **11111011**

**STOS/STOSB/STOSW/STOSD: Almacena una cadena de caracteres**

*Operación.* Almacena el contenido del acumulador en memoria. Cuando es usada con un prefijo REP junto con un contador en el CX, la operación duplica una cadena un número especificado de veces; esto es adecuado para acciones como limpiar el área de memoria. Para STOSB cargue el valor en el AL, para STOSW cargue el valor en el AX y para STOSD cargue el valor en el EAX. El par ES:DI hace referencia a la localidad de memoria en donde el valor está almacenado. Si la bandera DF es 0, la operación almacena de izquierda a derecha e incrementa el DI. Si la bandera DF es 1, la operación almacena de derecha a izquierda y disminuye el DI. REP disminuye el CX para cada repetición y termina cuando se convierte en 0.
*Banderas.* No las afecta.
*Código fuente.* [REP] STOSB/STOSW/STOSD (sin operando)
*Código objeto.* **íoioioiw**

**SUB: Resta números binarios**

*Operación.* Resta números binarios en un registro, memoria o inmediato de un registro o resta valores en un registro o inmediato de memoria. Los números pueden ser de un byte, palabra o palabra doble (80386 y posteriores). (Véase también SBB.)
*Banderas.* Afecta AF, CF, OF, PF, SF y ZF.
*Código fuente.* SUB {registro/memoria}, {registro/memoria/inmediato}
*Código objeto.* Tres formatos:

- Reg/mem con registro: |**ooioiodw|modregr/m|**
- Inmed de acumulador: |**ooionow|--datos--|datos si w=i|**
- Inmed de reg/mem: |**iooooosw|modioir/m| --datos--1datos si sw=oi|**

**TEST: Examina bits**

*Operación.* Examina un campo por una configuración específica de bits tal como AND, pero no cambia el operando destino. Ambos operandos son bytes, palabras o palabras dobles (80386 y posteriores) en un registro o memoria; el segundo operando puede ser inmediato. TEST utiliza AND lógico para establecer banderas, que puede probar con JE o JNE.
*Banderas.* Pone en cero CF y OF y afecta PF, SF y ZF. (AF está indefinida.)
*Código fuente.* TEST {registro/memoria}, {registro/memoria/inmediato}
*Código objeto:* Tres formatos:

- Reg/mem y registro: **jiooooiow|modregr/m|**
- Inmed a acumulador: |**íoioioowl--datos--¡datos si w=i|**
- Inmed a reg/mem: |**iinoiiw|modooor/m|--datos--|datos si w=i|**

### WAIT: Pone en estado de espera al procesador

*Operación.* Permite al procesador principal permanecer en estado de espera hasta que ocurra una interrupción externa, a fin de sincronizarlo con un coprocesador. El procesador principal espera hasta que el procesador termina de ejecutar y reasume el procesamiento al recibir una señal en la pata (pin) TEST.
*Banderas.* **No** las afecta.
*Código fuente.* WAIT (sin operando)
*Código objeto.* **10011011**

### XCHG: Intercambiar

*Operación.* Intercambia datos entre dos registros (como XCHG AH,BL) o entre un registro y memoria (como XCHG CX,palabra).
*Banderas.* No las afecta.
*Código fuente.* XCHG {registro/memoria}, {registro/memoria}
*Código objeto.* Dos formatos:

- Reg con acumulador: |**íooioreg|**
- Reg/mem con reg: |**1000011w|mod reg r/m¡**

### XLAT/XLATB: Traducir

*Operación.* Traduce bytes en formatos diferentes, como de ASCII a EBCDIC. Se define una tabla, se carga su dirección en el BX y después se carga el AL con el valor que será traducido. La operación utiliza el AL como un desplazamiento dentro de la tabla, selecciona el byte de la tabla y lo almacena en el AL. (XLATB es un sinónimo para XLAT.)
*Banderas.* No las afecta.
*Código fuente.* XLAT[AL] (AL operando es opcional)
*Código objeto.* **11010111**

### XOR: Disyunción (OR) exclusiva

*Operación.* Realiza una disyunción excluyeme sobre los bits de los dos operandos. Ambos operandos son bytes, palabras o palabras dobles (80386 y posteriores), que el XOR compara bit a bit. Si ambos coinciden, el bit en el primer operando se pone en cero; si los bits comparados son diferentes el bit en el primer operando se pone en uno. (Véase también AND y OR.)
*Banderas.* Afecta CF (0), OF(0), PF, SF y ZF. (AF está indefinido.)
*Código fuente.* XOR {registro/memoria}, {registro/memoria/inmediato}
*Código objeto.* Tres formatos:

- Reg/mem con registro: |**oonoodw|mod reg r/m|**
- Inmed a reg/mem: |**ioooooow|mod no r/m|—dato_____|datos si w=i|**
- Inmed a acumulador: **|oonoiow|—dato_____|datos si w=i|**

## APÉNDICE A

# Conversión entre hexadecimal y decimal

Este apéndice proporciona los pasos para convertir entre formatos hexadecimal y decimal. La primera sección muestra cómo convertir A7B8 hex al decimal 42,936, y la segunda sección muestra cómo convertir 42,936 de regreso a A7B8 hex.

### CONVERSIÓN DE HEXADECIMAL A DECIMAL

Para convertir el número hexadecimal A7B8 a un número decimal, inicie con el dígito hexadecimal de más a la izquierda **(A),** de forma continua, multiplique cada dígito hexadecimal por 16 y acumule los resultados. Puesto que la multiplicación es en decimal, convierta los dígitos hexadecimales desde **A** hasta H a los decimales 10 a 15. Los pasos proceden de la siguiente manera:

| | |
|---|---|
| Primer dígito: A (10) | 10 |
| Multiplicar por 16 | **x** 16 |
| | 160 |
| Sumar el dígito siguiente, 7 | + 7 |
| | 167 |
| Multiplicar por 16 | **x** 16 |
| | 2,672 |
| Sumar el dígito siguiente, B (11) | + 11 |
| | 2,683 |

| | |
|---|---|
| Multiplicar por 16 | x 16 |
| | 42,928 |
| Sumar el dígito siguiente, 8 | +8 |
| Número decimal | 42,936 |

También puede utilizar una tabla de conversión. Para A7B8H, piense en el dígito de más a la derecha (8) como la posición 1, el siguiente a la izquierda (B) como la posición 2, el siguiente (7) como la posición 3 y el de más a la izquierda (A) como la posición 4. Consulte a tabla A-l y localice el valor para cada dígito hexadecimal:

| | |
|---|---|
| Para la posición 1 (8), columna 1 = | 8 |
| Para la posición 2 (B), columna 2 = | 176 |
| Para la posición 3 (7), columna 3 = | 1,792 |
| Para la posición 4 (A), columna 4 = | 40,960 |
| Número decimal | 42,936 |

## CONVERSIÓN DE DECIMAL A HEXADECIMAL

Para convertir el número decimal 42,936 a hexadecimal, primero divida 42,936 entre 16; el residuo se convierte en el dígito hexadecimal de más a la derecha, 6. Ahora divida el cociente, 2,683, entre 16; el residuo, 11 o B, se convierte en el siguiente dígito hexadecimal hacia la izquierda. Desarrolle el número hexadecimal a partir de los residuos de cada paso de la división. Continúe de esta manera hasta que el cociente sea cero. Los pasos son como sigue:

| OPERACIÓN | COCIENTE | RESIDUO | HEXADECIMAL | |
|---|---|---|---|---|
| 42,936/16 | 2,683 | 8 | 8 | (más a la derecha) |
| 2,683/16 | 167 | 11 | B | |
| 167/16 | 10 | 7 | 7 | |
| 10/16 | 0 | 10 | A | (más a la izquierda) |

También puede utilizar la tabla A-l para convertir de decimal a hexadecimal. Para el número decimal 42,936, localice el número que sea igual o el siguiente menor que él. Anote el número hexadecimal equivalente y su posición en la tabla. Reste el número decimal de ese dígito hexadecimal de 42,936, y localice la diferencia en la tabla. E' procedimiento funciona como sigue:

| | DECIMAL | HEX |
|---|---|---|
| Número decimal inicial | 42,936 | |
| Reste el siguiente número más pequeño | -40,960 | A000 |
| Diferencia | 1,976 | |
| Reste él siguiente número más pequeño | -1,792 | 700 |
| Diferencia | 184 | |
| Reste el siguiente número más pequeño | -176 | B0 |
| Diferencia | 8 | 8 |
| Número hexadecimal final | | A7B8 |

| Hex | Dec | Hex | Dec | Hex | Dec | Hex | Dec |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 4,096 | 1 | 256 | 1 | 16 | 1 | 1 |
| 2 | 8,192 | 2 | 512 | 2 | 32 | 2 | 2 |
| 3 | 12,288 | 3 | 768 | 3 | 48 | 3 | 3 |
| 4 | 16,384 | 4 | 1,024 | 4 | 64 | 4 | 4 |
| 5 | 20,480 | 5 | 1,280 | 5 | 80 | 5 | 5 |
| 6 | 24,576 | 6 | 1,536 | 6 | 96 | 6 | 6 |
| 7 | 28,672 | 7 | 1,792 | 7 | 112 | 7 | 7 |
| 8 | 32,768 | 8 | 2,048 | 8 | 128 | 8 | 8 |
| 9 | 36,864 | 9 | 2,304 | 9 | 144 | 9 | 9 |
| A | 40,960 | A | 2,560 | A | 160 | A | 10 |
| B | 45,056 | B | 2,816 | B | 176 | B | 11 |
| C | 49,152 | C | 3,072 | C | 192 | C | 12 |
| D | 53,248 | D | 3,328 | D | 208 | D | 13 |
| E | 57,344 | E | 3,584 | E | 224 | E | 14 |
| F | 61,440 | F | 3.840 | F | 240 | F | 15 |
| | 4 | | 3 | | 2 | | 1 |

APÉNDICE B

# Códigos de caracteres ASCII

El término ASCII significa American Standard Code for Information Interchange, "código estándar estadounidense para intercambio de información". La tabla B-l lista la representación de todos los 256 códigos de caracteres ASCII (OOH a FFH), junto con sus representaciones hexadecimales. Las categorías son:

| | |
|---|---|
| 00-1FH | Códigos de control para la pantalla, impresoras y transmisión de datos, que son utilizados para provocar una acción. |
| 20-7FH | Códigos de caracteres para números, letras, y puntuación. Observe que 20H es el espacio o blanco estándar. |
| 80-FFH | Códigos ASCII ampliados, caracteres de otras escrituras, griegos y símbolos matemáticos y caracteres gráficos para dibujar cajas. |

A continuación están los códigos de control desde OOH hasta 1FH; los que están entre paréntesis no se imprimen:

| HEX | CARÁCTER | HEX | CARÁCTER | HEX | CARÁCTER |
|---|---|---|---|---|---|
| 00 | (Nulo) | 01 | Carita sonriente | 02 | Carita sonriente |
| 03 | Corazón | 04 | Diamante | 05 | Club |
| 06 | Espada | 07 . | (Beep) | 08 | (Retroceso) |
| 09 | (Tabulador) | 0A | (Avance de línea) | 0B | (Tab vertical) |
| OC | (Avanza página) | 0D | (Return) | 0E | (Shift out) |
| 0F | (Shift in) | 10 | (Línea de datos esc) | 11 | (Dev ctl 1) |
| 12 | (Dev ctl 2) | 13 | (Dev ctl 3) | 14 | (Dev ctl 4) |
| 15 | (Reconocimiento neg) | 16 | (Sincr. ociosa) | 17 | (Fin de trans. de bloque ) |

TABLA A-l TABLA DE CONVERSIÓN HEXADECIMAL-DECIMAL.

| Hex | Dec | Hex | Dec | Hex | Dec | Hex | Dec |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 268,435,456 | 1 | 16,777,216 | 1 | 1,048,576 | 1 | 65,536 |
| 2 | 536,870,912 | 2 | 33,554,432 | 2 | 2,097,152 | 2 | 131,072 |
| 3 | 805,306,368 | 3 | 50,331,648 | 3 | 3,145,728 | 3 | 196,608 |
| 4 | 1,073,741,824 | 4 | 67,108,864 | 4 | 4,194,304 | 4 | 262,144 |
| 5 | 1,342,177,280 | 5 | 83,886,080 | 5 | 5,242,880 | 5 | 327,680 |
| 6 | 1,610,612,736 | 6 | 100,663,296 | 6 | 6,291,456 | 6 | 393,216 |
| 7 | 1,879,048,192 | 7 | 117,440,512 | 7 | 7,340,032 | 7 | 458,752 |
| 8 | 2,147,483,648 | 8 | 134,217,728 | 8 | 8,388,608 | 8 | 524,288 |
| 9 | 2,415,919,104 | 9 | 150,994,944 | 9 | 9,437,184 | 9 | 589,824 |
| A | 2,684,354,560 | A | 167,772,160 | A | 10,485.760 | A | 655,360 |
| B | 2,952,790,016 | B | 184,549,376 | B | 11,534,336 | B | 720,896 |
| C | 3,221,225,472 | C | 201,326,592 | C | 12,582,912 | C | 786,432 |
| D | 3,489,660,928 | D | 218,103,808 | D | 13,631,488 | D | 851,968 |
| E | 3,758,096,384 | E | 234,881,024 | E | 14,680,064 | E | 917,504 |
| F | 4,026,531,840 | F | 251,658,240 | F | 15,728,640 | F | 983,040 |
| | 8 | | 7 | | 6 | | 5 |

APÉNDICE C

# Palabras reservadas

El ensamblador reconoce algunas palabras con un significado especial; puede usar estas palabras sólo bajo condiciones prescritas. Las palabras que el ensamblador reserva pueden ser clasificadas en cuatro categorías:

- Nombres de registros, como AX y AH.
- Instrucciones simbólicas, como ADD y MOV.
- Directivas (instrucciones para el ensamblador), como PROC y END.
- Operadores, como DUP y SEG.

Si se utilizan para definir datos, muchas de las palabras reservadas que siguen pueden confundir al ensamblador o causar un error al ensamblar.

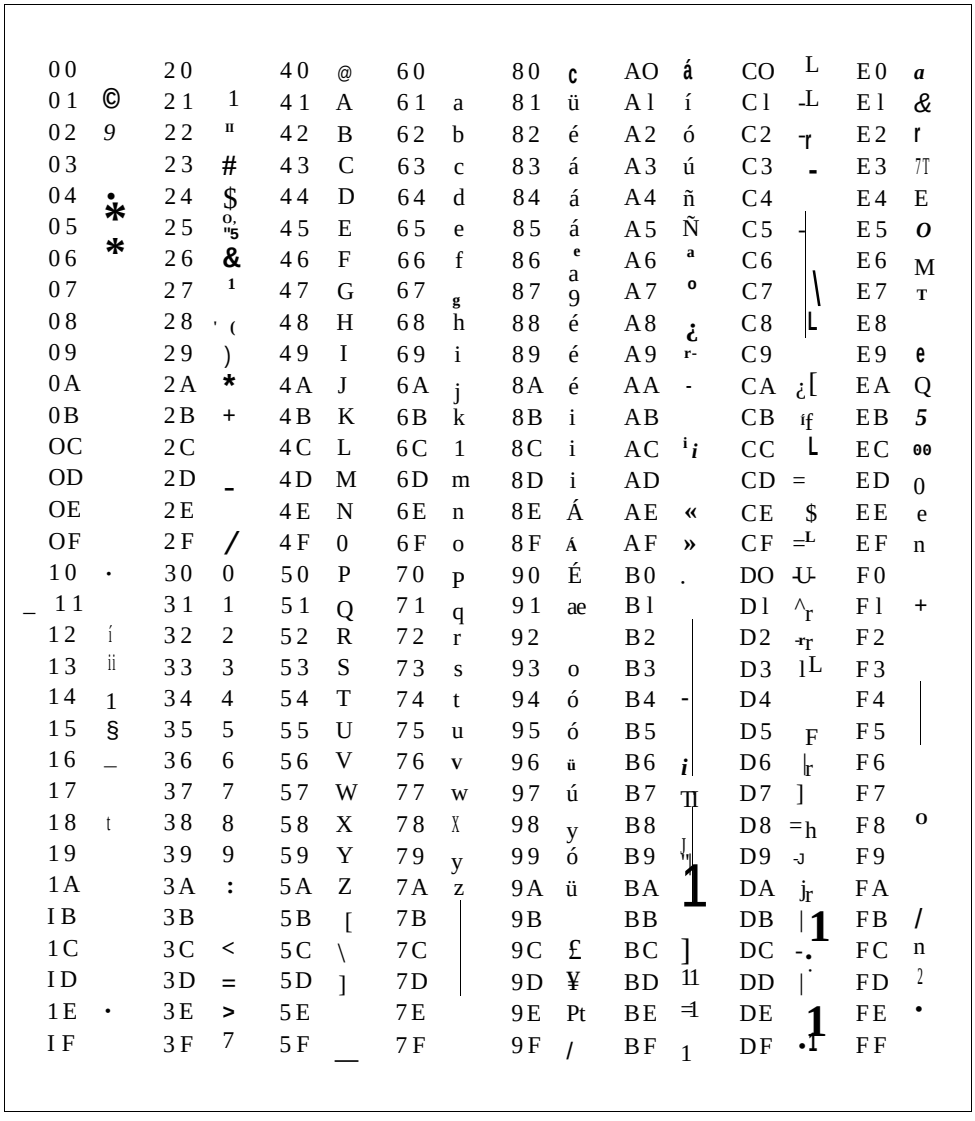
**Nombres de registros**

AH, AL, AX, BH, BL, BP, BX, CH, CL, CS, CX, DH, DI, DI, DL, DS, DX, EAX, EBP, EBX, ECX, EDI, EDX, EIP, ES, ES, ESI, FS, GS, IP, SI, SP, SS

**Instrucciones simbólicas**

AAA, AAD, AAM, AAS; ADC, ADD, AND, ARPL, BOUND, BSF, BSR, BTn, CALL, CBW, CDQ, CLC, CLD, CLI, CLTS, CMC, CMP, CMPSn, CWDn, DAA, DAS, DEC, DIV, ENTER, ESC, HLT, IDIV, IMUL, IN, INC, INSw, INT, INTO, IRET, JA, JAE, JB,

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 18 | (Cancelar) | 19 | (Fin de medio) | 1A | (Sustituir) | | |
| IB | (Escape) | 1C | (Separador de archivo) | ID | (Separador de grupo) | | |
| 1E | (Separador de registro) | IF | (Separador unit) | | | | |



| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | | 20 | | 40 | @ | 60 | | 80 | ç | AO | á | CO | L | E0 | α |
| 01 | © | 21 | 1 | 41 | A | 61 | a | 81 | ü | A l | í | C l | ⊥ | E l | & |
| 02 | 9 | 22 | " | 42 | B | 62 | b | 82 | é | A2 | ó | C2 | ┬ | E2 | Γ |
| 03 | | 23 | # | 43 | C | 63 | c | 83 | á | A3 | ú | C3 | ─ | E3 | π |
| 04 | • | 24 | $ | 44 | D | 64 | d | 84 | á | A4 | ñ | C4 | | E4 | E |
| 05 | * | 25 | % | 45 | E | 65 | e | 85 | á | A5 | Ñ | C5 | ┼ | E5 | σ |
| 06 | * | 26 | & | 46 | F | 66 | f | 86 | ª | A6 | ª | C6 | | E6 | M |
| 07 | | 27 | ' | 47 | G | 67 | g | 87 | a 9 | A7 | º | C7 | \ | E7 | τ |
| 08 | | 28 | ( | 48 | H | 68 | h | 88 | é | A8 | ¿ | C8 | ╚ | E8 | |
| 09 | | 29 | ) | 49 | I | 69 | i | 89 | é | A9 | r- | C9 | | E9 | ê |
| 0A | | 2A | * | 4A | J | 6A | j | 8A | é | AA | - | CA | ¿[ | EA | Q |
| 0B | | 2B | + | 4B | K | 6B | k | 8B | i | AB | | CB | ıf | EB | 5 |
| OC | | 2C | | 4C | L | 6C | 1 | 8C | i | AC | ¡ | CC | ╚ | EC | ∞ |
| OD | | 2D | - | 4D | M | 6D | m | 8D | i | AD | | CD | = | ED | 0 |
| OE | | 2E | | 4E | N | 6E | n | 8E | Á | AE | « | CE | $ | EE | e |
| OF | | 2F | / | 4F | 0 | 6F | o | 8F | Á | AF | » | CF | ═ | EF | n |
| 10 | · | 30 | 0 | 50 | P | 70 | p | 90 | É | B0 | . | DO | ╨ | F0 | |
| 11 | | 31 | 1 | 51 | Q | 71 | q | 91 | æ | B l | | D l | ^r | F l | + |
| 12 | í | 32 | 2 | 52 | R | 72 | r | 92 | | B2 | | D2 | ╥ | F2 | |
| 13 | ii | 33 | 3 | 53 | S | 73 | s | 93 | o | B3 | | D3 | ╙ | F3 | |
| 14 | 1 | 34 | 4 | 54 | T | 74 | t | 94 | ó | B4 | - | D4 | | F4 | |
| 15 | § | 35 | 5 | 55 | U | 75 | u | 95 | ó | B5 | | D5 | F | F5 | |
| 16 | _ | 36 | 6 | 56 | V | 76 | v | 96 | ü | B6 | ¡ | D6 | ╞ | F6 | |
| 17 | | 37 | 7 | 57 | W | 77 | w | 97 | ú | B7 | ╖ | D7 | ] | F7 | |
| 18 | t | 38 | 8 | 58 | X | 78 | x | 98 | y | B8 | | D8 | =h | F8 | o |
| 19 | | 39 | 9 | 59 | Y | 79 | y | 99 | ó | B9 | ╣ | D9 | ╕ | F9 | |
| 1A | | 3A | : | 5A | Z | 7A | z | 9A | ü | BA | 1 | DA | ╓ | FA | |
| IB | | 3B | | 5B | [ | 7B | | 9B | | BB | | DB | ╢1 | FB | / |
| 1C | | 3C | < | 5C | \ | 7C | | 9C | £ | BC | ] | DC | -. | FC | n |
| ID | | 3D | = | 5D | ] | 7D | | 9D | ¥ | BD | 11 | DD | ╎ | FD | 2 |
| 1E | · | 3E | > | 5E | | 7E | | 9E | Pt | BE | ═1 | DE | 1 | FE | • |
| IF | | 3F | 7 | 5F | _ | 7F | | 9F | / | BF | 1 | DF | ·1 | FF | |

TABLA B-l CONJUNTO DE CARACTERES ASCII

APÉNDICE D

# Opciones de ensamblado y de enlace

Este apéndice cubre las reglas para ensamblar, enlazar, generar archivos de referencias cruzadas y convertir programas .EXE a .COM. La versión de ensamblador de Microsoft es MASM, la de Borland es TASM y la de SLR System es OPTASM, todas ella son similares. Desde la versión 6.0 de Microsoft utiliza el comando ML, que realiza un ensamblado y enlace en un comando. Los ejemplos en este apéndice utilizan la unidad D; los usuarios de otras unidades pueden sustituir la letra y ruta apropiadas.

## ENSAMBLANDO UN PROGRAMA

Puede usar una línea de comando para solicitar un ensamblado, aunque MAMS también proporciona indicaciones (prompts).

### Ensamblando con una línea de comando

El formato genera para usar una línea de comando para ensamblar es

```
MASM/TASM [opciones] fuente [, objeto] [, listado] [,refcrz]
```

- Las opciones son explicadas posteriormente.
- Fuente, identifica al programa fuente. El ensamblador supone la extensión .ASM así que no necesita ingresarla. También puede teclear la unidad de disco o la ruta (o ambas).

JBE, JCXZ, JE, JECXZ, JG, JGE, JL, JLE, JMP, JNA, JNAE, JNB, JNBE, JNE, JNG, JNGE, JNL, JNLE, JNO, JNP, JNS, JNZ, JO, JP, JPE, JPO, JS, JZ, LAHF, LAR, LDS, LEA, LEA VE, LES, LFS, LGDT, LGS, LIDT, LLDT, LMSW, LOCK, LODSn, LOOP, LOOPE, LOOPNE, LOOPNZ, LOOPZ, LSL, LSS, LSS, LTR, MOV, MOVSn, MOVSX, MOVZX, MUL, NEG, NOP, NOT, OR, OUTn, POP, POPA, POPAD, POPF, POPFD, PUSH, PUSHAD, PUSHF, PUSHFD, RCL, RCR, REN, REP, REPE, REPNE, REPNZ, REPZ, RET, RETF, ROL, ROR, SAHF, SAL, SAR, SBB, SCASn, SETnn, SGDT, SHL, SHLD, SHR, SHRD, SIDT, SLDT, SMSW, STC, STD, STI, STOSn, STR, SUB, TEST, VERR, VERRW, WAIT, XCHG, XLAT, XOR

**Directivas**

ALIGN, .ALPHA, ASSUME, .CODE, COMM, COMMENT, .CONST, .CREF, .DATA, .DATA?, DB, DD, DF, DOSSEG, DQ, DT, DW, ELSE, END, ENDIF, ENDM, ENDP, ENDS, EQU, .ERRnn, EVEN, EXITM, EXTRN, .FARDATA, .FARDATA?, GROUP, IF, IF1, IF2, IFB, IFDEF, IFDIF, IFE, IFIDN, IFNB, IFNDEF, INCLUDE, INCLUDELIB, IRP, IRPC, LABEL, .LALL, .LFCOND, .LIST, LOCAL, MACRO, .MODEL, ÑAME, ORG, OUT, PAGE, PROC, PUBLIC, PURGE, RADIX, RECORD, REPT, SALL, SEGMENT, .SEQ, .SFCOND, .STACK, STRUC, SUBTTL, .TFCOND, TITLE, .XALL, .XCREF, .XLIST

**Operadores**

AND, BYTE, COMMENT, CON, DUP, EQ, FAR, GE, GT, HIGH, LE, LENGTH, LINE, LOW, LT, MASK, MOD, NE, NEAR, NOT, NOTHING, OFFSET, OR, PTR, SEG, SHL, SHORT, SHR, SIZE, STACK, THIS, TYPE, WHILE, WIDTH, WORD, XOR

### Opciones de ensamblador

Las opciones de ensamblador para MASM, TASM y OPTASM incluyen las siguientes:

| | |
|---|---|
| /A | Acomoda los segmentos fuente en orden alfabético. |
| /C | Crea un archivo de referencias cruzadas. |
| /D | MASM: Produce archivos de listado de la pasada 1 y de la pasada 2 para localizar errores. Para TASM, /Disímbolo significa define un símbolo. |
| /E | Acepta instrucciones del coprocesador 80x87 y genera un enlace a BASIC, C o FORTRAN para emular instrucciones de punto flotante. |
| /H | Muestra opciones de ensamblador con una breve explicación. Ingrese /H (help, ayuda) sin nombre de archivo u otras opciones. |
| /L | Crea un listado de archivo normal. |
| /ML | Hace todos los nombres sensibles a mayúsculas y minúsculas. |
| /MU | Convierte todos los nombres a mayúsculas. |
| /MX | Hace a los nombres públicos y externos sensibles a mayúsculas y minúsculas. |
| /N | Suprime la generación de la tabla de símbolos. |
| /R | Proporciona soporte para coprocesador matemático. |
| /S | Deja los segmentos fuente en la secuencia original. |
| *IT* | (Breve) Muestran diagnóstico al final del ensamblador sólo si se encuentra un error. |
| /V | (En extenso) Al final del ensamblado, muestra el número de líneas y símbolos procesados. (No para OPTASM.) |
| /Wn | Establece el nivel de mensajes de advertencia: 0 = muestra sólo errores críticos; 1 = muestra errores críticos y advertencias graves (por omisión); 2 = muestra errores críticos, advertencias graves y advertencias de consulta. |
| /Z | Muestra líneas fuente en la pantalla para errores. |
| /ZD | Incluye información de números de líneas en archivo objeto para CodeView, TurboDebugger o SYMDEB. |
| /ZI | Incluye información acerca de números de línea e información simbólica en el archivo objeto para CodeView, TurboDebugger o SYMDEB. |

Puede solicitar opciones tanto en modo de petición o de línea de comando. Para peticiones, por ejemplo, podría codificar MASM/A/V[Enter], y después teclee el nombre del archivo de la manera usual. O puede teclear las opciones en cualquier línea de petición; por ejemplo, como

```
Fuente noraarch [.ASM]: /A/V nomarch o noraarch /A/V [Enter]
```

Las opciones /A/V le indican al ensamblador que escriba segmentos en orden alfabético y despliegue diagnóstico adicional al final del ensamblado.

### Características adicionales de Turbo Assembler

Turbo Assembler le permite ensamblar varios archivos, cada uno con sus propias opciones, en una línea de comando. También puede usar los comodines del DOS (* y ?). Para ensamblar todos los programas fuente en el directorio actual, teclee TASM *. Para ensamblar todos los archivos

- Objeto, proporciona un archivo OBJ. La unidad o ruta y el nombre del archivo pueden ser los mismos o diferentes a los de la fuente.
- Listado, provee de un archivo .LST que contiene los códigos fuente y objeto. La unidad o ruta y el nombre del archivo pueden ser los mismos o diferentes a los del fuente.
- Refcrz, provee un archivo generado con los símbolos para un listado de referencias cruzadas. La extensión es .CRF para MASM y .XRF para TASM. La unidad o ruta y el nombre del archivo pueden ser los mismos o diferentes.

Este ejemplo escribe todos los archivos:

```
MASM D:nombre, ASM,D:nombre. OBJ,D:nombre, LST,D:nombre. CRF
```

El siguiente comando abreviado permite, por omisión, los valores para los archivos objeto, de listado y de referencias cruzadas, todos con el mismo nombre:

```
MASM D:nomarchivo, D:,D:,D:,
```

Este ejemplo solicita un archivo de referencias cruzadas pero no un archivo con el listado:

```
MASM D:nomarchivo, D:,,D:
```

**Ensamblado con indicaciones**

También puede teclear los nombres del ensamblado sin línea de comando, aunque TASM y MASM (hasta la versión 5.1) responden de manera diferente. TASM muestra el formato general para la línea de comando y una explicación de las opciones, mientras que MASM muestra una lista de indicaciones a las cuales hay que responder:

```
Source filename [.ASM]:

Object filename [source.OBJ]:

Source listing  [NUL.LST]:

Cross-reference [NUL.CRF]:
```

- Source filename identifica el archivo fuente. Teclee la unidad o ruta (si no es por omisión) y el nombre del archivo fuente, sin la extensión ASM.
- Object filename pregunta por el archivo objeto. La indicación supone el mismo nombre de archivo, aunque se podría cambiar. Para obtener un archivo objeto en la unidad D, teclee D: y presione Enter.
- Source listing proporciona un listado del ensamblador, aunque la indicación supone que no se quiere. Para obtener un listado en la unidad D, teclee D: y presione Enter.
- Cross-reference provee de un listado de referencias cruzadas, aunque la petición supone que no se quiere. Para obtener una en la unidad D, teclee D: y presione Enter.

Para las últimas tres peticiones, sólo presione Enter si quiere aceptar los valores por omisión.

## ARCHIVO DE REFERENCIAS CRUZADAS

Un archivo .CRF o .XRF es utilizado para producir un listado de referencias cruzadas de etiquetas, símbolos y variables de un programa. Sin embargo, tiene que usar CREF para Microsoft o TCREF para Borland a fin de convertir el listado a un archivo de referencias cruzadas ordenado. Puede teclear CREF o TCREF con una línea de comando o utilizar peticiones.

### Uso de una línea de comando

El formato general para utilizar una línea de comando es

```
CREF/TCREF  archxref,archref
```

La línea de comando contiene referencias al archivo original de referencias cruzadas (.CRF o .XRF) y para un archivo generado .REF. El ejemplo siguiente utiliza CREF para escribir un archivo de referencias cruzadas llamado ASMPROG.REF en la unidad D:

```
CREF/TCREF D:ASMPROG,D:
```

### Uso de peticiones

Puede teclear sólo CREF o TCREF sin línea de comando. TCREF sólo despliega el formato general para el comando y una explicación de sus opciones, mientras que CREF muestra estas peticiones:

```
Cref flléname [.CRF]:
List filename [cross-ref.REF]:
```

Para la primera petición, teclee el nombre del archivo sin la extensión .CRF. Para la segunda petición, puede teclear la unidad y/o la ruta y aceptar el nombre del archivo por omisión.

## ENLACE DE UN PROGRAMA

El enlazador de Microsoft es LINK y el de Borland es TLINK. LINK y TLINK aceptan líneas de comando para solicitar un enlace. LINK también proporciona peticiones.

### Enlace con una línea de comandos

El formato general para usar una línea de comando para enlazar es

```
LINK/TLINK [opciones] objfile,exefile[,mapfile][,libraryfile]
```

- *Opciones* se describen más adelante.
- *Objfile* describe el archivo objeto generado por el ensamblador. El enlazador supone la extensión .OBJ, de modo que no tiene que ingresarla. También puede teclear la unidad o la ruta.
- *Exefile* proporciona la generación de un archivo .EXE. El nombre de archivo y la unidad o ruta puede ser la misma o diferente del fuente.

fuente PROG1.ASM, PROG2.ASM, etcétera, teclee TASM PROG?. Puede teclear grupos (o conjuntos) de nombres de archivos, con cada grupo separado por un punto y coma. El siguiente comando ensambla PROG A y PROGB con la opción /C y PROGC con la opción /A:

```
TASM /C PROGA PROGB; /A PROGC
```

Microsoft Versión 6.x

La línea de comando para el ensamblador de Microsoft desde la versión 6.0 es:

```
ML [opciones] nomarchs [[opciones] nomarchs] . . . [/opciones enlace]
```

El ensamblador le permite ensamblar cualquier número de programas en un módulo ejecutable. Una opción útil es ML -?, que despliega la sintaxis completa de la línea de comando y las opciones.

Tablas

Siguiendo a un listado de ensamblador .LST están una tabla de segmentos y grupo y una tabla de símbolos.

Tabla **de** segmento y grupo. Esta tabla tiene el encabezado siguiente:

```
Ñame        Length    Align    Combine    Class
```

La columna *ñame* da los nombres de todos los segmentos y grupos, en orden alfabético. La columna *length* da el tamaño en hexadecimal, de cada segmento. La columna *align* da el tipo de alineación, tal como BYTE, WORD o PARA. *Combine* lista el tipo combinar definido, tal como STACK para una pila, NONE en donde no está codificado tipo, PUBLIC para definiciones externas, o una dirección hexadecimal para tipos AT. La columna *class* lista los nombres de clase de segmento, como están codificados en la instrucción SEGMENT.

Tabla **de** símbolos. Una tabla de símbolos tiene el encabezado siguiente:

```
Ñame        Type      Valué      Attribute
```

La columna *ñame* lista los nombres de todos los elementos definidos, en orden alfabético. La columna *type* da el tipo, como sigue:

- L NEAR o L FAR: Una etiqueta cercana o lejana
- N PROC o F PROC: Un procedimiento cercano o lejano
- BYTE, WORD DWORD, FWORD, QWORD, TBYTE: Un elemento de dato
- ALIAS: Un alias para otro nombre
- NUMBER: Una etiqueta absoluta
- OPCODE: Una equivalencia para un operando de instrucción
- TEXT: Una equivalencia para texto

La columna *valué* da el desplazamiento hexadecimal desde el inicio de un segmento para nombres, etiquetas y procedimientos. La columna *attribute* lista los atributos de símbolos, incluyendo su segmento y longitud.

```
LINK /CO nomarch ...
TLINK nomarch ...
```

### Conversión de archivos objeto de Turbo a programas .COM

TLINK de Borland le permite convertir un programa objeto directamente a formato .COM, dado que el programa fuente fue originalmente codificado de acuerdo a los requisitos de .COM. Utilice la opción /T:

```
TLINK /T archobj,archcom,CON
```

## LISTADO DE REFERENCIAS CRUZADAS

El ensamblador genera un archivo opcional .CRF o .XRF que puede utilizar para producir un listado de referencias cruzadas de etiquetas, símbolos y variables de un programa. El programa que realiza esta función es CREF para Microsoft o TCREF para Borland. Puede teclear CREF o TCREF sin línea de comandos o por medio de indicaciones.

### Uso de una línea de comando

```
CREF/TCREF  d:xreffile,d:reffile
```

- *archxref* identifica al archivo de referencia cruzada generado por el ensamblador. El programa asume la extensión, así que no se necesita indicarla.
- *archref* proporciona lo necesario para generar un archivo .REF. La unidad de disco, subdirectorio y el nombre del archivo pueden ser los mismos o diferentes de los de la fuente.

### Uso de una indicación

Puede teclear TCREF o CREF sin línea de comando, aunque responden de forma diferente, TCREF muestra el formato general para el comando y una explicación de las opciones, mientras que CREF muestra indicaciones. Aquí están las indicaciones de CREF a las cuales debe responder:

```
Cross-reference  [.CRF]:
Listing [filename.REF]:
```

Para la primera indicación, teclee el nombre del archivo .CRF, tal como D:EXASM1. Para la segunda indicación, puede teclear sólo el nombre de la unidad y aceptar el nombre de archivo por omisión. Esta selección hace que CREF escriba un archivo de referencias cruzadas llamado EX ASM 1. REF en la unidad D.

## OPCIONES DE EXE2BEN

El programa EXE2BIN del DOS convierte módulos .EXE generados por MASM en módulos .COM, dado que el programa fuente fue originalmente codificado de acuerdo con los requisitos .COM. Introduzca el comando siguiente:

- *Mapfile* proporciona la generación de un archivo con extensión .MAP que indica la localidac relativa y el tamaño de cada segmento y cualquier error que LINK ha encontrado. Un errot común es el fallo al definir el segmento de la pila. Ingresando CON le indica al ensamblado] que muestre el mapa en la pantalla (en lugar de escribirlo en disco) de modo que pueda vei de forma inmediata los errores.
- *Libraryfile* proporciona la opción de bibliotecas.

Para enlazar más de un archivo objeto en un módulo ejecutable, combínelos en uno como:

```
LINK D:PROGA+D:PROGB+D:PROGC
```

**Enlace por medio de indicaciones**

Puede teclear sólo en nombre del enlazador sin línea de comando, aunque TLINK y LINK responden de forma diferente. TLINK muestra el formato general para el comando y una explicación de las opciones, mientras que LINK muestra una lista de indicaciones. A continuación están la; indicaciones de LINK a las cuales tiene que responder:

```
Object Modules [.OBJ]:

Run File [EXASM1.EXE]:

List File [NUL.MAP]:

Libraries [.LIB]:
```

- *Object Modules* pregunta por el nombre(s) del módulo(s) objeto que serán enlazados; s omite la extensión, por omisión es .OBJ.
- *Run File* solicita el nombre del archivo que será ejecutado y permite por omisión el nombre del archivo objeto. Sólo necesita teclear la unidad y/o la ruta.
- *List File* proporciona el archivo del mapa, aunque por omisión es NUL.MAP (esto es, sir mapa). Al responder CON le indica al enlazador que despliegue el mapa en la pantalla, un* selección adecuada.
- *Libraries* pregunta por la opción de bibliotecas, que está fuera del alcance de este libro.

Para las últimas peticiones, sólo presione Enter para aceptar los valores por omisión. ***E*** ejemplo siguiente le dice al ensamblador que produzca archivos .EXE y .CON:

```
Object Modules [.OBJ]: D:ASMPROG [Enter]

Run File [ASMPROG.EXE]: D: [Enter]

List File [NUL.MAP]: CON [Enter]

Libraries [.LIB]: [Enter]
```

**Opciones para depurar**

Si utiliza CodeView, TurboDebugger o SYMDEB, utilice la opción /ZI de ensamblador para *e* ensamblador. Para enlazar, utilice la opción /CO de DOS LINK, en cualquiera ya sea en línea de comando o en modo de peticiones, o la opción /V de TLINK:

APÉNDICE E

# El programa Debug del DOS

El programa DEBUG en el disco del DOS es útil para escribir programas muy pequeños para depurar programas en lenguaje ensamblador y para examinar el contenido de un archivo o de la memoria. Puede ingresar uno de dos comandos para iniciar DEBUG:

1. Para crear un archivo o examinar memoria, teclee DEBUG sin especificar archivo.
2. Para modificar o depurar un programa (.COM o .EXE) o para modificar un archivo teclee DEBUG con una especificación de archivo, como DEBUG D:PROGC.COM.

El DOS carga a DEBUG en la memoria y DEBUG muestra un guión (-) como una indicación. El área de memoria para su programa es conocida como un *segmento de programa.* Los registros CS,DS,ES y SS son inicializados con la dirección del prefijo de segmento de programa (PSP) y su área de trabajo empieza en PSP + 100H.

Una referencia a una dirección de memoria puede estar en términos de un segmento y un desplazamiento, tal como DS:120, o sólo un desplazamiento, como 120. También puede hacer referencias directas a las direcciones de memoria, como 40:417, en donde 40[0]H es el segmento y 417H es el desplazamiento. DEBUG supone que todos los números ingresados están en hexadecimal, de modo que no tiene que teclear la H al final. Las teclas Fl y F3 funcionan para DEBUG igual que para el DOS; esto es, Fl duplica el comando anterior un carácter a la vez y F3 duplica todo el comando anterior. También DEBUG no distingue entre letras mayúsculas y minúsculas.

A continuación está una descripción de cada comando DEBUG, en orden alfabético.

```
EXE2BIN D:nomarch D:nomarch.COM
```

El primer operando es el nombre del archivo .EXE, que teclea sin extensión. El segundo operando es el nombre del archivo .COM; puede cambiar el nombre, pero asegúrese de codificar la extensión .COM. Borra los archivos .OBJ y .EXE.

```
C 050 L30 300   Compara usando una longitud de 30H
C 050 080 300   Compara usando un intervalo
```

La operación despliega las direcciones y contenido de bytes diferentes.

**D (desplegar** o **sacar).** Despliega el contenido de una parte de la memoria en hexadecimal y ASCII. Por omisión el registro es DS y el formato general es

```
D [dirección] o D[rango]
```

Puede especificar una dirección inicial o una dirección inicial junto con un intervalo. La omisión de un intervalo o longitud hace que se tome 80H. Ejemplos del comando D son:

```
D 200         Despliega 80H bytes empezando en DS:200H
D             Despliega 80H bytes empezando desde el último despliegue
D CS:150      Despliega 80H bytes empezando en CS:150H
D DS:20 L5    Despliega 5 bytes empezando en DS:20H
D 300 32C     Despliega los bytes desde 300H hasta 32CH
```

**E (ingresar).** Ingresa datos o instrucciones de máquina. Por omisión el registro es DS y el formato general es

```
E dirección [lista]
```

La operación permite dos opciones: reemplazar bytes con aquellos de la lista o permitir la edición secuencial de bytes. Ejemplos de la primera opción son los siguientes:

```
E 105 13 3A 21        Ingresa tres bytes empezando en DS:105H
E CS:211 21 2A        Ingresa dos bytes empezando en CS:211H
E 110 'cualquiera'    Ingresa una cadena empezando en DS:110H
```

Para la segunda opción, teclee la dirección en que quiere desplegar:

```
E 12C                 Muestra el contenido de DS:12CH
```

La operación espera a que usted ingrese algo. Ingrese uno o más bytes de valores hexadecimales, separados por un espacio, empezando en DS: 12CH. Las cadenas de caracteres aceptan apóstrofos y comillas.

**F (llenar).** Llena un intervalo de localidades de memoria con valores de una lista. Por omisión el registro es el DS. El formato general es

```
F intervalo lista
```

Estos ejemplos llenan localidades en memoria empezando en DS:214H con bytes repeticiones de 'SAM':

A **(ensambla).** Traduce instrucciones fuente en ensamblador a código de máquina. La operación es en especial útil para escribir pequeños programas en lenguaje ensamblador y para examinar pequeños segmentos de código. Por omisión la dirección inicial para el código es CS:0100H y el formato general para el comando es

```
A [dirección]
```

El ejemplo siguiente crea un programa en lenguaje ensamblador que consiste en cinco instrucciones. Codifique las instrucciones en ensamblador (pero no los comentarios); a la izquierda, DEBUG genera el segmento de código (mostrado aquí como XXXX:) y un desplazamiento iniciando en 0100H:

```
            A (o A 100)  [Enter]   Explicación
xxxx:0100   MOV CX, [10D]  [Enter]  Obtiene contenido en 10D
xxxx:0104   ADD CX,1A [Enter]      Suma un valor inmediato
xxxx:0107   MOV [10D] , CX [Enter]  Almacena CX en 10D
xxxx:010B   JMP 100 [Enter]        Regresa al inicio
xxxx:010D   DW 2500 [Enter]        Define constante
            [Enter]                Fin del comando
```

Como DEBUG pone el IP a 100H a causa del tamaño de PCP las instrucciones empiezan en 100H. La última tecla Enter (esto es, dos en sucesión) indican a DEBUG finalizar el programa. Ahora puede usar el comando U (desensamblar) para ver el código de máquina y el comando T (rastrear) para ejecutarlo.

Puede cambiar cualquiera de las instrucciones anteriores teniendo cuidado que la longitud de la nueva instrucción sea igual a la de la anterior. Por ejemplo, para cambiar el ADD en 104H por SUB, introduzca

```
A 104 [Enter]
xxxx:0104   SUB CX,1A [Enter] [Enter]
```

Cuando reejecuta el programa el IP aún está incrementado. Utilice el comando R para restablecerlo a 100H. Utilice Q para salir.

Observe que puede usar DB y DW para definir datos.

C **(comparar).** Compara el contenido de 2 bloques de memoria. El registro por omisión es el DS y el formato general es

```
C [rango] [dirección]
```

Puede codificar el comando de una de dos formas: (1) una dirección inicial (comparar desde), una longitud y una dirección inicial (comparar a); o (2) una dirección inicial y una dirección final (comparar desde) y una dirección inicial (comparar a). Estos ejemplos comparan bytes que inician el DS:050 con bytes que inician en DS:300:

El ejemplo siguiente carga 15H sectores, empezando en CS: 100 de la unidad 0 (A), iniciando en el sector 20H:

```
L 100 0 20 15
```

La operación L regresa al BX:CX el número de bytes cargados. Para un archivo .EXE, DEBUG ignora el parámetro de la dirección (si existe) y utiliza la dirección que carga en el encabezado .EXE. También quita el encabezado; para conservarlo, primero renombre el archivo con una extensión diferente.

M (mover). Mueve (o copia) el contenido de localidades de memoria. Por omisión el registro es el DS y el formato general es

```
M intervalo dirección
```

Estos ejemplos copian los bytes empezando en DS:050H hasta 150H en la dirección que empieza en DS:400H:

```
M DS:50 L100 DS:400    Utiliza una longitud
M DS:50 150 DS:400     Utiliza un intervalo
```

N (nombrar). Pone nombre a un programa o a un archivo que tiene la intención de leer o escribir en disco. Codifique el comando N espec-arch, como

```
N D:SAM.COM
```

La operación pone el nombre de CS:80 en el PSP. El primer byte de CS:80 contiene la longitud (OAH), seguida por el espacio y la especificación de archivo (espec-arch). Entonces, puede utilizar L (cargar) o W (escribir) para leer o escribir el archivo.

O (Salida). Envía un byte a un puerto. Codifíquelo como O dir-puerto byte.

P (Proceed). Efectúa una llamada a una subrutina (CALL), iteración (LOOP), interrupción (INT) o repite una cadena de instrucción (REP) hasta la siguiente instrucción. El formato general es

```
P [=dirección] [valor]
```

donde =dirección es una dirección de inicio opcional y valor es un número de instrucciones opcional para proceder con ellas. La falta de =dirección ocasiona una omisión al par de registro CS:IP. Por ejemplo, si su trazo de ejecución está en una operación INT 21H, sólo presione la tecla P para ejecutar la operación.

Q (Quit). Salida de DEBUG. La operación no guarda los archivos, utilice W para este propósito.

R (registro). Despliega el contenido de registros y también la siguiente instrucción. El formato general es

```
R [nom-registro]
```

```
F 214 L21 'SAM'   Utiliza una longitud de 21H

F 214 234 'SAM'   Utiliza un rango, 214 hasta 234H
```

G **(Ir).** Ejecuta un programa en lenguaje de máquina que se está examinando hasta un punto de interrupción especificado. Asegúrese de examinar el listado de código de máquina para direcciones válidas IP, ya que una dirección no válida puede provocar resultados impredecibles. También, coloque puntos de interrupción sólo en sus programas, no en el DOS ni en el BIOS. La operación se ejecuta por medio de las interrupciones y se detiene, si es necesario, para esperar por una entrada desde el teclado. Por omisión el registro es el CS. El formato general es

```
G [=dirección] dirección [dirección ...]
```

La entrada =dirección proporciona una dirección opcional de inicio. Las otras entradas proporcionan hasta 10 puntos de interrupción. El ejemplo siguiente le indica a DEBUG que ejecuta hasta la localidad 11A:

```
G 11A
```

**H (hexadecimal).** Muestra la suma y diferencia de dos números hexadecimales, codificado como H valor. La longitud máxima es de cuatro dígitos hexadecimales. Por ejemplo, H 14F 22 despliega el resultado 171 (suma) y 12D (diferencia).

**I (entrada).** Ingresa y despliega un byte desde un puerto. Codifique esto como I dir-de-puerto.

**L (cargar)**. Carga un archivo o sector de disco en memoria. Existen dos formatos generales:

**1.** Carga un archivo con nombre:

```
L [dirección]
```

Utilice el parámetro dirección para hacer que L cargue empezando en una localidad específica. La omisión de la dirección hace que L cargue en CS:100. Para cargar un archivo, observe que ya se le debe haber puesto un nombre (véase N):

```
N espec-arch    Nombre del archivo

L               Carga el archivo en CS:100H
```

Para volver a cargar un archivo, sólo emita L sin dirección.

2. Para cargar datos desde sectores:

```
L [dirección [unidad inicio número]]
```

- Dirección proporciona la localidad de memoria para cargar los datos. (Por omisión es CS: 100.)
- Unidad identifica la unidad de disco, en donde 0 = A, 1 = B, etcétera.
- Inicio especifica el número hexadecimal del primer sector por cargar. (Este es un número relativo, en donde el cilindro 0, pista 0, sector 1, es el sector relativo 0.)
- Número da el número hexadecimal de sectores consecutivos por cargar.

El área especificada debe contener código válido de máquina, que la operación muestra como instrucciones simbólicas. A continuación se presentan tres ejemplos:

```
U 0100        Desensambla 32 bytes empezando en CS:100
U             Desensambla 32 bytes desde el último U, si hubo
U 100 140     Desensambla desde 100H hasta 140H
```

Note que DEBUG no traduce de forma apropiada algunas instrucciones de salto condicional, aunque se ejecutan correctamente.

**W (escribir).** Escribe un archivo desde DEBUG. Primero debe ponérsele nombre al archivo, si no estaba ya cargado. El registro por omisión es el CS y el formato general es

```
W [dirección [unidad sector-inic núm-de-sectores]]
```

Escriba archivos sólo con la extensión .COM, ya que W no da soporte al formato .EXE. (Para modificar un programa .EXE, puede cambiar la extensión de manera temporal.) El ejemplo siguiente utiliza W sin operandos y coloca el tamaño del archivo en la pareja BX:CX (primero asegúrese que el BX es cero):

```
N espec-arch      Nombre del archivo
R CX              Solicita el registro CX
longitud          Inserta el tamaño del archivo
W                 Escribe el archivo
```

Si modifica un archivo y no hace cambios a su longitud o nombre, DEBUG aún puede escribir correctamente el archivo de regreso a su posición original en el disco. También puede escribir el archivo directamente en sectores de disco, aunque esta práctica requiere de mucho cuidado.

Véase el manual del DOS para estos comandos:

- XA: Asigna memoria expandida.
- XD: Desasigna memoria expandida.
- XM: Mapa de páginas lógicas sobre páginas físicas.
- XS: Despliega estado de la memoria expandida.

Los ejemplos siguientes ilustran el uso de este comando:

R Despliega todos los registros
R DX Despliega el DX, DEBUG le da una opción:

**1.** Presione Enter; no ocurre cambio alguno en el DX.
**2.** Ingrese uno a cuatro caracteres hexadecimales para cambiar el contenido del DX.

R IP Despliega el IP. Teclee otro valor para cambiar su contenido.
R F Despliega la configuración actual de cada bandera como un código de dos letras Puede cambiar cualquier número de banderas, en cualquier secuencia:

| BANDERA | EN UNO | EN CERO |
|---|---|---|
| desbordamiento | ov | nv |
| dirección | dn | up |
| signo | ng(-) | pl(+) |
| cero | zr | nz |
| acarreo | cy | nc |

S **(buscar).** Busca en memoria caracteres de una lista. Por omisión el registro es el DS,; el formato general es

```
S intervalo lista
```

Si los caracteres son encontrados, la operación envía sus direcciones; de otra forma no responde El ejemplo siguiente busca la palabra "VIRUS" empezando en DS:300 en los siguientes 20001 bytes:

```
S 300 L 2000 "VIRUS"
```

Este ejemplo busca desde CS: 100 hasta CS:400 por un byte con 51H:

```
S CS:100 400 51
```

**T (rastrear).** Ejecuta un programa en modo de un solo paso. Observe que, por lo común debe utilizar P (proceder) para rastrear a lo largo de las instrucciones INT. El registro por omi sión es la pareja CS:IP y el formato general es

```
T [=dirección] [valor]
```

La entrada opcional = dirección le indica a DEBUG en dónde inicia el rastreo y el valor opciona le indica el número de instrucciones por rastrear. La omisión de los operandos provoca qui DEBUG ejecute la instrucción siguiente y despliegue los registros. A continuación se presenta] dos ejemplos:

```
T-        '   Ejecuta la instrucción siguiente
T 10          Ejecuta las siguientes 16 (10H) instrucciones
```

U **(desensamblar).** Desensambla instrucciones de máquina. El registro por omisión es 1; pareja CS:IP y el formato general es

```
U [dirección]  o  U [rango]
```

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| k y K | 25 | 6B | 25 | 4B | 25 | 0B | 25 | 00 |
| l y L | 26 | 6C | 26 | 4C | 26 | OC | 26 | 00 |
| m y M | 32 | 6D | 32 | 4D | 32 | 0D | 32 | 00 |
| n y N | 31 | 6E | 31 | 4E | 31 | 0E | 31 | 00 |
| o y 0 | 18 | 6F | 18 | 4F | 18 | OF | 18 | 00 |
| p y P | 19 | 70 | 19 | 50 | 19 | 10 | 19 | 00 |
| q y Q | 10 | 71 | 10 | 51 | 10 | 11 | 10 | 00 |
| r y R | 13 | 72 | 13 | 52 | 13 | 12 | 13 | 00 |
| s y S | IF | 73 | IF | 53 | IF | 13 | IF | 00 |
| t y T | 14 | 74 | 14 | 54 | 14 | 14 | 14 | 00 |
| u y U | 16 | 75 | 16 | 55 | 16 | 15 | 16 | 00 |
| v y V | 2F | 76 | 2F | 56 | 2F | 16 | 2F | 00 |
| w y W | 11 | 77 | 11 | 57 | 11 | 17 | 11 | 00 |
| x y X | 2D | 78 | 2D | 58 | 2D | 18 | 2D | 00 |
| y y Y | 15 | 79 | 15 | 59 | 15 | 19 | 15 | 00 |
| z y Z | 2C | 7A | 2C | 5C | 2C | 1A | 2C | 00 |
| Barra espaciadora | 39 | 20 | 39 | 20 | 39 | 20 | 39 | 20 |
| **TECLAS DE FUNCIÓN** | **NORMAL** | | **SHIFT** | | **CTRL** | | **ALT** | |
| Fl | 3B | 00 | 54 | 00 | 5E | 00 | 68 | 00 |
| F2 | 3C | 00 | 55 | 00 | 5F | 00 | 69 | 00 |
| F3 | 3D | 00 | 56 | 00 | 60 | 00 | 6A | 00 |
| F4 | 3E | 00 | 57 | 00 | 61 | 00 | 6B | 00 |
| F5 | 3F | 00 | 58 | 00 | 62 | 00 | 6C | 00 |
| F6 | 40 | 00 | 59 | 00 | 63 | 00 | 6D | 00 |
| F7 | 41 | 00 | 5A | 00 | 64 | 00 | 6E | 00 |
| F8 | 42 | 00 | 5B | 00 | 65 | 00 | 6F | 00 |
| F9 | 43 | 00 | 5C | 00 | 66 | 00 | 70 | 00 |
| FIO | 44 | 00 | 5D | 00 | 67 | 00 | 71 | 00 |
| FU | 85 | 00 | 87 | 00 | 89 | 00 | 8B | 00 |
| F12 | 86 | 00 | 88 | 00 | 8A | 00 | 8C | 00 |
| **TECLADO NUMÉRICO** | **NORMAL** | | **SHIFT** | | **CTRL** | | **ALT** | |
| Ins y 0 | 52 | 00 | 52 | 30 | 92 | 00 | | |
| End y 1 | 4F | 00 | 4F | 31 | 75 | 00 | 00 | 01 |
| Flecha abajo y 2 | 50 | 00 | 50 | 32 | 91 | 00 | 00 | 02 |
| PgDn y 3 | 51 | 00 | 51 | 33 | 76 | 00 | 00 | 03 |
| Flecha arriba y 4 | 4B | 00 | 4B | 34 | 73 | 00 | 00 | 04 |
| 5 (teclado numérico) | 4C | 00 | 4C | 35 | 8F | 00 | 00 | 05 |
| Flecha a la derecha y 6 | 4D | 00 | 4D | 36 | 74 | 00 | 00 | 06 |
| Home y 7 | 47 | 00 | 47 | 37 | 77 | 00 | 00 | 07 |
| Flecha arriba y 8 | 48 | 00 | 48 | 38 | 8D | 00 | 00 | 08 |
| PgUp y 9 | 49 | 00 | 49 | 39 | 84 | 00 | 00 | 09 |
| + (gris) | 4E | 2B | 4E | 2B | 90 | 00 | 4E | 00 |

## APÉNDICE F

# Códigos de rastreo del teclado y códigos ASCII

En las listas siguientes, las teclas están agrupadas en categorías. En cada una, las columnas muestran el formato para una tecla normal (no en combinación con otra tecla) y el formato cuando está combinado con las teclas Shift, Ctrl y Alt. Bajo las columnas encabezadas por "Normal", "Shift", "Ctrl" y "Alt" están dos bytes hexadecimales tal como aparecen cuando la operación del teclado las envía a los registros AH y AL. Por ejemplo, al presionar la letra "a" el normal envía 1EH al AH para el código de rastreo 61H y en el AL para el carácter ASCII. Cuando se oprime junto con la tecla Shift ("A"), la letra envía lEh y 41H respectivamente. Los códigos de rastreo 85H y superiores son para el teclado extendido de 101 teclas.

| LETRAS | NORMAL | | SHIFT | | CTRL | | ALT | |
|---|---|---|---|---|---|---|---|---|
| a y A | 1E | 61 | 1E | 41 | 1E | 01 | 1E | 00 |
| b y B | 30 | 62 | 30 | 42 | 30 | 02 | 30 | 00 |
| c y C | 2E | 63 | 2E | 43 | 2E | 03 | 2E | 00 |
| d y D | 20 | 64 | 20 | 44 | 20 | 04 | 20 | 00 |
| e y E | 12 | 65 | 12 | 45 | 12 | 05 | 12 | 00 |
| f y F | 21 | 66 | 21 | 46 | 21 | 06 | 21 | 00 |
| g y G | 22 | 67 | 22 | 47 | 22 | 07 | 22 | 00 |
| h y H | 23 | 68 | 23 | 48 | 23 | 08 | 23 | 00 |
| i e I | 17 | 69 | 17 | 49 | 17 | 09 | 17 | 00 |
| j y J | 24 | 6A | 24 | 4A | 24 | 0A | 24 | 00 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Flecha a la izquierda | 4B | EO | 4B | EO | 73 | EO | 9B | 00 |
| Flecha a la derecha | 4D | EO | 4D | EO | 74 | EO | 9D | 00 |
| Flecha arriba | 48 | EO | 48 | EO | 8D | EO | 98 | 00 |
| Ins | 52 | EO | 52 | EO | 92 | EO | A2 | 00 |
| Del | 53 | EO | 53 | EO | 93 | EO | A3 | 00 |

Las teclas de control también tienen identificación de códigos de rastreo, aunque el BIOS no las envía al búfer del teclado. A continuación están sus códigos de rastreo:

| | |
|---|---|
| CapsLock | 3A |
| NumLock | 45 |
| ScrollLock | 46 |
| Shift (Izquierdo) | 2A |
| Shift (Derecho) | 36 |
| Alt | 38 |
| Ctrl | ID |
| PrtScreen | 37 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| - (gris) | 4A | 2D | 4A | 2D | 8E | 00 | 4A 00 |
| Del y | 53 | 00 | 53 | 2E | 93 | 00 | |
| * (gris) | 37 | 2A | 37 | 2A | 96 | 00 | 37 00 |
| **FILA SUPERIOR** | **NORMAL** | | **SHIFT** | | **CTRL** | | **ALT** |
| ' y ~ | 29 | 60 | 29 | 7E | | | 29 00 |
| l y ! | 02 | 31 | 02 | 21 | | | 78 00 |
| 2 y @ | 03 | 32 | 03 | 40 | 03 | 00 | 79 00 |
| 3 y # | 04 | 33 | 04 | 23 | | | 7A 00 |
| 4 y $ | 05 | 34 | 05 | 24 | | | 7B 00 |
| 5 y % | 06 | 35 | 06 | 25 | | | 7C 00 |
| 6 y " | 07 | 36 | 07 | 5E | 07 | 1E | 7D 00 |
| 7 y & | 08 | 37 | 08 | 26 | | | 7E 00 |
| 8 y * | 09 | 38 | 09 | 2A | | | 7F 00 |
| 9 y ( | 0A | 39 | OA | 38 | | | 80 00 |
| O y ) | 0B | 30 | OB | 29 | | | 81 00 |
| - y _ | OC | 2D | OC | 5F | OC | IF | 82 00 |
| = y + | 0D | 3D | OD | 2B | | | 83 00 |
| **TECLAS DE OPERACIÓN** | **NORMAL** | | **SHIFT** | | **CTRL** | | **ALT** |
| Esc | 01 | IB | 01 | IB | 01 | IB | 01 00 |
| Retroceso | 0E | 08 | OE | 08 | OE | 7F | OE 00 |
| Tab | OF | 09 | OF | 00 | 94 | 00 | A5 00 |
| Enter | 1C | OD | 1C | OD | 1C | OA | 1C 00 |
| **PUNTUACIÓN** | **NORMAL** | | **SHIFT** | | **CTRL** | | **ALT** |
| [ y { | 1A | 5B | 1A | 7B | 1A | IB | 1A 00 |
| l y } | IB | 5D | IB | 7D | IB | ID | IB 00 |
| ; y : | 27 | 3B | 27 | 3A | | | 27 00 |
| ' y " | 28 | 27 | 28 | 22 | | | 28 00 |
| \ y \| | 2B | 5C | 2B | 7C | 2B | 1C | 2B 00 |
| , y < | 33 | 2C | 33 | 3C | | | 33 00 |
| . y > | 34 | 2E | 34 | 3E | | | 34 00 |
| / y ? | 35 | 2F | 35 | 3F | | | 35 00 |

A continuación están las teclas duplicadas en los teclados ampliados (las primeras dos entradas son caracteres ASCII y el resto son teclas del cursor):

| **TECLA** | **NORMAL** | | **SHIFT** | | **CTRL** | | **ALT** |
|---|---|---|---|---|---|---|---|
| Diagonal | EO | 2F | EO | 2F | 95 | 00 | A4 00 |
| Enter | EO | OD | EO | OD | EO | OA | A6 00 |
| Home | 47 | EO | 47 | EO | 77 | EO | 97 00 |
| End | 4F | EO | 4F | EO | 75 | EO | 9F 00 |
| PageUp | 49 | EO | 49 | EO | 84 | EO | 99 00 |
| PageDown | 51 | EO | 51 | EO | 76 | EO | Al 00 |
| Flecha abajo | 50 | EO | 50 | EO | 91 | EO | AO 00 |

## CAPÍTULO 2

**2-4.** (a) El prefijo del segmento del programa (PSP).

**2-5.** (a) CS = la dirección del segmento de código; IP = la dirección del desplazamiento de la primera instrucción, por lo común cero.

**2-7.** (a) El DOS define la pila para un programa .COM.

**2-8.** (a) Dos bytes (una palabra).

**2-9.** (a) **5A302.**

**2-10.** (a) **5B37A.**

## CAPÍTULO 3

**3-1.** Los comandos están identificados al inicio del capítulo.

**3-2.** (a)DDS:264;(c)EDS:200A8B3 64.

**3-3.** (a)B82946.

**3-4.** ECS: 101 54.

**3-5.** (a)
```
MOV AX, 3 0 04
ADD   AX,3000
NOP
```
(c) Utilice R e IP para restablecer el IP a 100.

**3-6.** El producto es 0612H.

**3-8.** Utilice el comando N para ponerle nombre al programa, coloque el tamaño en el BX:CX y utilice el comando W para escribir el programa.

## CAPITULO 4

**4**-3. Nombre (de un elemento de dato) y etiqueta (de una instrucción).

**4-4.** (d) No válido ya que inicia con un número; (e) válido sólo si se prefiere al registro AX.

**4-6.** (a) TITLE.

**4-8.** (a) Provoca alineación de un segmento en una frontera, tal como la de un párrafo.

**4-9.** (a) Proporciona una sección de código relacionado, como una subrutina.

**4-10.** (a) END; (c) ENDS.

**4-11**. La directiva END le indica al ensamblador que no existen más instrucciones por ensamblar; instrucciones que hagan que el control regrese al sistema operativo son MOV AX,4C00H e INT 21H.

**4-12.** ASSUME SS:STKSEG,DS:DATSEG,CS:CDSEG.

**4-15.** (a)4;(c) 10; (e) 1.

**4-16.** TITLE 1 DB 'RGB Electronics'

**4-17.** (a) FLDA DD 73H
(c) FLDC DW ?
(e)FLDE DW 17, 19, 21, 26, 31

**4-18.** (a) ASCII 3238; (b) hex 1C.

**4-19.** (a) 28; (c) 3A732800.

# Respuestas a preguntas seleccionadas

## CAPÍTULO 1

1-1. (a) 0110; (c) 10110.
1-2. (a) 00100010; (c) 00100000.
1-3. (a) 11101010; (c) 11000100.
1-**4.** (a) 00111000; (c) 00000010.
1-5. (a)51;(c)5D.
1-6. (a) 23C8; (c) 8000.
1-7. (a) 13;(c)59;(e)FFF.
1-8. (a) 01010000; (c) 00100011.
1-10. ROM (memoria de sólo lectura) es permanente, realiza el proceso de arranque y maneja la entrada/salida. RAM (memoria de acceso aleatorio) es temporal y es el área en donde los programas y datos residen cuando se ejecutan.
1-12. (a) Una sección de un programa, hasta 64K en tamaño, con código, datos o la pila.
1-13. (a) Pila, datos y código.
1-15. (a) AX, BX, CX, DX, DI, SI; (c) AX y DX; (e) banderas.
1-17. (a) MOV CH.25.

## CAPÍTULO **8**

**8-1.** (a) En el intervalo -128 a +127 bytes.

**8-2.** (a) En el intervalo -128 a +127 bytes. (b) El operando es un número de un byte que permite desde OOH hasta 7FH (0 hasta +128) y de 80H a FFH (-128 a -1).

**8-3.** (a) 64B; (c) 5EA.

**8-4.** A continuación se presentan algunas soluciones posibles:

```
        MOV AX,00
        MOV BX,01
        MOV CX,12
        MOV DX,0 0
B20 :
        ADD AX,BX     ;E1 número está en el AX
        MOV BX,DX
        MOV DX,AX
        LOOP B20
```

**8-5.** (a) CMP DX.CX
JA dirección

(c) JCXZ dirección
o CMP CX,0
JZ dirección

(e)CMP BX.AX
JLE o JNG

**8-6.** (a) OF (1); (c) ZF (1); (e) DF (1).

**8-8.** El primer PROC (principal) debe ser FAR porque el DOS enlaza a su dirección de ejecución. Un atributo NEAR significa que la dirección está dentro de este segmento particular.

**8-10.** Tres (una para cada CALL).

**8-11.** **(a)** 1001 1010; **(c)** 1111 1011; (e) 0000 0000.

**8-13.** **(a) 5CDCH; (c) CDC8H;** (e) **3737H;** (g) **72B9H.**

## CAPITULO **9**

**9-1.** (a) Renglón = 00 y columna .= 00

**9-3.**
```
MOV  AX,0S0BH    ;Petición
MOV  BH,atributo ; para limpiar
MOV  CX,0C00H    ;  la pantalla
MOV  DX,164H
INT  10H
```

**9-4.**
```
MSSGE  DB  'What is the date (mm/dd/yy)?',07H,'$'
       MOV  AH,09H        ;Request display
       LEA  DX,MSSGE      ; of date
       INT  21H
```

## CAPITULO 5

5-1. MASM/TASM C:DISCOUNT, C:, C:, C:.

5-3. (a) DEBUG C:DISCOUNT.EXE

5-4. (a) Programa fuente en lenguaje ensamblador; (c) archivo del listado del ensamblado con el código fuente y el código objeto; (e) archivo objeto ensamblado.

5-5.
```
MOV   AX, DATSEG
MOV   DS.AX
```

5-6. Codificación parcial:

```
MOV   AL,40H          Carga 40H
SHL   AL,1            Desplazamiento a la izquierda (Doble)
MOV   BL,22H          Multiplica AL
MUL   BL               por 22H
```

5-8. El segmento de datos debe contener estos elementos de datos:

```
FIELDA DB 40H

FIELDB DB 22H
FIELDC DW ?
```

## CAPITULO 6

6-2. (a) El primer MOV mueve el valor inmediato 325AH; el segundo MOV mueve el contenido de las localidades 325AH y 325BH al AX.

6-4. Mueve el contenido al CX de la localidad de memoria apuntada por la suma de la dirección de desplazamiento en el BX, más el SI, más 4 (técnicamente por DS:[BX + SI+4]).

6-5. (a) El procesador no puede mover datos directamente la localidad de una memoria a otra.

6-7. (a) `MOV AX,320`
(c) `ADD BX,40H`
(e) `SHL FLDB, 1 (or SAL)`

6-8. UseXCHNG.

6-9. Utilice LEA.

6-11. (a) Guarda en la pila las banderas, IP y CS, reemplaza las banderas IF y TF y almacena la dirección de la interrupción en el CS:IP.

## CAPÍTULO 7

7-1. 64K.

7-4. Utiliza el área alta del programa .COM o si es espacio no es suficiente ahí, utiliza el final de la memoria.

7-5. (a) EXE2BIN SAMPLE SAMPLE.COM.

**11-8.** A cualquier presión o liberación de una tecla.

**11-10.** (a) Localidad 40:1EH (41EH).

## CAPITULO 12

**12-1.** (a) ES:DIyDS:SI

**12-4.** (a)
```
        JCXZ   label2
label1  MOV    AX,[SI]
        MOV    [DI],AX
        INC    DI
        INC    DI
        INC    SI
        INC    SI
        LOOP   label1
label2: ...
```

**12-5.** Establezca el DF para un movimiento de derecha a izquierda. Para MOVSB, inicialice en NAME1 **+9** y NAME2 + 9. Para MOVSW, inicialice en NAME1 +8 y en NAME2 + 8.

**12-6.**
```
(a) CLD                         Izquierda a derecha
    MOV    CX,20                Inicializa
    LEA    SI,CONANME            para mover
    LEA    DI,PRLINE             20 bytes
    REP    MOVSB                Mueve cadena

(c) CLD
    LEA    SI,CONAME+2          Inicia en el tercer byte
    LODSW                       Carga 2 bytes
(e) CLD                         Izquierda a derecha
    MOV    CX,20                20 bytes
    LEA    SI,CONAME            Inicializa
    LEA    DI,PRLINE             la dirección
    REPE   CMPSB                Compara cadena
```

**12-7.** Aquí está una solución:
```
H10SCAS  PROC NEAR
         CLD                      Izquierda a derecha
         MOV CX,10                10 bytes
         LEA DI,ÑAME1             Inicializa la dirección
         MOV AL,'e'                y el carácter de rastreo
H20:
         REPNE SCASB              Busca
         JNE H30                  ,•¿Encontró?
         CMP BYTE PTR [DI] , 'r'  ,-Sí, ¿el siguiente byte
         JNE H20                  ,- es igual a 'r'?
         MOV AL,03
H30:     RET
H10SCAS  ENDP
```

9-5.
```
DATEPAR   LABEL   BYTE
MAXLEN    DB      9              ;Espacio para diagonales y Enter
ACTLEN    DB
DATEFLD   DB      9 DUP('

          MOV     AH, OAH        ;Petición de para entrada
          LEA     DX,DATEPAR     ; de datos
          INT     21H
```

9-8. (a) 00.

## CAPÍTULO 10

10-1. (a) 0000 0001; (c) 0111 1000.

10-2. (a) 1011 0101; (c) 1000 1100.

10-3. (a)
```
MOV   AH,00H      ,-Petición para fijar el modo
MOV   AL, 02      ,-80 columnas monocromático
INT   10H
```
(c)
```
MOV   AH,060AH    /Petición para recorrer 10 líneas
MOV   BH, 07      /Video normal
MOV   CX,0000     /Pantalla completa
MOV   DX,184FH
INT   10H
```

10-4. Ocho colores para el fondo y 16 para el primer plano (frente).

10-5.
```
MOV   AH,0 9H          |/Petición para desplegar
MOV   AL, 04           |/Diamante
MOV   BH, 00           |/Página número 0
MOV   BL, 01011010B    |/Verde claro sobre magenta
MOV   CX, 05           |/Cinco veces
INT   10H
```

10-11. Primero establezca el modo gráfico; después utilice la INT 10H, función 0BH, para fijar el color del fondo.

10-12. Primero establezca el modo gráfico, después lea el punto así:
```
MOV   AH,0DH          Petición para leer el punto
MOV   CX,13           Columna
MOV   DX,12           Renglón
INT   10H
```

## CAPITULO 11

11-1. (a) Localidad 40:17H (4127H).

11-2. (a) Entrada desde el teclado con repetición en la pantalla. Requiere dos interrupciones si es una función extendida.

11-4. (a) 48H; (c) 47H.

11-6. Utilice la función 00H o la 10H de la INT 16H, para ingresar y probar el código de rastreo y utilice la INT 10H para colocar el cursor.

## CAPITULO 15

**15-2.** TABLEX DW 50 DUP (' ').

**15-3.** (a) ITEMNO DB '06710714721724'
(c) ITPRICE DW 9395,8225,9067,8580,1385

**15-4.** Los procedimientos siguientes constituyen una posible organización:

| SUBRUTINA | PROPÓSITO |
|---|---|
| B10READ | Mostrar la petición, aceptar un número. |
| C10SRCH | Buscar en la tabla, muestra un mensaje si es un elemento no válido. |
| D10MOVE | Sacar la descripción y el precio de la tabla. |
| E10CONV | Convertir la cantidad de ASCII a binario. |
| F10CALC | Calcular el valor (cantidad precio). |
| GIOCONV | Convertir el número de binario a ASCII. |
| K10DISP | Mostrar la descripción y el valor en la pantalla. |

**15-5.** La rutina siguiente copia la tabla. Consulte la figura 15-7 para ordenación de las entradas de la tabla.

```
SORTAB  DB    5 DUP(9 DUP(?))

        LEA   SI,ITDESC          Inicializa
        LEA   DI,SORTAB           la dirección de la tabla y
        MOV   CX,45               el número de caracteres
        CLD                      Izquierda a derecha
        REP   MOVSB              Mueve cadena
```

**15-6.** La intención es utilizar XLAT para traducción

## CAPÍTULO 16

**16-1.** 512.

**16-4.** (a) Un grupo de sectores (1, 2, 4 u 8) que el DOS trata como una unidad de espacio de almacenamiento en disco.

**16-5.** (a) 40 cilindros x 9 sectores X 2 lados X 512 bytes = 368,640.

**16-7.** (a) Lado 0, pista 0, sector 1.

**16-8.** En el directorio, el primer byte del nombre de archivo es puesto a E5H.

**16-11.** (a) Posiciones 28-31 del directorio; (b) 0B4AH, almacenado como 4A0B.

**16-12.** (a) El primer byte (descriptor de medios) contiene F8H.

## CAPITULO 17

**17-1.** (a) 02.

**17-3.** (b)

```
        MOV   AH,3CH             Petición para crear
        MOV   CX,00              Archivo normal
        LEA   DX,PATH1           Cadena ASCIIZ
        INT   21H                Llama al DOS
        JC    error              Si hay error sale
        MOV   CUSTHAN,AX         Guarda el manejador
```

```
12-8.  PATTERN  DB   03H,04H,05H,0B4H
       DISPLAY  DB   80 DUP(' '),'$'
                CLD                  Izquierda a derecha
                MOV  CX,20           20 bytes
                LEA  SI, PATTERN     Inicializa la
                LEA  DI, DISPLAY      dirección
                REP  MOVSW           Mueve el patrón
```

Después utilice la función 09H de la INT 21H, para desplegar la variable DISPLAY.

## CAPITULO 13

13-1. (a) 127 y 255.

```
13-3. (a) MOV AX , DATA Y
          ADD AX, DATAX       ;Suma DATAX
          MOV DATAY, AX       ; a DATAY
```

(b) Véase la figura 13-2 para suma de palabras múltiples.

13-4. STC pone en uno la bandera de acarreo. La suma es 0148H, más 0237H más 1.

```
13-5. (a) MOV  AX, DATAX
          MUL DATAY           ;El producto está en el DX:AX
```

(c) Véase la figura 13-4 para la multiplicación de una palabra doble por una palabra.

```
13-7. (a)  MOV  AX, DATAX
           MOV BL,25          ,-Divide  DATAX
           DIV BL             ; entre 23
```

## CAPITULO 14

14-1. (a) ADD genera 6CH y AAA genera 0102H.

(c) SUB genera 02H y A AS no tiene efecto.

```
14-2.       LEA  SI,UNPAK          ;Inicializa la dirección
            MOV  CX,04             ; y 4 ciclos
      B20:
            OR   [SI],30H          ;Inserta 3 ASCII
            INC  SI                /Incrementa para el byte siguiente
            LOOP  B20              /Repite 4 veces
```

14-3. Utilice como guía la figura 14-2, pero inicialice el CX con 03.

14-4. Utilice como guía la figura 14-3, pero inicialice el CX con 03.

14-5. (a) Convierta el decimal 46,328 a binario:

| | Decimal | Hex |
|---|---|---|
| 8 X 1 = | 8 | 8 |
| 2 X 10 = | 20 | 14 |
| 3 X 100 = | 300 | 12C |
| 6 X 1000 = | 6000 | 1770 |
| 4 X 10000 = | 40000 | 9C40 |
| | | B4F8 |

(b)
```
        LEA  SI,NAMEFLD        ;Inicializa el nombre
        MOV  CX,longitud       ; y la longitud
   B20:
        MOV  AH, 05H           ,-Petición para imprimir
        MOV  DL, [SI]          ,-Caracteres de nombre
        INT  21H               ;
        INC  SI                /Siguiente carácter de nombre
        LOOP B20               /Repetir longitud veces
```

(c) Podría codificar un avance de línea (OAH) al frente de la dirección. La solución es similar a la parte (b).

(e) Emita otro avance de página (OCH).

**20-4.** HEADING DB 13,10,15,'Title',12

**20-5.** (a) En el AH.

**20-7.** El CX no está disponible para realizar ciclos ya que el ciclo que imprime el nombre utiliza el CX. Podría utilizar el BX así:

```
        MOV  BX,05             /Establece 5 ciclos
   C20:

        DEC  BX                /Disminuye el contador del ciclo
        JNZ  C20               /Repite si aún no es cero
```

## CAPÍTULO 21

**21-1.** (a) Unidad de medida para el movimiento del ratón en incrementos de 1/200 de una pulgada.

**21-2.** Todas estas funciones están identificadas cerca del inicio del capítulo.

**21-3.** Observe el efecto de las funciones 01H y 02H sobre la bandera.

**21-6.** Observe la figura que invierte los puertos paralelos LPT1 y LPT2.

## CAPÍTULO 22

**22-1.** La introducción a este capítulo da tres razones.

**22-2.** Los enunciados incluyen MACRO y ENDM.

**22-5.** (a) SALL.

**22-6.** (a)
```
MULTBY   MACRO   MULTPR,MULTCD
         MOV     AL,MULTCD
         MUL     MULTPR
         ENDM
```

**22-7.** Para incluir la macro en la pasada 1, codifique lo siguiente:

```
IFl
        INCLUDE  nombre-de-biblioteca
ENDIF
```

/

(a)
```
MOV   AH,3DH          ;Petición para abrir
MOV   AL, 00          ;Sólo lectura
LEA   DX,PATH1        ;Cadena ASCIIZ
INT   21H             /Llama al DOS
JC    error           ,-Si hay error, sale
MOV   CUSTHAN, AX     ,•Guarda manejador
```

17-5. En donde un programa abre muchos archivos.

17-7. Utilice la figura 17-2 como una guía para la creación de un archivo en disco y la figura 14-5 para la conversión de ASCII a binario.

17-8. Utilice la figura 17-3 como una guía para la lectura del archivo y la figura 14-6 para la conversión de binario a ASCII.

17-10. Véase la figura 17-4 para el uso de la función 42H.

17-11. Todas las funciones que implican la INT 21H; (a) 16H; (c) 15H: (e) 14H.

17-12. (a) 4; (b) 108 (9 sectores X 3 pistas x registros/pista); (c) un acceso por sector o 27 en total.

## CAPÍTULO 18

Todas las preguntas para este capítulo son ejercicios que implican el uso de DEBUG.

## CAPÍTULO 19

19-2. Más apropiadamente como desarrollador de programas de utilerías de disco.

19-3. (a) En el AH.

19-5. Utilice la INT 13H y la función 00H.

19-6. Utilice la INT 13H y la función 01H.

```
MOV   AH,03H          ;Petición de escritura
MOV   AL,03           ;3 sectores
LEA   BX,OUTDSK       ,-Área de salida ' * " "•'
MOV   CH,08           ;Pista 08
MOV   CL,01           ;Sector 01
MOV   DH,00           ¡Cabeza #0
MOV   DL, 01          ,-Unidad B
INT   13H
```

19-9. El byte de estado en el AH contiene 00000011.

## CAPÍTULO 20

20-1. (a) 09.

20-3. (a)
```
MOV AH, 0 5H      ;Petición para imprimir
MOV   DL,0CH      ;Avance de página
INT   21H
```

CAPÍTULO 25

**25-1.** Estos tipos se tratan al inicio del capítulo en la sección sobre interrupciones.
**25-2.** Estas líneas se tratan al inicio del capítulo en la sección sobre interrupciones.
**25-3.** (a) FFFF[0]H.
**25-5.** En 40[0]H.
**25-6.** (a) Estado del equipo; (c) segundo byte del estado del shift.
**25-7.** (a) Las direcciones de COMÍ y COM2 (en secuencia inversa de bytes).
**25-8.** (a) INT OOH.

CAPÍTULO 26

**26-1.** Las interrupciones de la 20H a la 3FH.
**26-2.** (a) 03H; (c) 30H o 3306H.
**26-3.** (a) Salida de impresora; (c) entrada que se envía al búfer del*teclado.

22-8. La definición de la macro podría empezar con

```
BIPRINT MACRO PRTLINE,PRLEN
```

PRTLINE y PRLEN son argumentos mudos para la dirección y longitud, respectivamente, de la línea que será impresa. Véase el capítulo 20 para usar la INT 17H del BIOS para imprimir.

22-9. Observe que no puede utilizar un IF condicional para examinar por una división entre cero. Un IF condicional funciona sólo durante el ensamblado, mientras que la prueba debe ocurrir durante la ejecución del programa. Codifique las instrucciones en ensamblador tal como éstas:

```
CMP   DIVISOR,00        ;¿E1 divisor es cero?
JNZ   (pasa)            ;No, pasa
CALL  (rutina de mensaje de error)
```

## CAPÍTULO 23

23-1. La introducción de este capítulo da razones.

23-2. (a) PARA.

23-3. (a) NONE.

23-4. (a) 'code'

23-6. (a) EXTRN SUBPRO:FAR

23-7. (a) PUBLIC QTY,VALUÉ,PRICE

23-8. Utilice la figura 23-6 como guía.

23-9. Utilice la figura 23-8 como guía. Sin embargo, esta pregunta implica guardar tres variables en la pila. Por lo tanto, el programa llama tiene que accesar [BP+10] para la tercer entrada (PRICE) en la pila. Puede definir su propio estándar para regresar PRICE por la pila. También tenga cuidado con el valor que se saca de la pila en el operando RET.

23-10. Este programa implica el material de los capítulos 9 (E/S de la pantalla), 13 (multiplicación binaria), 14 (conversión entre ASCII y binario) y 23 (enlace a subprogramas). Sea cuidadoso con la pila.

## CAPÍTULO 24

24-1. (a) En el sector 1, pista 0.

24-2. Actúa como una interfaz de bajo nivel a las rutinas del BIOS en ROM.

24-4. A continuación del MSDOS.SY.

24-5. (a) En los primeros 256 bytes de un programa cuando se carga en memoria para su ejecución.

24-6. 5CH: 03 41 4C 46 20 20 20 20 20 44 4F 43
80H: 0A 20 43 3A 41 4C 46 2E 44 4F 43 0D

24-8. (a)2BAl.

24-9. (a) 1A25[0] + 100H (PSP) + 30H = 1A38[0].

24-10. (a)Significa el inicio de un bloque de memoria (no el último).

24-11. (a) INT 09H, en la tabla de servicios de interrupción en 24H.

# índice


%OUT, directiva, 503
.28Ó/.386/.486, directivas, 504, 584
12 bit, entradas FAT de, 291
16 bits, entradas FAT de, 291
80x86, procesador, 7, 8, 12, 13, 95
80x87, coprocesador, 237
@CODE, 79, 586
©CODESIZE, 79, 586
@CPU, 79, 586
©DATASIZE, 79, 586
©FILENAME, 79, 586
©VERSIÓN, 79, 586

A

Abrir un archivo, 294, 297, 303, 318
Acarreo hacia/fuera del bit de signo, 5, 223
Acceso denegado, 150
Actual
  fecha, 32, 42, 483
  número de bloque, 316
  número de registro, 316
Adaptador de pantalla monocromática (MDA), 154
Adición
  con acarreo. *Véase* instrucción ADC
  de datos ASCII, 243
  de datos binarios, 4, 218
AF (bandera). *Véase* Bandera auxiliar
Ajustar lado derecho de pantalla, 212
Ajuste final, 447
ALIGN
  directiva, 102, 494
  tipo de operando, 508
Alineación de un segmento, 76, 508
Almacenar una cadena de caracteres. *Véase* Instrucción STOS
Ambiente de software, 19
Ambiente, 411
Ángulos en las esquinas, 167
Apóstrofo (en cadenas de caracteres), 62
Apuntador
  a una celda, 275
  al búfer del teclado, 471
  área de exclusión, 384
  entradas en la FAT, 289, 293
  registros, 14
Archivo .ASM, 74
Archivo .CRF, 75, 555
Archivo .LST, 74, 550
Archivo .MAP, 81. 82
Archivo .OBJ (objeto), 73, 81, 108, 550, 553
Archivo .REF, 84, 553
Archivo guardado. 288
Archivo oculto, 287
Archivo XRF, 75, 555
Archivo(s), 294
  abrir, 294
  administración, 20
  apuntador a, 298, 311
  atributo, 287, 343, 444
  borrar, 343, 347
  cerrar, 294
  crear, 294
  manejador de, 148, 149, 297, 344
  renombrar, 346
  tabla de asignación de. *Véase* FAT

un programa para ejecución, 19, 22, 73
un programa, 455
un registro de segmento, 530
una cadena. *Véase* Instrucción LODS
Cercana(o)
dirección, 102, 113, 114
llamada, 121, 413
procedimiento, 414
RET, 414
Cero a la extrema derecha para el segmento de dirección, 11
Cerrar archivo, 294, 300, 318
CF (bandera). *Véase* Bandera de acarreo
CGA (adaptador de video), 154, 157, 173, 180
Ciclo (ejemplo), 114
Cilindro, 283
Clasificación de macros, 401
Cociente, 232
Código de regreso, 57
Códigos de rastreo, 185, 187, 188, 189, 190, 194, 195, 564
Columnas en pantalla, 137, 138
Comando DIR, 456
Comando LINK, 81, 553
Comando MASM, 73, 549
Comando ML, 81
Comando MODE, 75
Comando SORT (DOS), 300
Comando TASM, 73, 549
Comando TCREF, 84, 553
Comando TLINK, 81, 108, 553
Comentarios
en un programa, 49, 496
en una macro, 396
Comillas (en cadenas de caracteres), 62
COMMAND.COM, 10, 20, 286, 438, 439
Comparación
de datos en cadenas de caracteres, 200, 206
de datos, 118
instrucción de, 118, 522
Comparación alfanumérica, 208
Compilador, 49
Comunicaciones
en BIOS, 476
en DOS, 482
en puerto, 476
Concatenación (&) en macros, 402
Condicional
directivas de error, 498
directivas, 404
instrucciones de salto, 118
instrucciones de transferencia, 91
Conmutador de puertos de impresión, 390
Constante de palabra larga, 66
Constante numérica, 63, 64
Constante, 61
Contador de Mickey, 378, 383
Contador de posición, 502
Contenido de una localidad de memoria, 10
Control
carácter de, para impresión, 140, 146, 148
de datos, 330
teclas de, 183, 567
Control de dispositivo paralelo, 476
Control de E/S para dispositivos, 329, 485
Controlador (de discos), 283, 284
Conversión
de byte a palabra, 219
de decimal a hexadecimal, 543
de formato .EXE a .COM, 107
de formato ASCII a binario, 250, 253
de formato binario a ASCII, 250
de palabra a palabra doble, 522
Coordenada horizontal, 380
Coordenada vertical, 380
Copias de la FAT, 287
Coprocesador, 80x87, 237
Corchetes. *Véase* Operador de índice
Corrección de una entrada en DEBUG, 37
Corrimiento
contador de, 507
de bits a la derecha, 128
de bits a la izquierda, 129
de bits, 127, 538, 539
para dividir, 236
para multiplicar, 231
valor del, 127
y redondeo de datos, 251
y rotación de palabra doble, 131
Corrimiento a la derecha, 128
Corta(o)
dirección, 113, 114
entero, 238
real, 238
salto, 116
CRC (verificación de redundancia cíclica), 328, 358
Crear
archivo en disco usando un FCB, 316
archivo en disco, 294, 298. 299, 347
un subdirectorio, 338
CREF
comando, 84, 553, 555
directiva, 496
Ctrl+Break
dirección de salida, 440. 481
interrupción, 478
petición, 185
Ctrl + PrtSc, 475
Cursor
establecer, 11, 24, 55, 56, 58, 77, 82
tamaño del, 159

D

Dato(s), 4, 37
bit, 2
bus de, 7, 8
definición de, 61
elemento de, 61
instrucciones de transferencia de, 89
registro del segmento de. *Véase* Registro DS
segmento de, 11, 24, 55, 56, 58, 77, 82
Datos comunes en subprogramas, 423
Datos de números reales temporales, 238
Datos descriptivos, 4
Datos EBCDIC, 271
Datos no inicializados, 61
DEBUG
comando A, 29, 40
comando D, 29
comando E, 29, 32, 33, 38
comando G, 29
comando L, 292
comando N, 29, 43
comando P, 29, 41
comando Q, 29, 32
comando R, 29, 34
comando T, 29, 34
comando U, 29, 41
comando W, 29, 43
despliegue de memoria, 29
programa, 28, 557
DEBUG, comando M (mover), 561
DEBUG, comando A (ensamblar), 29, 558

tabla de manejadores de, 441
tamaño de un, 316
Área de datos de tiempo terminado, 473
Área de datos del puerto paralelo, 470
Área de datos en un disco duro, 473
Área de memoria alta (HMA), 439
Argumento mudo en macros, 396
Aritmética(o)
acarreo, 16, 223
desbordamiento, 223, 235
en registros de 32 bits, 222
información, 4, 217
instrucciones, 88, 217
operador, 488
Arreglo gráfico mejorado. *Véase* EGA
ASCII
adición, 243
ajustar antes de dividir. *Véase* Instrucción AAD
ajustar después de multiplicar.. *Véase* Instrucción AAM
ajustar después de restar. *Véase* Instrucción AAS
ajustar después de sumar. *Véase* Instrucción AAA
archivo, 307, 369
caracteres, 139, 165, 166, 197, 545
conjunto de caracteres, 546
datos, 7, 243, 271, 272
división, 246
formato, 62, 241, 242, 250
multiplicación, 246
resta, 245
Asignación de memoria, 453
Atributo
de archivo, 287, 343, 444
valor (pantalla), 138, 156, 161, 162, 165
Atributo de subrayado, 157
AUTOEXEC.BAT, 21
Aviso de derechos reservados, 31

Bandera auxiliar de acarreo (AF), 16, 117,243
Bandera de acarreo, 16, 117, 119, 128, 129, 130, 223, 539
Bandera de cero (ZF), 16, 101, 117, 119, 120
Bandera de dirección (DF), 16, 117, 202, 539
Bandera de signo, 16, 117, 120
Bandera de trampa (TF), 16, 101, 117
Bandera de verificación de escritura, 335
Bandera InDOS, 466
Base
especificador de, 63
punto, 241, 253
BCD (decimal codificado en binario)
adición, 248
formato, 241, 242, 250
valor, 67
Binario(a)
aritmética, 4, 218
datos, 3, 217
resta, 5, 218
BIOS de bajo nivel, 136
BIOS, 9, 21, 136
área de datos, 20, 30, 353, 469, 470
byte de estado, 353
interrupciones, 469, 475
operaciones en disco, 352
Bit de signo, 5, 128, 218
Bit, 1, 2, 3, 89
de prueba, 520
instrucciones de corrimiento, 89
rastreo (búsqueda), 520
Bit-w (bit indicador del largo del registro), 515
Bits de modo, 516
Bits r/m, 516
Bits reg (bits de registro), 516
Bloque de datos, 316
Bloque de dispositivo, 331
Bloque de memoria superior (UMB), 447, 448
Bloque de parámetros de unidad, *Véase* DBP
Bloqueo/desbloqueo el acceso a un archivo, 486
Borrar
área de entrada, 145
bandera de acarreo. Véase Instrucción CLC
bandera de dirección. *Véase* Instrucción CLD
búfer del teclado, 187
pantalla, 138
Borrar archivo, 343, 347
Botón (ratón)
estado, 380
oprimir, 381
soltar, 382
Búfer
de sector, 360, 361
del teclado, 185, 188, 194, 196
Búfer de sector, 360, 361
Bus, 7, 8
Buscar en una tabla, 266, 269
Buscar un cilindro, 360
BYTE
directiva. *Véase* Directiva DB
PTR, 44, 93
Byte de modo, 515
Byte más significativo, 10
Byte menos significativo, 10
Byte, 1, 2, 65. 141
entre palabra (división), 232
frontera, 412
por byte (multiplicación), 224
Bytes por sector, 286, 292

C

Cabeza de lectura-escritura, 283
Cadena ASCIIZ, 297
Cadena de caracteres
comparación de, 200, 206, 269
datos de, 62, 139. 200
operaciones con, 200
Cadena de caracteres, 62. 64
Cambiar el signo, 237
Cambio
de mayúsculas/minúsculas, 126
del directorio actual, 339
del estado del disco flexible, 361
Campana, 148, 144
Campo, 3
Capacidad del disco, 329
Carácter comodín, 342
Carácter de avance de línea, 140, 146, 148, 365
Carácter de punto y coma (;) para comentarios, 49
Carácter de retorno de carro, 137, 141, 146, 148, 365
Carácter de retroceso (backspace) (\), 140, 142, 148, 373
Carácter Enter, 137, 142, 145
Carácter Esc, 374
Carácter recibido, 477
Caracteres de puntos, 168
Caracteres imprimibles, 365, 373, 375
Cargador de arranque, 21, 470, 478
Cargar
con traslape (superposición), 4
o ejecutar una función de un programa, 454
un módulo, 449
un programa .COM, 448
un programa .EXE, 449

memoria de acceso. *Véase* DMA
operando de memoria, 93
tabla de direccionamiento, 262
Directorios en disco, 285, 287, 292, 339, 346
Disco
archivos en, 294
área de datos, 285, 286
área de sistema, 285
área de transferencia. *Véase* DTA
biblioteca en, 401
capacidad, 284
características, 282
controlador, 284, 358
estado de búsqueda, 472
estado del motor del, 472
estado, 472
parámetros de la unidad de, 359
procesamiento, 296
superficie del, 283
verificación de escritura en, 328
Disco duro. *Véase* Disco
Disco flexible
área de datos de la unidad de, 472
tipo de, 361
Dispositivo auxiliar, 148
Dispositivo, 148
Dividendo, 232
División
ASCII, 246
binaria, 232, 523, 524
con signo. *Véase* Instrucción IDIV
entre cero, 235, 475
por corrimiento. 236
por resta, 235
sin signo. *Véase* Instrucción DIV
División entre dos, 128
Dos puntos (:), para etiquetas, 113
DOS, 19, 21
administración de memoria, 20, 437
bandera de ocupado, 466
bandera de versión, 484
cargador, 57, 78
comando SORT, 300
funciones INT 21H, 136, 481
interrupciones, 480
organización, 20
versión, 41, 292, 439, 483, 484
DPB (bloque de parámetros de datos), 328, 329, 333
DTA (área de transferencia a disco), 312, 317, 318, 442
Duplicar
un manejador de archivo, 344
un patrón, 211
DWORD
directiva. *Véase* DD
especificador de tipo, 412, 487

E/S (entrada/salida), 120
E/S absolutas en disco, 321
ECC (código de corrección de error), 360
EGA (adaptador de video), 154, 157, 173, 178
Ejecutar
instrucciones, 34, 37, 38
un programa .COM, 109
un programa .EXE, 83
un programa, 1, 83
Ejemplos de programas,
aceptar y desplegar nombres, 143
ajuste de datos a la derecha en la pantalla, 213
borrar archivos de forma selectiva, 347
búsqueda en una tabla por medio de CMP, 268
búsqueda en una tabla por medio de CMPSB, 271
cambio de minúsculas a mayúsculas, 127
clasificación (ordenamiento) de una tabla de nombres, 278
cómo pasar parámetros, 428
conversión de ASCII a EBCDIC, 272
datos comunes en subprogramas, 424
definición de datos en dos programas, 426
despliegue gráfico en color, 179
diagnóstico de ensamblador, 85
ejecución de DIR desde dentro de un programa, 457
enlazar C a ensamblador, 435
enlazar Pascal a ensamblador, 430
exhibición de salarios de empleados, 253
exhibición del directorio, 341
exhibición directa en pantalla, 172, 208
exhibición en ASCII y hexadecimal, 274
función del DOS para exhibir caracteres ASCII, 140
generación de sonido, 391
impresión con encabezados de página y overflow de página, 367
instrucción de macros ensamblada simplificada, 395
intermitencia, video inverso y desplazamiento de pantalla, 169
lectura de sectores de disco, 337
lectura directa de un archivo en disco, 312
listar y suprimir la expansión de macros, 398
listas enlazadas (ligadas), 280
llamada a un subprograma y traslape, 459
operaciones de movimiento ampliado. 100
programa residente, 465
segmento de código definido como PUBLIC, 420
selección de un elemento de un menú, 194
tablas de direccionamiento directo, 263
uso de directivas de segmento simplificadas, 422
uso de EXTRN y PUBLIC, 418
uso de IF y de IFNDEF, 407
uso de INT 13H para leer sectores de disco, 357
uso de la biblioteca INCLUDE, 402
uso de la directiva RECORD, 506
uso de la macro IFIDN, 409
uso de LOCAL en una macro, 400
uso de parámetros en una macro, 397
uso de un manejador de archivo para crear un archivo, 300
uso de un manejador de archivo para leer un archivo, 304
uso de una estructura, 512
uso del BIOS para exhibir caracteres ASCII, 165
uso del ratón, 386
Eliminar subdirectorio, 339
Empacado
datos, 241, 242
decimal, 239
formato BCD, 248
Encabezado arena, 444
Encontrar archivos por comparación, 344
Enlace
a un subprograma, 411
con indicadores, 554
con una línea de comando, 55?
de C y ensamblador, 431

DEBUG, comando C (comparar), 558
DEBUG, comando D (desplegar), 29, 30, 559
DEBUG, comando E (introducir), 29, 559
DEBUG, comando F (llenar), 559
DEBUG, comando G (Ir), 29, 560
DEBUG, comando H (hexadecimal), 446, 560
DEBUG, comando I (introducir), 560
DEBUG, comando L (cargar), 292, "560
DEBUG, comando N (nombrar), 29, 561
DEBUG, comando O (salida), 561
DEBUG, comando P (proceder), 29, 561
DEBUG, comando Q (salir), 29, 561
DEBUG, comando R (registro), 29. 561
DEBUG, comando S (buscar), 562
DEBUG, comando T (rastreo), 29, 562
DEBUG, comando U (desensamblar), 29, 562
DEBUG, comando W (escribir), 29, 563
Decimal
  ajustar después de restar, 248
  ajustar después de sumar, 248
  ajustar. 523
  formato, 63, 242
  punto, 258
Decimal codificado en binario. *Véase* BCD
Definir
  byte. *Véase* Directiva DB
  datos, 37
  diez bytes. *Véase* Directiva DT
  palabra cuádruple. *Véase* Directiva DQ
  palabra doble. *Véase* Directiva DD
  palabra larga. *Véase* Directiva DF
  palabra. *Véase* Directiva DW
  tabla, 260
Depurador CODEVIEW, 28
Desbordamiento
  aritmético, 218, 223
  bandera de, 16, 118, 120, 218, 224
  en la división, 235
Desempaquetados
  datos BCD, 245, 246
  datos, 241, 242
Desensamblar, 29, 562
Desplazamiento v, <U
  en un archivo, 311 <* >
  en un segmento, 12, 25, 26, 93, 99, 102, 115
Despliegue de video gráfico de color, 137, 157, 174
Despliegue en pantalla, 137, 148
Determinar el tipo de adaptador de video, 178
Determinar si el medio es removible, 331
DF (bandera). *Véase* Bandera de dirección
DF (definir palabra larga), 61, 64. 66
Diagnóstico (ensamblador), 85
Dirección -,.,
  alineación, 101 !j
  de un segmento, 491
  de una instrucción, 50
  de una localidad de memoria, 10
  del PSP, 485
  desplazamiento, 94 .*-•:«.
Dirección cilindro-pista, 283
Dirección con numeración par. *Véase* Límite de una palabra
Dirección efectiva, 419
Dirección final, 481 • s ",
Direccionamiento
  capacidad, 12 (
  en modo byte, 515
  operando, 92 ;
Direccionamiento indirecto, 94
Directiva %OUT, 503
Directiva .ALPHA, 495
Directiva .CODE, 61, 495 s Ü "
Directiva .CONST, 496 ¡ ; v
Directiva .DATA, 496
Directiva .DOSSEG, 497
Directiva .ENDIF, 401, 404 -
Directiva .ERR, 498
Directiva .EXIT, 61
Directiva .FARDATA, 500 •
Directiva .LALL, 396
Directiva MODEL, 502
Directiva .SALL, 397
Directiva .SEQ, 510
Directiva .STACK, 61, 510
Directiva .STARTUP, 61
Directiva .XALL, 397 •
Directiva .XLIST, 513
Directiva ASSUME, 54, 56, 58, 77, 495
Directiva COMM, 496
Directiva COMMENT, 50, 496
Directiva DB (definir byte), 52, 61, 62, 63, 65
Directiva DD (definir palabra doble), 61, 64, 66
Directiva DQ, 61, 64, 66
Directiva DT, 61, 64, 67
Directiva DW, 61, 63, 66
Directiva ELSE. 404
Directiva END, 55. 78, 497
Directiva ENDP, 54, 121, 497
Directiva ENDS, 53, 497
Directiva EQU, 68, 147, 497
Directiva EVEN, 499
Directiva EXITM, 406
Directiva EXTRN, 415. 416. 417, 499
Directiva FWORD. *Véase* Directiva DF
Directiva GROUP, 500
Directiva IF, 404
Directiva IFE, 405, 408
Directiva INCLUDE, 401, 500
Directiva IRP, 403, 404
Directiva IRPC, 403, 404
Directiva LABEL, 141, 501
Directiva LIST, 501
Directiva LOCAL, 399
Directiva ORG, 107, 502
Directiva PROC, 54, 77. 123, 504
Directiva PURGE, 402
Directiva QWORD. *Véase* Directiva DQ
Directiva RECORD, 505
Directiva REPT, 403
Directiva SEGMENT AT, 171, 197
Directiva SEGMENT. 53, 76, 412, 419, 508
Directiva STRUC, 510
Directiva SUBTTL, 511
Directiva TBYTE. *Véase* Directiva DT
Directiva TEXTEQU, 68, 511
Directiva TITLE. 52, 55, 511
Directiva WORD. *Véase* Directiva DW
Directiva XCREF, 512
Directiva .286/.386/.486. 504
Directivas de repetición, 403
Directivas IF1/IF2, 401, 405
Directivas IFB/IFNB, 405
Directivas IFDEF/IFNDEF, 405
Directivas IFDIF/IFIDN, 405
Directivas para los listados, 52, 494
Directivas simplificadas de segmento, 59, 78, 109, 421
Directivas, 51, 52, 63, 487, 494
Directo
  bloque, 320
  procesamiento, 310, 319
Directo(a)
  despliegue de video, 170
  E/S, de la consola, 186
  entrada, del teclado, 186

Hexadecimal
  aritmética, 6
  formato, 6, 63, 542
  representación, 6
  tabla de conversión decimal a, 544
HMA (área de memoria alta), 439
Hora
  del día en que fue creado el archivo, 288
  del sistema, 475, 478, 483

I

IBMBIO.COM/IBMDOS.COM, 20
Identificación de modelo, 32
Identificador, 50
IF (bandera). *Véase* Bandera de interrupciones
Impresora
  caracteres de control, 365, 373
  estado, 374
  puerto, 375, 389
Independencia de dispositivo, 19
índice
  operador, 25 26, 93, 94, 489
  registro, 15, 127
Información dependiente del país, 488, 486
Inicialización, 20, 470
Inicializar
  el puerto de impresión, 375
  el ratón, 379
  el registro DS, 56, 78
  un programa .COM, 107
  un programa .EXE, 55
  unidad, 360
Inicio del búfer, 195
Inmediato(s)
  datos, 32
  operando, 93, 97
Instalación del manejador de interrupciones para sucesos del ratón, 383
Instrucción (es), 51
  cola de, 8, 114
  conjunto de, 88, 518
  etiqueta de una, 113
  registro de apuntador a, 14
Instrucción AAA, 243, 518
Instrucción AAD, 245, 246, 518
Instrucción AAM, 245, 246, 519
Instrucción AAS, 243, 245, 519
Instrucción ADC, 222, 519
Instrucción ADD, 35, 218, 519
Instrucción AND, 125, 520
Instrucción CALL, 113, 121, 123, 124, 413, 520
Instrucción CBW, 219, 521
Instrucción CDQ, 521
Instrucción CLC, 222, 521
Instrucción CLD, 202, 521
Instrucción CLI, 521
Instrucción CMC, 521
Instrucción CMP, 118, 522
Instrucción CMPS, 200, 206, 522
Instrucción CMPSB, 208, 522
Instrucción CMPSD, 522
Instrucción CMPSW, 208, 522
Instrucción DAA, 248, 523
Instrucción DAS, 248, 523
Instrucción DEC, 99, 119, 523
Instrucción DIV, 232, 233, 523
Instrucción ESC (Escape), 524
Instrucción HLT, 524
Instrucción IDIV, 232, 233, 524
Instrucción IMUL, 224, 226, 230, 524
Instrucción IN, 195, 388, 525
Instrucción INC, 99, 525
Instrucción INT, 41, 101, 136, 525
Instrucción INTO, 526
Instrucción IRET, 474, 526
Instrucción IRETD, 526
Instrucción JC, 120, 131, 527
Instrucción JE, 119, 120, 527
Instrucción JMP, 113, 114, 528
Instrucción JNC, 120, 528
Instrucción JNE, 101, 119, 120, 529
Instrucción JNZ, 119, 120, 529
Instrucción JS, 120, 530
Instrucción JZ, 119, 120, 527
Instrucción LAHF, 530
Instrucción LDS, 530
Instrucción LEA, 99, 100, 530
Instrucción LES, 530
Instrucción LFS, 530
Instrucción LGS, 530
Instrucción LOCK, 531
Instrucción LODS, 200, 204, 531
Instrucción LODSB, 204, 531
Instrucción LODSD, 531
Instrucción LODSW, 205, 531
Instrucción LOOP, 113, 116, 531
Instrucción LOOPD, 531
Instrucción LOOPW, 531
Instrucción LSS, 530
Instrucción MOV, 26, 32, 35, 77, 95, 532
Instrucción MOVS, 200, 202, 211, 532
Instrucción MOVSD, 532
Instrucción MUL. 224, 226, 533
Instrucción NEG, 143, 237, 533
Instrucción NOP, 36, 43, 533
Instrucción NOT, 126, 534
Instrucción OR, 125, 534
Instrucción OUT, 388, 534
Instrucción para mover y llenar, 96
Instrucción POP, 23, 123, 429, 534
Instrucción POPA, 24, 535
Instrucción POPF, 24, 535
Instrucción PUSH, 23, 123, 427, 535
Instrucción PUSHA, 24, 535
Instrucción PUSHF, 24, 536
Instrucción REP, 201, 202, 536
Instrucción RET, 121, 123, 124, 414, 429, 537
Instrucción ROL, 130, 537
Instrucción ROR, 130, 537
Instrucción SAHF, 537
Instrucción SAL, 129, 538
Instrucción SAR, 128, 538
Instrucción SBB, 538
Instrucción SCAS, 200, 209, 538
Instrucción SCASB, 209, 538
Instrucción SCASD, 538
Instrucción SCASW, 210, 538
Instrucción SETnn, 538
Instrucción SHLD, 539
Instrucción SHRD, 539
Instrucción STC, 539
Instrucción STD, 202, 539
Instrucción STI, 540
Instrucción STOS, 200, 205, 211, 540
Instrucción STOSD, 540
Instrucción SUB, 218, 540
Instrucción TEST, 125, 540
Instrucción WAIT, 541
Instrucción XCHG, 98, 541
Instrucción XLAT, 271, 541
Instrucción XLATB, 541
Instrucción XOR, 125, 541
Instrucción, 51
Instrucciones BSF/BSR, 520
Instrucciones BT/BTC/BTR/BTS, 520
Instrucciones CWD/CWDE, 522
Instrucciones de comparación, 89
Instrucciones de transferencia incondicional, 92
Instrucciones JA/JAE, 119, 526
Instrucciones JB/JBE, 119, 527
Instrucciones JCXZ/JECXZ, 527
Instrucciones JG/JGE, 120, 527
Instrucciones JL/JLE, 120, 528
Instrucciones JNA/JNAE, 119, 527
Instrucciones JNB/JNBE, 119, 526

de Pascal y ensamblador, 429
de un programa .COM, 108
de un programa .EXE, 73
de un programa, 49, 81
mapa de, 82, 451
Ensamblado(r)
con indicadores, 550
con una línea de comando, 549
de un programa, 73
diagnóstico, 85
lenguaje, 48
opciones, 74, 551
pasos, 74
programa (.ASM), 44, 48
Ensamblador de dos pasadas, 79
Ensamblador de Microsoft, 552
Entero largo, 238
Entrada *.
área de, 141
dispositivo de, 148
estado de, 330
Entrada/salida (E/S), 20
Entradas de una tabla de ordenamiento, 274
Equipo
determinación del, 476
estado del, 30, 470
Error i;
clase, 335
código de corrección de, 360
código de regreso de, 298
cuando se ensambla, 75
diagnóstico de, desde un ensamblador, 85
Error crítico
dirección de salida de, 440
manejador de, 481
Error de fase entre pasos (pases o pasadas), 80, 86
ES:NOTHING, 55
Escribir
datos de control a una unidad, 330
directamente en un registro, 319
directamente un bloque, 320
en el búfer de sector, 361
en el búfer extendido de sector, 360
en sectores, 356
en teletipo, 164
en un pixel de punto, 176
en un registro, 299, 317
un archivo en disco, 298
un sector de disco, 331
Espacio en disco, 329
Especificación de archivo, 342
Especificadores de tipo, 487 V. O
Establecer •?
área de exclusión del apuntador, 384
atributo de archivo, 343
bandera de dirección. *Véase* Instrucción STD
cursor, 138, 159
dirección de interrupción, 464
direcciones del PSP, 485
dispositivo de información, 330
enlace con memoria superior, 448
error extendido, 486
estrategia de asignación de memoria, 448
fecha y hora de archivo, 346
fecha, 483
hora, 478, 483 -·¡¿i
identificación de medios, 332
límites horizontales para el apuntador, 382
límites verticales para el apuntador, 383
modo de video, 155, 159, 175
modo gráfico, 175, 178
paletas de colores, 176 !
registros de las paletas, 177
sensibilidad del ratón, 384
tamaño del cursor, 159
tipo de disco flexible, 361
tipo de recurso, 362
ubicación del apuntador (para el ratón), 380
velocidad doble de umbral, 384
Establecer/restablecer verificación de escritura en disco, 328
Estacionar las cabezas del disco, 362
Estado
byte de, 353, 354
de la impresora, 375
de las banderas, 16
del puerto de comunicaciones, 477
Estado Ctrl+C, 484
Estado de detención, 524
Estado de terminación, 298
Estructura (esqueleto) de un programa .EXE, 56
Estructura de tubería, 9
Etiqueta de volumen, 288 "¡rt
Etiqueta, 113
Exhibición .• •'-»«•
de atributo o carácter, 162
de cadena de caracteres, 139, 165
de carácter gráfico, 176
de carácter, 158, 163
de caracteres ASCII, 140, 272
del apuntador del ratón, 380
del contenido de la memoria, 36
del segmento de datos, 67
Expresión (en un operando), 61
Extendido (as)
búfer de sector, 360
búfer, 360
caracteres ASCII, 166
código de error, 335
memoria, 477
operación de movimiento, 99
teclas de función, 183, 187, 189
Extensión en el nombre de un archivo, 287, 315

FAT (tabla de asignación de archivos), 285, 288, 292
FAT1/FAT2, 288
FCB #1, 441
FCB (bloque de control de archivo), 296, 312, 315, 341
FCB estándar no abierto, 441
Fecha
del directorio/archivo, 288, 346
en BIOS, 32, 42, 483
obtener/establecer, 483
Filaxolumna de inicio, 161, 162
Fin de
archivo, 304, 310, 319
ejecución de un programa, 57, 109, 481
entrada de datos, 145
línea de rastreo, 159, 160
página, 366
renglónxolumna, 161, 162
Fondo (pantalla), 157
Formateo de pistas, 332, 359
Formato de datos de real largo, 238
Formato en base 2, 217
Forzar la terminación de búsqueda en una tabla, 269
Frecuencia de una nota, 390
Función
código de una, 57
del DOS, 136, 481
teclas de, 183

G

Generación de sonidos, 390
Generador de caracteres, 164
Guardar un programa en DEBUG, 43

**03H entrada de comunicaciones, 482**
**04H salida de comunicaciones, 482**
**05H salida a la impresora, 373**
**06H teclado y despliegue directo, 186**
**07H entrada directa del teclado sin eco, 186**
**08H entrada del teclado sin eco, 186**
**09H despliegue de una cadena de caracteres, 139**
**OAH entrada del teclado mediante búfer, 141, 186**
**OBH verificación del estado del teclado, 186**
**OCH limpiar el búfer y solicitar una entrada, 187**
**ODH reinicializa la unidad de disco, 326**
**OEH selecciona la unidad de disco por omisión, 326**
**OFH abrir un FCB de un archivo, 318**
**10H cerrar un FCB de un archivo, 318**
**11H buscar la primer entrada que coincida en un disco, 482**
**12H buscar la siguiente entrada que coincida en un disco, 482**
**13H borrar un FCB de un archivo, 482**
**14H leer registro secuencial de un FCB. 319**
**15H escribir registro secuencial de un FCB, 317**
**16H crear un archivo FCB, 317**
**19H determinar la unidad de disco por omisión, 327**
**1BH obtener información de la unidad de disco por omisión, 327**
**1CH obtener información de una unidad específica, 327**
**1FH obtener bloque de parámetros de la unidad por omisión, 328**
**21H leer directamente registro del FCB, 319**
**22H escribir directamente registro del FCB, 319**
**23H obtener el tamaño de un archivo FCB, 482**
**24H establecer el campo de registro aleatorio de FCB, 482**
**25H establecer la dirección de la tabla de interrupciones, 464, 482**
**26H crear nuevo PSP, 483**
**27H leer directamente un bloque de disco, 320**
**28H escribir directamente un bloque de disco, 320**
**29H Análisis sintáctico del nombre de archivo, 341**
**2AH obtener fecha del sistema, 42, 483**
**2BH establecer fecha del sistema, 483**
**2CH obtener hora del sistema, 483**
**2DH establecer hora del sistema, 483**
**2EH establecer/restaurar verificación de disco, 328**
**30H obtener número de versión del DOS, 483**
**31H termina pero permanece residente, 462**
**32H obtener bloque de parámetros de la unidad, 329**
**3300H obtener/verificar el estado de Ctrl + C, 484**
**3305H obtener unidad de arranque, 484**
**3306H obtener número de versión del DOS, 439, 484**
**34H obtener bandera del DOS ocupado, 466**
**35H obtener tabla de dirección de interrupción, 463**
**36H obtener cantidad de espacio libre en disco, 329**
**38H obtener/establecer información dependiente del país, 484**
**39H crear subdirectorio, 338**
**3AH eliminar subdirectorio, 339**
**3BH cambiar de directorio actual, 339**
**3CH crear archivo con un manejador, 299**
**3DH abrir archivo con un manejador, 303**
**3EH cerrar archivo con un manejador, 300**
**3FH leer archivo/dispositivo, 149, 304**
**40H escribir archivo/dispositivo con un manejador, 148, 299, 365**
**41H borrar archivo de un directorio, 343**
**42H mover el apuntador de archivo, 311**
**43H verificar/cambiar atributo de archivo, 343**
**4400H obtener información de dispositivo, 329**
**4401H establecer información de dispositivo, 330**
**4404H leer datos de control de la unidad, 330**
**4405H escribir datos de control en la unidad, 330**
**4406H verificar estado de la entrada, 330**
**4407H verificar estado de la salida, 330**
**4408H determinar si el medio es removible, 331**
**440DH código secundario 41H, escribir en sector de disco, 331**
**440DH código secundario 42H, formatear pistas, 332**
**440DH código secundario 46H, establecer identificación de medio, 332**
**440DH código secundario 60H, obtener parámetros de dispositivo, 333**
**440DH código secundario 61H, leer sector de disco, 334**
**440DH código secundario 66H, obtener identificación de medio, 334**
**440DH código secundario 68H, obtener tipo de medio, 334**
**44H control de E/S para dispositivos, 329, 485**
**45H duplicar un manejador de archivo, 344**
**46H forzar la duplicación de un manejador, 344**
**47H obtener el directorio actual, 339**
**48H asignar un bloque de memoria, 453**
**49H liberar un bloque de memoria asignada, 454**
**4AH establecer el tamaño del bloque de asignación de memoria, 454**
**4BH cargar/ejecutar un programa, 454**
**4CH terminar un programa, 57, 109**
**4DH recupera el código de regreso de un subproceso, 456**
**4EH encuentra la primera entrada de un directorio que cumpla un criterio, 344**

Instrucciones JNG/JNGE, 120, 528
Instrucciones JNL/JNLE, 120, 528
Instrucciones JNO/JNP/JNS, 120, 529
Instrucciones JO/JPO, 120, 529
Instrucciones JP/JPE, 120, 530
Instrucciones LOOPE/LOOPZ, 116, 531
Instrucciones LOOPNE/LOOPNZ, 116, 532
Instrucciones MOVSB/MOVSW, 203, 532
Instrucciones MOVSX/MOVZX. 96, 533
Instrucciones RCL/RCR, 130, 536
Instrucciones REPE/REPZ, 202, 536
Instrucciones REPNE/REPNZ, 202, 536
Instrucciones RETF/RETN, 537
Instrucciones STOSB/STOSW, 205, 540
INT 00H División entre cero, 475
INT 01H Un solo paso, 475
INT 02H Interrupción no enmascarable, 475
INT 03H Punto de interrupción, 475
INT 04H Desbordamiento, 475
INT 05H Impresión de pantalla, 475
INT 08H Cronómetro, 475
INT 09H Interrupción desde el teclado, 195, 466, 475
INT 0BH Control del puerto COMÍ, 475
INT 0CH Control del puerto COM2, 475
INT 0DH Control de dispositivos en paralelo, 476
INT 0EH Control del disco flexible, 476
INT 0FH Control de dispositivos en paralelo, 476
INT 10H Funciones de exhibición de video
00H establece modo del video, 155, 159, 173, 175
01H establece tamaño del cursor, 159
02H establece posición del cursor, 138, 159
03H lee la posición del cursor, 160
04H lee la posición de la pluma óptica, 175
05H selecciona la página activa, 160
06H avanza una pantalla, 161
07H retrocede una pantalla, 162
08H lee el atributo/carácter, 162, 175
09H despliega atributo/carácter, 158, 162, 168, 176
0AH exhibe carácter, 163
0BH establece la paleta de colores, 176
0CH escribe un punto en un pixel, 176
0DH lee un punto de un pixel, 177
0EH escribe en un teletipo, 164
0FH obtiene el modo actual de video, 164
10H establece el registro de la paleta, 177
11H generador de caracteres, 164
12H selecciona rutina alterna de la pantalla, 164
13H despliega una cadena de caracteres, 165
1AH código de combinación de *l despliegue de lectura/escritura, 177
1BH regresa información de funcionalidad/estado, 177
descripción/listado, 137, 476
INT 11H Determinación del equipo, 239, 470, 476
INT 12H Determinación del tamaño de la memoria, 42, 470, 476
INT 13H Funciones del BIOS de E/S en disco,
00H reinicialización del disco de sistema, 354
01H estado de lectura de disco, 354
02H lectura de sectores, 354
03H escritura de sectores, 356
04H verificación de sectores, 358
05H formateo de pistas, 359
08H obtención de parámetros de la unidad, 359
09H inicializar unidad, 360
0AH lectura del búfer extendido de sector, 360
0BH escritura en el búfer extendido de sector, 360
0CH búsqueda de cilindro, 360
0DH reinicialización alterna del disco, 360
0EH lectura del búfer de sector, 361
OFH escritura en el búfer de sector, 361
10H prueba si el disco está preparado, 361
11H recalibrar la unidad de disco duro, 361
12H diagnóstico de la ROM, 361
13H diagnóstico de la unidad, 361
14H diagnóstico del controlador, 361
15H obtención del tipo de disco, 361
16H cambio del estado del disco flexible, 361
17H establecer tipo de disco flexible, 361
18H establecer tipo de medio para formatear, 362
19H estacionar las cabezas del disco, 362
códigos de estado, 353
descripción/listado, 354, 476
INT 14H Comunicaciones de E/S, 476
INT 15H Servicios del sistema, 477
INT 16H Funciones de entrada del teclado
00H lectura de un carácter, 187
01H determina si está presente un carácter, 188
02H regresa el estado actual del corrimiento, 188
05H escritura en el teclado, 188
10H lectura de un carácter, 188
11H determina si está presente un carácter, 189
12H regresa el estado actual del corrimiento, 189
descripción/listado, 187, 477
INT 17H funciones de impresión de BIOS
00H impresión de un carácter, 375, 389
01H inicializar puerto de la impresora, 375
02H obtener estado del puerto de la impresora, 375
INT 18H Entrada con el BASIC de ROM, 478
INT 19H Cargador del arranque, 478
INT 1AH Leer y establecer la hora, 478
INT 1BH Obtener el control con una interrupción del teclado, 478
INT 20H Terminar un programa, 481
INT 21H Funciones del DOS
00H programa terminado, 482
01H entrada del teclado con eco, 185
02H exhibición de un carácter, 147

Línea NMI, 474
Lista de enlace, 275
Llamada
  a un procedimiento, 121
  a un segmento, 413, 414
Llamada en un segmento. 413
Llamada entre segmentos, 414

M

Macro(s)
  biblioteca de, 401
  comentarios en una, 396
  cómo escribir una. 393
  definición de una, 394
  expansión de una, 394
  instrucciones, 393
Manejador de eventos (ratón), 383
Manejador no válido de archivo, 148, 150
Mapa de memoria, 9, 21, 438
Máquina
  código de, 24, 33, 37, 515
  lenguaje de, ejemplo, 32
  lenguaje de, instrucción, 49
Marcador de fin de archivo (1A hex), 310
MCGA (adaptador de video), 154, 157
Medios
  bloque de, 154, 157, 180
  byte descriptor de, 334, 335
  identificación de, 332
  tipo de, 362
Medios removibles, 331
Megabyte, 3
Mejor ajuste, 447
Memoria de sólo lectura. *Véase* ROM
Memoria. 1, 9. 10, 25
  administración de, 20, 437
  bloques de, 444
  determinación del tamaño, 31, 476
  estrategia de asignación de, 447
  modelo, 59, 502
  referencias, 26
  registro de control de. 444
  tamaño de, del área de datos, 471
Menú, 191
Mickey, 378
Microsoft C, 432
Modelo COMPACT, 59, 502
Modelo de memoria LARGE, 59, 502
Modelo HUGE, 502
Modelo MÉDIUM, 59, 502
Modelo SMALL, 59, 502
Modelo TINY, 59, 502
Modificación del bloque de memoria asignado, 454
Modo (pantalla), 155, 164, 175
Modo condensado (impresora), 373
Modo de impresión enfatizado, 374
Modo de paso sencillo, 29, 117, 475
Modo de texto, 137, 155, 156
Modo expandido (impresión), 374
Modo gráfico, 137, 157, 173, 174
Modo protegido, 59
Modo real, 7, 12, 439
Mover el apuntador de archivo, 311
Mover una cadena de caracteres. *Véase* Instrucción MOVS
MSDOS.SYS, 20, 286, 438, 480
Multipalabra
  aritmética, 220
  multiplicación con, 226
Multiplicación
  datos ASCII, 246
  datos binarios, 224, 226, 524, 533
  por corrimiento, 231

N

Nivel más alto de procesamiento en disco, 294
Nivel más bajo de procesamiento en disco, 294
Nombre (de un elemento de información), 50, 61
Nombre de archivo para su análisis sintáctico, 341
Nombre de archivo, 287, 341
Nombre de segmento TEXT, 60, 79
Nombre del segmento DATA, 60, 79
Notación en complemento a dos, 4
Número de cabezas de lectura-escritura, 287
Número de la unidad física, 287
Número de unidad, 287
Número total de sectores, 287
Números negativos, 4, 65, 258

O

Obtener
  atributo de archivo, 343
  dirección de la bandera de DOS ocupado, 466
  dirección de la lista interna del DOS, 445
  dirección del PSP, 443, 485
  direcciones de interrupción, 463
  directorio actual, 339
  DPB por omisión, 328
  DPB, 329
  enlace con la memoria superior, 448
  errores extendidos, 335
  espacio libre en disco, 329
  estado de verificación, 335
  estado del botón (ratón). 380
  estado del puerto de impresión, 375
  estrategia de asignación de memoria, 447
  exhibición de página por medio del apuntador (ratón), 385
  fecha y hora de un archivo, 346
  fecha, 483
  hora, 478, 483
  identificación de medios, 334
  información de botón-liberado, 382
  información de botón-pulsado, 381
  información de la unidad por omisión, 327
  información de una unidad específica, 327
  información del dispositivo, 329
  información del ratón, 385
  modo actual de video, 164
  parámetros de la unidad, 359
  sensibilidad del ratón, 385
  tipo de disco, 361
  unidad de disco por omisión, 327
  valor de retorno de un subprograma, 456
  versión del DOS, 439, 483, 484
Ocultar el apuntador del ratón, 380
OF (bandera). *Véase* Bandera de desbordamiento (overflow)
Opción de bibliotecas, 81
Opciones de-depuración para ensamblar, 554
Operación de instrucciones de banderas, 90
Operación, 51
Operaciones booleanas, 125
Operaciones de cálculo, 488
Operador .TYPE, 279, 493
Operador DUP, 62, 490
Operador FAR, 54, 121
Operador HIGH, 489
Operador HIGHWORD, 489
Operador LENGTH, 279, 490, 493
Operador lógico, 490
Operador LOW, 490
Operador LOWWORD, 490
Operador MASK, 507
Operador NEAR, 54, 413

4FH encuentra la siguiente entrada de un directorio que cumpla un criterio, 345
50H establece la dirección del PSP, 485
51H obtiene la dirección del PSP, 443, 485
52H obtiene la dirección de la lista del DOS, 445
54H obtiene el estado de la verificación, 335
56H renombra un archivo, 346
57H obtiene/establece la fecha y hora de un archivo, 346
5800H obtiene la estrategia de asignación de memoria, 447
5801H establece la estrategia de asignación de memoria, 448
5802H obtiene enlace con la memoria superior, 448
5803H establece enlace con la memoria superior, 448
59H obtiene código extendido de error, 335
5AH crea un archivo temporal, 347
5BH crea un archivo nuevo, 347
5CH bloqueo/desbloqueo del acceso a un archivo, 486
5DH establece tabla ampliada de errores, 486
5EH servicios de red de área local, 486
5FH servicios de red de área local, 486
62H obtiene la dirección del PSP. 486
65H obtiene información ampliada del país, 486
66H obtiene/establece página global de código, 486
67H establece el total máximo de manejadores, 441
6CH abre archivos, 486
descripción/listado, 42, 137, 481
INT 22H dirección de terminación, 481
INT 23H dirección de Ctrl/break, 481
INT 24H Manejador de error crítico, 481
INT 25H lectura absoluta en disco, 321, 481
INT 26H escritura absoluta en disco, 321, 481
INT 27H termina pero permanece residente, 481
INT 2FH interrumpe la multiplexión, 481
INT 33H funciones del ratón
00H inicializa el ratón, 379
01H despliega el apuntador del ratón, 380
02H oculta el apuntador del ratón, 380
03H obtiene el estado del botón y la ubicación del apuntador, 380
04H establece la ubicación del apuntador, 380
05H obtiene la información de pulsación del botón, 381
06H obtiene la información de liberación del botón, 382
07H establece límites horizontales para el apuntador, 382
08H establece límites verticales para el apuntador, 383
09H establece el tipo de apuntador en modo gráfico, 378
0AH establece el tipo de apuntador en modo de texto, 378
0BH lee los contadores de movimiento del ratón, 383
0CH instala el manejador de interrupciones para eventos del ratón, 383
10H establece el área de exclusión del apuntador, 384
1AH establece la sensibilidad del ratón, 384
1BH obtiene la sensibilidad del ratón, 385
1DH selecciona la página de despliegue para el apuntador, 385
1EH obtiene la página de despliegue para el apuntador, 385
24H obtiene información del ratón, 385
Intensidad (pantalla), 157
Interfaz DOS-BIOS, 21
Interlineado, 374
Intermitente (pantalla), 157, 158, 169
Interna(o)
interrupción, 475
lista, del DOS, 445
memoria. *Véase* Memoria
tablas, del DOS, 21
Interrupción (es)
bandera (IF) de, 16, 101, 117. 540
del DOS, 480
dirección, 463
ejecución de, 474
instrucción de, 525
manejo de, 20
tabla de servicio de, 20, 463, 470, 474
Interrupción de impresión de la pantalla, 475
Interrupción externa, 474
Interrupción multiplexada, 481
Interrupción no enmascarable (NMI), 474
Invertir la secuencia en una palabra, 220
IO.SYS, 20, 286. 438, 480
IOCTL (control de E/S para dispositivos), 329

K

Kilobyte, 3

L

Lectura secuencial, 303, 318
Leer
archivo en disco, 303
atributo o carácter, 162
búfer del sector, 361
búfer extendido de sector, 360
carácter del teclado, 187, 188
caracteres gráficos, 175
contadores de movimiento del ratón, 383
datos de control de una unidad, 330
directamente un bloque, 320
estado del disco, 354
hora, 478
posición de la pluma óptica, 175
posición del cursor, 160
punto de pixel, 177
registro, 304, 319
sector en disco, 334
sectores, 354
Lejana (o)
dirección, 102, 113
instrucción, RET, 415
llamada, 122, 414
procedimiento, 414
Lenguaje de alto nivel, 49
Lenguaje de bajo nivel, 49
Liberar la memoria asignada, 454
Límite de un párrafo, 10, 11, 22, 53, 412
Línea A20, 439
Línea inicial de rastreo, 159, 160