# LENGUAJE ENSAMBLADOR Y PROGRAMACIÓN PARA IBM® PC Y COMPATIBLES

Tercera edición

**Peter Abel**
British Columbia
Institute of Technology

**TRADUCCIÓN:**

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

**REVISIÓN TÉCNICA:**

**Prof. Raymundo Hugo Rangel Gutiérrez**
UNAM

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 |
| EDITORIAL/PRODUCTION SUPERVISION 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.


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 MEXICO

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.

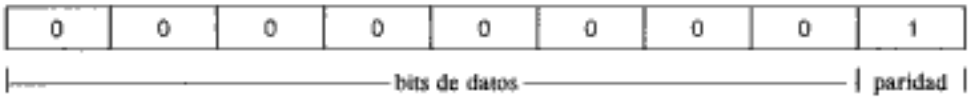
Un programa en lenguaje ensamblador consiste en uno o más *segmentos* para definir datos y almacenar instrucciones de máquina y un segmento llamado *stack* (o *pila*) que contiene direcciones almacenadas.

## 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 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Contenidos en bits (PC): | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 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 × 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 izquierda a 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.

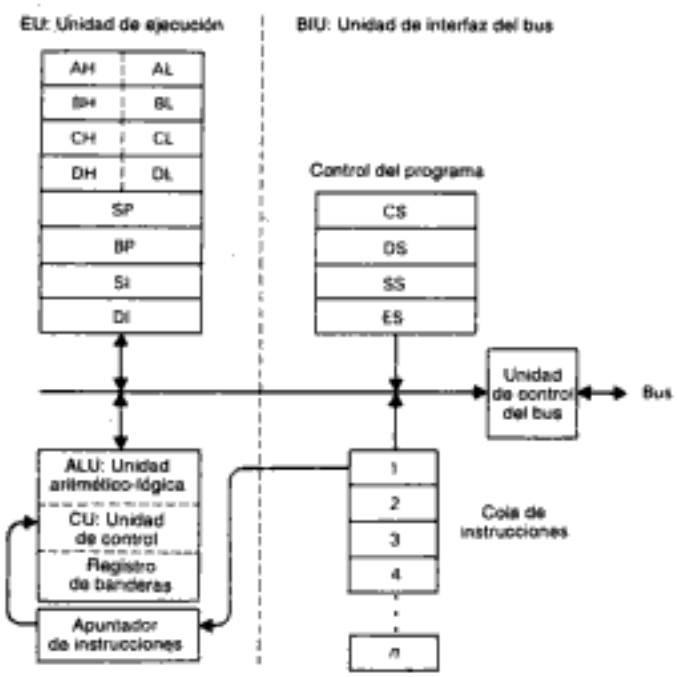



**Figura 1-2** Unidad de ejecución y unidad de interfaz del bus

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.

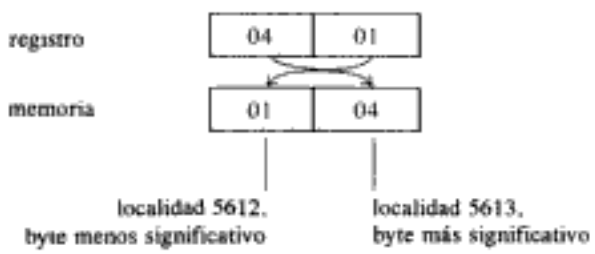

**Figura 1-3** Mapa de memoria física

**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 COMMAND.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 COMMAND.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 045F[0].

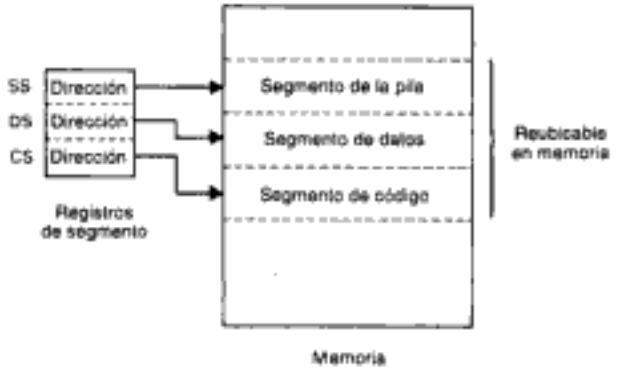
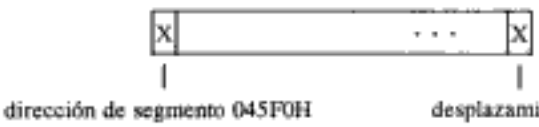



**Figura 1-4** Segmentos y registros

### 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 0000H 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.




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:

**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 |

27B3[0]H
Dirección del segmento SS

312H
Desplazamiento del SP

**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.

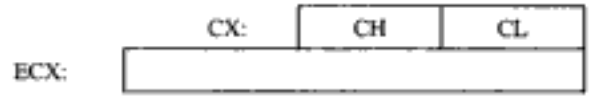
```
MOV   CX,00
MOV   CH,00
MOV   CL,00
```

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.

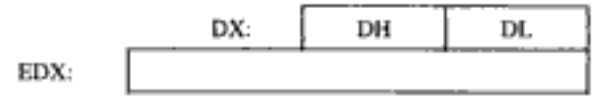
**Registro AX.** El registro AX, el acumulador principal, es utilizado para operaciones que implican entrada/salida y la mayor parte de la aritmética. Por ejemplo, las instrucciones para multiplicar, dividir y traducir suponen el uso del AX. También, algunas operaciones generan código más eficiente si 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) |
|---|---|---|---|---|
| 23A6 | 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 IBMBIO.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 IBMDOS.COM.

COMMAND.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 FFFF0H, la cual es el punto de entrada al BIOS en ROM.

La rutina de BIOS que inicia en FFFF0H 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 0 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.

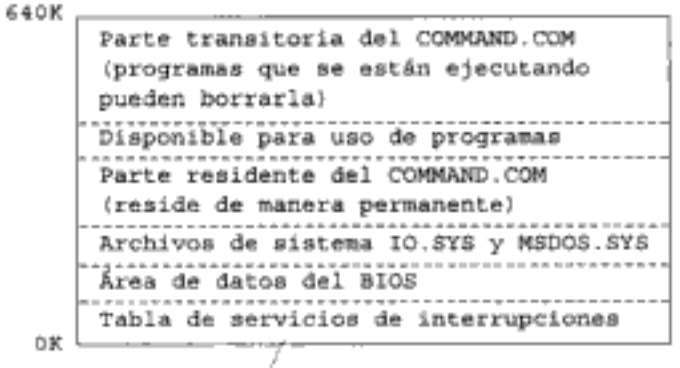



**Figura 2-1** Mapa de la memoria convencional

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 IO.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 COMMAND.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.

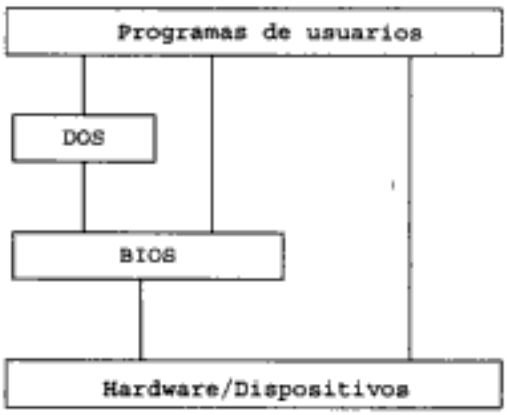



**Figura 2-2** Interfaz DOS-BIOS

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.

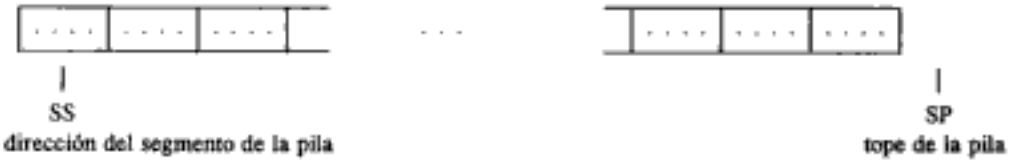
En esta forma, el cargador DOS inicializa correctamente los registros CS:IP y SS:SP. Pero note que el programa cargador almacena la dirección del PSP tanto en el registro DS como en el ES, aunque su programa normalmente necesita la dirección del segmento de datos en estos registros. Como consecuencia,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


2. 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:

5. 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 |
| 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   DW      0           ;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:xx10  xx ..... xx-xx ..... xx x.....x

 xxxx:xx20  xx ..... xx-xx ..... xx x.....x

 xxxx:xx30  xx ..... xx-xx ..... xx x.....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 xx13H, inicie con xx10H 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 00 = 1, 01 = 2, 10 = 3 y 11 = 4) |
| 5,4 | Modo inicial de video = 10 (donde 01 = 40 × 25 en color, 10 = 80 × 12 25 en color y 11 = 80 ×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 1 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. |
| 052500 | 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 de CX. |
| 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 C0 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 EI 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 EI 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 EI 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 EI 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 EI 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 EI 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 EI 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 EI 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 EI 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  .#..%.......+.+.
21C1:0110  90 C3 8D 46 14 50 51 52-FF 76 28 E8 74 00 8B E5  ...F.PQR.v(.t...
21C1:0120  B8 01 00 50 FF 76 32 FF-76 30 FF 76 2E FF 76 28  ...P.v2.v0.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:.F..".....0.=..
21C1:0150  75 32 A1 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 MÁQUINA: 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: A1 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 A1 00 02 03 06 02 02 [presione Enter]

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

El primer 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 A1 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  A1 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 EI 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 EI 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 EI 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 EI 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, 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 pila
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 n:nombredearchivo.
   - 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,[11A]
103   ADD   AX,[11C]
107   ADD   AX,25
10A   MOV   [11E],AX
10D   MOV   WORD PTR [120],25
113   MOV   BYTE PTR [122],30
118   NOP
119   NOP
11A   DB    14 23
11C   DB    05 00
11E   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 D0 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): A0 00 02 D0 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.

## IDENTIFICADORES

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 60,132
```

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 combinado 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 P04ASM1. |
| 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'
5           ...
6   STACKSG ENDS
7   ;--------------------------------------------------
8   DATASG  SEGMENT PARA 'Data'
9           ...
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, anímese: 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 00H.

## 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, MEDIUM, 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 |
| MEDIUM | 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   60,132
TITLE   P04ASM2 (EXE)  Operaciones de mover y sumar
;-------------------------------------------------
        .MODEL  SMALL
        .STACK  64                  ;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

```
FLD1  DB  ?      ;Elemento no inicializado
```

En este caso, cuando su programa inicie la ejecución el valor inicial de FLD1 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 (0EH) 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 el 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 y 0FD8H, 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 000A[ 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                              ;Cadena de caracteres
     70 75 74 65 72
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 0000000E 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 474D000000000000   FLD2DQ  DQ  04D47H          ;Constante hexadecimal
0072 3C7F000000000000   FLD3DQ  DQ  32572           ;Constante decimal

                        ;       Se definen Tenbytes - DT:
                        ;       -------------------
007A 00000000000000000  FLD1DT  DT  ?               ;No se inician
     00
0084 563412000000000000 FLD2DT  DT  123456          ;Constante decimal
     00
008E 435000000000000000 FLD3DT  DT  'PC'            ;Cadena de caracteres
     00
```

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

```
Segments and Groups:
                N a m e         Length   Align   Combine Class

DGROUP . . . . . . . . . . . . GROUP
   _DATA . . . . . . . . . . . 0098     WORD    PUBLIC  'DATA'
_TEXT  . . . . . . . . . . . . 0000     WORD    PUBLIC  'CODE'

Symbols:
                N a m e         Type     Value   Attr

FLD1DB . . . . . . . . . . . . L BYTE    0000    _DATA
FLD1DD . . . . . . . . . . . . L DWORD   004A    _DATA
FLD1DQ . . . . . . . . . . . . L QWORD   0062    _DATA
FLD1DT . . . . . . . . . . . . L TBYTE   007A    _DATA
FLD1DW . . . . . . . . . . . . L WORD    0030    _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    0034    _DATA
FLD4DB . . . . . . . . . . . . L BYTE    0003    _DATA
FLD4DD . . . . . . . . . . . . L DWORD   005A    _DATA
FLD4DW . . . . . . . . . . . . L WORD    0036    _DATA
FLD5DB . . . . . . . . . . . . L BYTE    0004    _DATA   Length = 000A
FLD5DD . . . . . . . . . . . . L DWORD   005E    _DATA
FLD5DW . . . . . . . . . . . . L WORD    0040    _DATA   Length = 0005
FLD6DB . . . . . . . . . . . . L BYTE    000E    _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   ..<.............
-D
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 00-72 03 E9 6B 01 2B C0 50   ........r..k.+.P
0F07:00A0  50 FF 76 04 E8 F5 5D 83-C4 06 0B D0 74 03 E9 57   P.v...]....t..W
0F07:00B0  01 B8 FF FF 50 2B C0 50-FF 76 04 E8 DE 5D 83 C4   ....P+.P.v...]..
0F07:00C0  06 8B 1E A4 43 FF 06 A4-43 D1 E3 D1 E3 A1 0E 3C   ....C...C......<
0F07:00D0  8B 16 10 3C 89 87 8A 32-89 97 8C 32 5E 8B E5 5D   ...<...2...2^..]
0F07:00E0  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   l....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      ;El 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
MPY       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), MEDIUM (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 SAL_EMP.
- 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,25H     ;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) PC_AT; (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, MEDIUM, 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:P05ASM1.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:

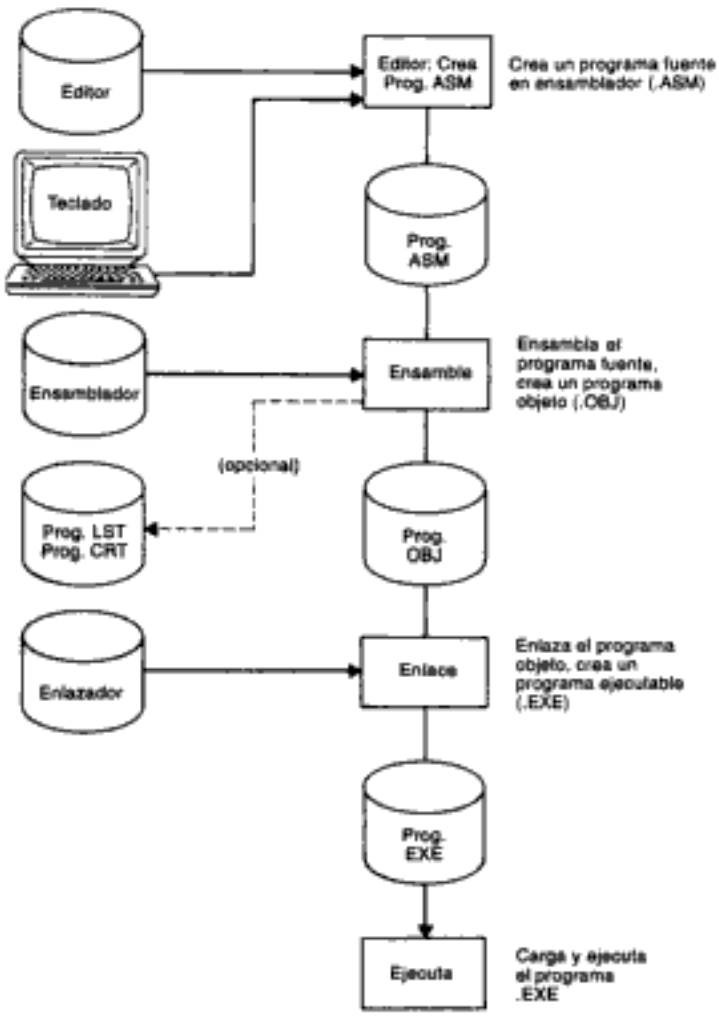



**Figura 5-1** Pasos para ensamblar, enlazar y ejecutar

```
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           ]
 8
 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  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  A1 0000 R              MOV     AX,FLDA         ;Mover 0250 a AX
24 0008  03 06 0002 R           ADD     AX,FLDB         ;Sumar 0125 a AX
25 000C  A3 0004 R              MOV     FLDC,AX         ;Almacenar suma en FLDC
26 000F  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:
          N a m e                 Length    Align     Combine     Class
CODESG . . . . . . . . . . . . .  0014      PARA      NONE        'CODE'
DATASG . . . . . . . . . . . . .  0006      PARA      NONE        'DATA'
STACKSG  . . . . . . . . . . . .  0040      PARA      STACK       'STACK'

Symbols:
          N a m e                   Type      Value     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      0101h
@FILENAME  . . . . . . . . . . .  TEXT      p05asm1
@VERSION . . . . . . . . . . . .  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 × 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 A1 0000. El espacio entre A1 (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 *value* 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 @VERSION 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 |
| @VERSION | 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 simbolos que construyó en la pasada 1. Ahora que "conoce" la longitud y posiciones relativas de cada campo de datos e instrucción,

```
P05ASM2 (EXE)  Operaciones de mover y sumar                    Page     1-1

                              page   60,132
                      TITLE   P05ASM2 (EXE)  Operaciones de mover y sumar
                      ;--------------------------------------------------
                              .MODEL  SMALL
                              .STACK  64              ;Se define la pila
                              .DATA                   ;Se definen datos
0000  00FA            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  A1 0000 R               MOV     AX,FLDA         ;Mover 0250 a AX
0008  03 06 0002 R            ADD     AX,FLDB         ;Sumar 0125 a AX
000C  A3 0004 R               MOV     FLDC,AX         ;Almacenar suma en FLDC

000F  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:
                N a m e         Length     Align    Combine Class
DGROUP . . . . . . . . . . . .GROUP
  _DATA  . . . . . . . . . . .0006       WORD     PUBLIC     'DATA'
  STACK  . . . . . . . . . . .0040       PARA     STACK      'STACK'
_TEXT  . . . . . . . . . . . .0014       WORD     PUBLIC     'CODE'

Symbols:
                N a m e         Type       Value    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       0101h
@DATASIZE  . . . . . . . . . .TEXT       0
@FILENAME  . . . . . . . . . .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 | NAME | CLASS |
|---|---|---|---|---|
| 00000H | 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 | NAME | CLASS |
|---|---|---|---|---|
| 00000H | 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#
@VERSION . . . . . . . . . . . .  1#

BEGIN. . . . . . . . . . . . . . 18#    28      30

CODE . . . . . . . . . . . . . . 17
CODESG . . . . . . . . . . . . . 17#    19      29

DATA . . . . . . . . . . . . . . 11
DATASG . . . . . . . . . . . . . 11#    15      19      20

FLDA . . . . . . . . . . . . . . 12#    23
FLDB . . . . . . . . . . . . . . 13#    24
FLDC . . . . . . . . . . . . . . 14#    25+

STACK. . . . . . . . . . . . . .  4
STACKSG. . . . . . . . . . . . .  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. |

```
 1                                page 60,132
 2                    TITLE       P05ASM3 (EXE) Ilustra errores de ensamblado
 3                    ; ---------------------------------------------
 4 0000               STACKSG     SEGMENT PARA STACK 'Stack'
 5 0000  0020[                    DW      32 DUP(0)
 6             0000
 7                 ]
 8
 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  A1 0000 U               MOV     AX,DATSEG       ;Dirección de DATASG
p05asm3.ASM(17): error A2009: Symbol not defined: DATSEG
21 0003  8B D0                   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
- AAS: Ajuste ASCII después de restar
- DAA: 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

**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 | operando1, 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:

```
WORD1 DW 0
BYTE1 DB 0
      ...
      MOV AX,WORD1          ;Carga WORD1 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 *especificadores 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 en corchetes 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,DWORDVAL). 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  60,132
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      ?
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   20  DUP  (?)
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     'ABCDEPGHI'
NAME2   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,NAME1      ;Iniciación de direcciones para NAME1
        LEA    DI,NAME2      ;     Y NAME2
B20:
        MOV    AL,[SI]       ;Obtener carácter de NAME1,
        MOV    [DI],AL       ;Moverlo a NAME2
        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

```
NAME1:  A  B  C  D  E  F  G  H  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 NAME1 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 NAME1 y NAME2 en SI y DI como sigue:

```
LEA  SI,NAME1     ;Carga desplazamientos

LEA  DI,NAME2     ; de NAME1 y NAME2
```

El programa utiliza las direcciones de los registros SI y DI para mover el primer byte de NAME1 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 (NAME1 +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.

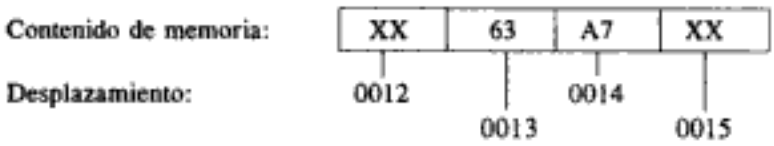
Para regresar de una interrupción, la rutina emite un IRET (regreso de interrupción), el cual saca los registros de la pila y regresa a la instrucción inmediata posterior al INT en su programa.

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:

```
BYTE1  DB ?
BYTE2  DB ?
WORD1  DW ?
 (a)   MOV BYTE1, 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 WORD1 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 /T para TLINK:

```
TLINK /T D:CALC
```

## EJEMPLO DE UN PROGRAMA .COM

El programa de la figura 7-1, llamado EXCOM1, 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:EXCOM1,D: | TASM D:EXCOM1,D: |
| LINK D:EXCOM1,D: | TLINK /T D:EXCOM1,D: |
| EXE2BIN D:EXCOM1 D:EXCOM1.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:EXCOM1.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.

CAPÍTULO 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,00
```

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 00H 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  D1 E1             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 D1 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: | | | | | O | 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 C10. 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     C10             ;Llamada a C10
                ;       ...
000B C3                 RET                      ;De regreso a
000C            B10     ENDP                     ;  quien llama
                ;-----------------------------------------
000C            C10     PROC     NEAR
                ;       ...
000C C3                 RET                      ;De regreso a
000D            C10     ENDP                     ;  quien llama
                ;-----------------------------------------
                        END      BEGIN
```

**Figura 8-3** Llamada a procedimientos

- Las directivas PROC para B10 y C10 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 C10 e inicia su ejecución.
- En el procedimiento C10, la instrucción RET hace que el control regrese a la instrucción que sigue a CALL C10.
- 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 C10. De hecho, si C10 no contiene un RET el programa ejecutaría, pasando el final de C10, 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.

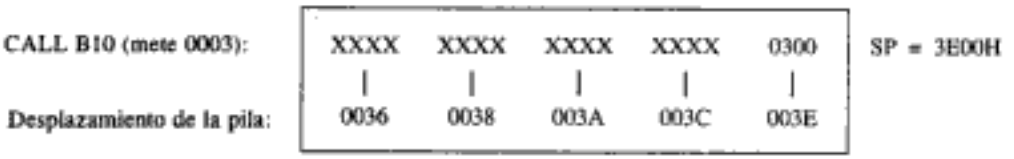
Como un recordatorio, al cargar un programa .EXE para ejecución el cargador de sistema establece los valores siguientes en los registros:

- 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.

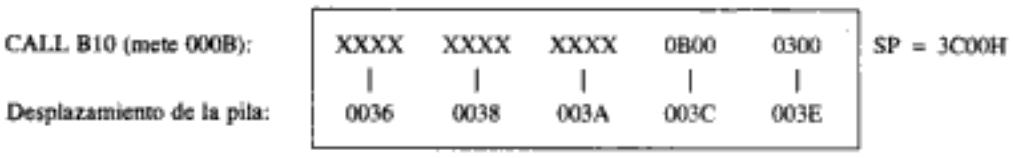
Rastree el programa sencillo de la figura 8-3 a lo largo de su ejecución. En la práctica, las llamadas procedimientos tendrían cualquier número de instrucciones.

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:

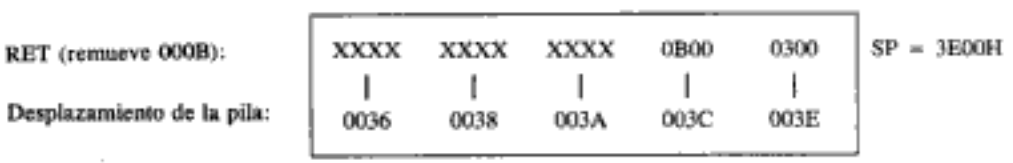
- CALL B10 disminuye en 2 el SP, de 40H a 3EH. Después mete el IP (con 0003) en el tope de la pila en el desplazamiento 3EH. Éste es el desplazamiento de la instrucción que sigue a la instrucción CALL. El procesador utiliza la dirección formada por CS:IP para transferir el control a B10. Las palabras en memoria contienen bytes en orden inverso; por ejemplo 0003 se convierte en 0300.



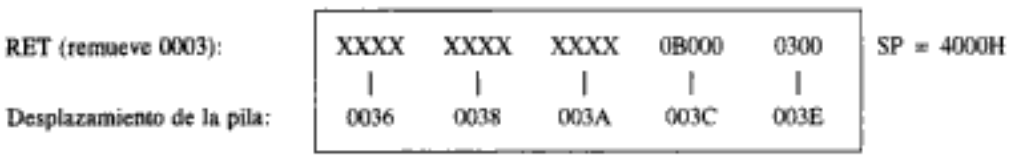

- En el procedimiento B10, CALL C10 disminuye en 2 el SP, a 3CH. Después agrega el IP (con 000B) en el tope de la pila en el desplazamiento 3CH. El procesador utiliza las direcciones CS:IP para transferir el control a C10.




- Para regresar de C10, 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    ...              ;  izquierda es BL en diferente de cero?
2. TEST   AL,00000001B     ;¿AL contiene
   JNZ    ...              ;  un número impar?
3. TEST   DX,0FFH          ;¿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,0FFH 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, invirtiendo 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+1         ;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).

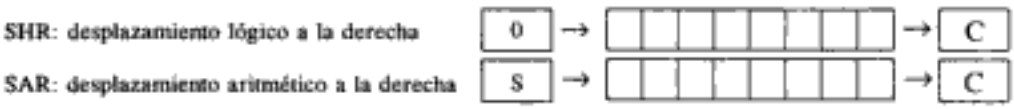
El segundo operando contiene el valor del corrimiento, que es una constante (un valor inmediato) o una referencia al registro CL. Para los procesadores 8088/8086, la constante inme-

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:] | corrim. | {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):




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):

ROR: Rotación lógica a la derecha

RCR: Rotación a la derecha con acarreo


Las siguientes instrucciones relacionadas ilustran ROR:

```
INSTRUCCIÓN              BH              COMENTARIO
MOV CL,03
MOV BH,10110111B     ; 10110111
ROR BH,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):

ROL: Rotación lógica a la izquierda

RCL: Rotación a la izquierda con acarreo

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     D00DSCT
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: CUSTTBL+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     D00NODSC  ;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         CUSCODE 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 C0              MOV    ES,AX         ;  de segmento
                ;       ...
0007 E8 000F R          CALL   D10JUMP       ;Invocar rutina de saltos
                ;       ...
000A B8 4C00            MOV    AX,4C00H      ;Salida a dos
000D CD 21              INT    21H
000F            BEGIN   ENDP

000F            D10JUMP PROC   NEAR
000F 8A 1E 000A R       MOV    BL,CUSCODE    ;Obtener código de descuento
0013 32 FF              XOR    BH,BH         ;Limpiar parte alta de BX
0015 D1 E3              SHL    BX,01         ;Multiplicar por dos el valor
0017 FF A7 0000 R       JMP    [CUSTTBL+BX]  ;A rutina de tabla
                ;       ...

001B            D00NODSC:                    ;Rutina código 0
                ;       ...
001B EB 0D 90           JMP    D90RET
001E            D10DSCT:                     ;Rutina código 1
                ;       ...
001E EB 0A 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
- Salto1:

- Mover un carácter del nombre
- Incrementar para pasar al siguiente carácter de nombre
- Decrementar el contador: si no es cero, Salto1
- 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 C10 y C10 llama a D10. 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,11111111B; (e) AND BL,00000000B.

**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; (f) 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 0DH 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 |
| | | 0AH | 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 = 00H para la pantalla completa
- BH = número del atributo
- CX = renglón:columna iniciales
- DX = renglón:columna 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 00H y de manera sucesiva es incrementado en uno para desplegar cada carácter hasta alcanzar FFH.

La primera línea desplegada inicia con un blanco (00H), 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, 0AH provoca un

avance de línea y 0DH (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 0EH, 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 0DH; puede querer experimentar con esta desviación, digamos, sólo para 08H (Retroceso) y 0DH (Retorno de carro).

```
        CMP   CHAR,08H          ;¿Menor a 08H?
        JB    D30               ;Sí, entonces aceptar
        CMP   CHAR,0DH          ;¿Menor o igual a 0DH?
        JBE   D40               ;Sí, entonces evitarlo
D30:
        MOV   AH,09H            ;Desplegar los menores que 08H
        ...                     ;y los mayores que 0DH
        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 0AH DEL DOS PARA ENTRADA DEL TECLADO

En particular, la función 0AH 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 0AH en el AH, cargue la dirección de la lista de parámetros (en el ejemplo NAMEPAR), en el DX, y emita INT 21H:

```
MOV  AH,0AH           ;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 (0DH) 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 | 0D | 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 (0DH) está en NAMEFLD+5. (Aquí el símbolo # indica este carácter, ya que 0DH no es un símbolo imprimible.) Puesto que la longitud máxima es de 20, incluyendo el 0DH, 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 en 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
NAMEPAR   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     'Name? ', '$'
;-----------------------------------------------------------
          .CODE
BEGIN     PROC   FAR
          MOV    AX,@data          ;Iniciar registros
          MOV    DS,AX             ; de segmento
          MOV    ES,AX
          CALL   Q10CLR            ;Despejar pantalla
A20LOOP:
          MOV    DX,0000           ;Fijar cursor en 00,00
          CALL   Q20CURS
          CALL   B10PRMP           ;Exhibir indicación
          CALL   D10INPT           ;Proporciona entradas del nombre
          CALL   Q10CLR            ;Despejar pantalla
          CMP    NAMELEN,00        ;¿Se ingresó el nombre?
          JE     A30               ; no, salida
          CALL   E10CODE           ;Fijar campana y '$'
          CALL   F10CENT           ;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,0AH            ;Petición de teclado
          LEA    DX,NAMEPAR        ; entrada
          INT    21H
          RET
D10INPT   ENDP
```

**Figura 9-2** Cómo aceptar y mostrar nombres

```
;                       Fijar campana y delimitador '$'
;                       -------------------------------
E10CODE   PROC    NEAR
          MOV     BH,00               ;Reemplaza carácter Enter (0D)
          MOV     BL,NAMELEN          ; con el de la campana (07)
          MOV     NAMEFLD[BX],07
          MOV     NAMEFLD[BX+1],'$'   ;Pone el delimitador de exhibición
          RET
E10CODE   ENDP
;                       Centrar y exhibir nombre
;                       ------------------------
F10CENT   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    Q20CURS             ;Fija cursor
          MOV     AH,09H
          LEA     DX,NAMEFLD          ;Exhibe nombre
          INT     21H
          RET
F10CENT   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
;                       ------------------------------
Q20CURS   PROC    NEAR                ;DX fija en entrada
          MOV     AH,02H              ;Petición de ubicar cursor
          MOV     BH,00               ;Página 0
          INT     10H
          RET
Q20CURS   ENDP

          END     BEGIN
```

**Figura 9-2** (continuación)

```
Renglón 12:  Pat    Brown
                |       |

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 (0DH)
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): |14|00|0D| ...
```

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 (0DH) 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              NAMEPAR (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 0DH)
LF       EQU  10     ;(o EQU 0AH)
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
A30:     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 y 01 para despliegue en pantalla. Otros manejadores de archivo, como aquellos para dispositivos de disco, tienen que ser establecidos por su programa. También puede utilizar estos servicios para redireccionar la entrada y la salida a otros dispositivos, aunque esta característica por el momento 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 las 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 de acarreo (la cual puede usted examinar).

Una operación INT fallida pone en uno la bandera de acarreo y regresa un código de error 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, la única forma de determinar una condición de error es probar la bandera de acarreo, aunque los 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 07H (Campana), 08H (Retroceso), 0AH (Avance de línea) y 0DH (Retorno de carro). Las instrucciones siguientes ilustran esta operación:

```
DISAREA  DB     'PC Users Society', 0DH, 0AH   ;Área de despliegue
         ...
         MOV    AH,40H                        ;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):

```
100  MOV AH,40
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 name'
```

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 0AH 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(' ') ;Área 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 (0DH) señala la terminación de una entrada. Por ejemplo, el tecleo de los caracteres "PC Users Group" introduce lo siguiente en INAREA:

```
|PC Users Group|0DH|0AH|
```

Los caracteres tecleados son seguidos de manera inmediata por un Enter (0DH), que usted tecleó, y un avance de línea (0AH) 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 SÍ 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, 0DH, 0AH     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 0DH y 0AH. 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 0AH 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 0AH 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 |
| 0AH | Despliega el carácter en la posición del cursor |
| 0BH | Establece la paleta de colores |
| 0CH | Escribe el pixel punto |
| 0DH | Lee el pixel punto |
| 0EH | Escribe en teletipo |
| 0FH | 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] Para HCG
- 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 0FH (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 × 25, usando un adaptador de color
- 10:80 × 25, usando un adaptador de color
- 11:80 × 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 cols) | Mono | CGA | 320 x 200 | |
| | | | EGA | 320 x 350 | |
| | | | MCGA | 320 x 400 | |
| | | | VGA | 360 x 400 | |
| 01 | (25 renglones, 40 cols) | 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 cols) | Mono | CGA | 640 x 200 | |
| | | | EGA | 640 x 350 | |
| | | | MCGA | 640 x 400 | |
| | | | VGA | 720 x 400 | |
| 03 | (25 renglones, 80 cols) | 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 cols) | 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 el *frente* 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 BL R G B | Frente I R G B | 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 | I R G B | Color | I R G B |
|---|---|---|---|
| Negro | 0 0 0 0 | Gris | 1 0 0 0 |
| Azul | 0 0 0 1 | Azul claro | 1 0 0 1 |
| Verde | 0 0 1 0 | Verde claro | 1 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 $80 \times 25 \times 2 = 4{,}000$ 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 y 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 cían, con primer plano en negro
MOV CX,0000     ;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,61H         ;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 00H. 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 está en el AL.

A continuación veremos un ejemplo que despliega 80 guiones y establece un atributo de 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 0AH: 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 0AH y 09H en modo de texto es que la función 0AH utiliza el atributo actual, mientras que la función 09H establece el atributo. Aquí está el código para esta función:

```
MOV   AH,0AH          ;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 0EH: Escribir en teletipo

Esta operación le permite utilizar un monitor como terminal para despliegue simple. Establezca la función 0EH 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,0EH          ;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 (0DH) y avance de línea (0AH) 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 0FH: Obtiene el modo actual de video

Utilice esta función para determinar el modo actual de video. (Véase también la función 00H.) Aquí está un ejemplo:

```
MOV AH,0FH          ;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 subfunciones 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.

0AH 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,0FFH        ;¿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    F10READ         ;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,0FH          ;Petición para obtener el modo
          INT     10H
          MOV     MODE,AL         ;Guardar el modo
          MOV     AH,00H          ;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,0AH          ;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
;                       ---------------------------------------------------------------------------
F10READ     PROC        NEAR
            MOV         AH,10H          ;Petición para obtener un carácter
            INT         16H
            RET
F10READ     ENDP
;                       Restaura el modo de video original
;                       ---------------------------------------------------
G10MODE     PROC        NEAR
            MOV         AH,00H          ;Petición para establecer el modo
            MOV         AL,MODE         ;Valor original
            INT         10H
            RET
G10MODE     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,0FH          ;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:

```
MENU    DB    0DAH  17 DUP(0C4H),     0BFH
        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,MENU         ;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 |
|---|---|
| B0 | 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 Franklin [intermitente]
|                                |
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 0AH de la INT 10H.

```
          page    60,132
TITLE     P10NMSCR (EXE) Video inverso, intermitencia y recorrido de la pantalla
          .MODEL  SMALL
          .STACK  64
; -----------------------------------------------------
          .DATA
NAMEPAR   LABEL   BYTE              ;Lista de parámetros:
MAXNLEN   DB      20                ;  longitud máxima del nombre
ACTNLEN   DB      ?                 ;  número de caracteres ingresados
NAMEFLD   DB      20 DUP(' ')       ;  para el nombre

COL       DB      00
COUNT     DB      ?
PROMPT    DB      'Name? '
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,0AH              ;Petición de entrada
            LEA     DX,NAMEPAR          ;  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 nombre
             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,0601H           ;  sí,
             CALL    Q10SCR             ;  recorre la pantalla
             RET
 E10NAME     ENDP
 ;                       Despliegue de carácter:
 ;                       -----------------------------------
 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 pantalla:
 ;                       ---------------------------------
 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 cursor en renglón/columna:
 ;                       ------------------------------------------------------
```

**Figura 10-3** (continuación)

```
Q20CURS   PROC   NEAR
          MOV    AH,02H
          MOV    BH,00          ;Página
          MOV    DH,ROW         ;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 $80 \times 25 \times 2 = 4{,}000$ 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.) VIDAREA 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[?]    VIDAREA DB      1000H DUP(?)
1000            VIDSEG  ENDS
                ; ------------------------------------------------
                        .STACK 64
                ; ------------------------------------------------
                        .CODE
0000            BEGIN   PROC    FAR
0000 B8 ---- R          MOV     AX,VIDSEG       ;Direccionabilidad para
0003 8E C0              MOV     ES,AX           ;  el área de video
                        ASSUME ES:VIDSEG
0005 B4 0F              MOV     AH,0FH          ;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,00H          ;Petición para designar
000D B0 03              MOV     AL,03           ;  el modo 03, y limpiar la pantalla
000F CD 10              INT     10H;
0011 B4 05              MOV     AH,05H          ;Petición para designar
0013 B0 01              MOV     AL,01H          ;  la página #01
0015 CD 10              INT     10H
0017 E8 002E R          CALL    C10PROC         ;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,00H          ;  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 B0 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 26: 89 85 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 C0              INC     AL              ;Carácter siguiente
0045 83 C7 28           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            C10PROC 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 × 2 = 160 columnas en un renglón. Entonces la posición inicial para el renglón 10, columna 10, es (160 × 10 renglones) + (10 columnas × 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 VIDAREA. 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 [VIDAREA+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,00H      ;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 | |
| 0DH | Color | EGA,VGA | 320 x 200 | 16 |
| 0EH | Color | EGA,VGA | 640 x 200 | 16 |
| 0FH | 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 |

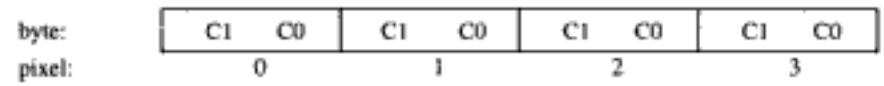
**Figura 10-5** Modos gráficos para despliegue en video

- *Modos gráficos 0DH, 0EH, 0FH 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 | C0 | Paleta 0 | Paleta 1 |
|---|---|---|---|
| 0 | 0 | fondo | fondo |
| 0 | 1 | verde | cian |
| 1 | 0 | rojo | magenta |
| 1 | 1 | café | blanco |

Utilice la función 0BH 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 0AH: Despliega un carácter en la posición del cursor

Véase la sección anterior, "Interrupción 10H del BIOS para el modo de texto".

### INT 10H, función 0BH: 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,0BH      ;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,0BH     ;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 0BH mientras está en modo de texto, el número establecido para el color 0 de la paleta determina el color del borde.

### INT 10H, función 0CH: Escribe un pixel punto

Utilice la función 0CH 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,0CH        ;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 0DH, 0EH, 0FH 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 0DH: Lee un pixel punto

Esta operación, la opuesta de la función 0CH, 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 0EH: 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:

- 0FH: Conserva el modo original
- 00H: Establece el modo gráfico
- 0BH: Selecciona el fondo en color verde
- 0CH: 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,0FH          ;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,00H          ;Establece el modo gráfico EGA/VGA
          MOV     AL,10H          ;640 cols x 350 renglones
          INT     10H
          MOV     AH,0BH          ;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,0CH          ;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 B1) 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 0AH 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 × 200; (b) 640 × 200; (c) 640 × 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 0AH 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 espacio en memoria para que usted teclee por adelantado antes de que un programa solicite en realidad una 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 |
| 0AH | Entrada al búfer del teclado |
| 0BH | Verificación del estado del teclado |
| 0CH | 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 (F1, 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, F11 y F12, proporcionan una función nueva; el resto duplican la función de teclas en el teclado original. Si sus programas permiten presionar F11, 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 00H. 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 0AH, 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, F1 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,00      ;¿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 0FFH 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, F1 o RePág. Para obtener el código de rastreo en el AL, repita de manera inmediata la operación INT 21H:

```
K10:    MOV  AH,06H      ;Petición directa a la consola
        MOV  DL,0FFH     ;Entrada del teclado
        INT  21H         ;Llama al DOS
        JZ   K10         ;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 0FFH) 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 0AH 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 0BH de la INT 21H: Verificación del estado del teclado**

Esta operación regresa FFH en el AL si un carácter está disponible y 00H si ningún carácter está disponible. La función está relacionada a aquellas otras que no esperan por entrada del teclado.

### Función 0CH 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 0AH. Cargue la función que necesite en el AL:

```
MOV  AH,0CH       ;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       ; sí
```

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 00H, 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 00H 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  16H             ;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 00H, salvo que también acepta las teclas adicionales de función extendidas (como F11 y F12) desde el teclado ampliado, mientras que la función 00H 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 F1, el carácter en el AL es 00H. En el teclado ampliado, F11 y F12 también regresan 00H en el AL, pero otras teclas de control (duplicados), como Inicio y RePág, regresan E0H. 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 o E0H |

Puede examinar el AL contra 00H o E0H 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   G40      ; -sí
CMP  AL,0E0H  ;¿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 F1 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 colo- 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 |
|---|---|
| 1E | 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 E0H 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 00H o E0H, 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,00H     ;¿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  16H      ;Llama al BIOS
        CMP  AL,00H   ;¿Es una función extendida?
        JE   G30      ; sí, entonces pasarlo
        CMP  AL,E00H  ;¿Es una función extendida?
        JNE  G90      ; no, entonces salir
G30:    CMP  AH,47H   ;¿Es el código de rastreo de Inicio?
        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 F11 y F12 generan 85H y 86H. El código siguiente prueba la tecla de función programable F10:

```
CMP AH,44H   ;¿Es la tecla de función F10?
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 o MOV AH,10
INT  16
JMP  100
```

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     ?               ;Líneas exhibidas
ATTRIB  DB     ?               ;Atributo de pantalla
NINTEEN DB     1               ;Ancho del menú
MENU    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(0CDH), 0BCH

PROMPT  DB     09, '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+1
        CALL   Q20CURS         ;Fijar cursor
        MOV    ROW,TOPROW+1    ;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,0DH          ;¿Enter presionado?
        JE     A10LOOP         ;     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

```
                LEA     SI,MENU
                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
;                ------------------------
D10INPT         PROC    NEAR
                MOV     AH,10H          ;Petición de entrada
                INT     16H             ;     del teclado
                CMP     AH,50H          ;¿Flecha hacia abajo?
                JE      D20
                CMP     AH,48H          ;¿Flecha hacia arriba?
                JE      D30
                CMP     AL,0DH          ;¿Tecla Enter?
                JE      D90
                CMP     AL,1BH          ;¿Tecla escape?
                JE      D90
                JMP     D10INPT         ;Ninguna, procesar de nuevo
D20:            MOV     ATTRIB,71H      ;Azul sobre blanco
                CALL    H10DISP         ;Fijar la línea anterior a video normal
                INC     ROW
                CMP     ROW,BOTROW-1    ;¿Se pasó la hilera interior?
                JBE     D40             ; no, muy bien
                MOV     ROW,TOPROW+1    ; sí, restablecer
                JMP     D40
D30:            MOV     ATTRIB,71H      ;Video normal
                CALL    H10DISP         ;Fijar línea anterior a video normal
                DEC     ROW
                CMP     ROW,TOPROW+1    ;¿Abajo de la hilera superior?
                JAE     D40             ; no, muy bien
                MOV     ROW,BOTROW-1    ; sí, restablecer
D40:            CALL    Q20CURS         ;Fijar cursor
                MOV     ATTRIB,16H      ;Video inverso
                CALL    H10DISP         ;Fijar nueva línea a video inverso
                JMP     D10INPT
D90:            RET
D10INPT         ENDP
;               Fijar línea de menú a normal/resaltada
;               --------------------------------------
H10DISP         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    Q20CURS             ;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+1        ;Restablecer columna a la izquierda
            CALL    Q20CURS             ;Fijar cursor
            RET
H10DISP     ENDP
;                    Despejar pantalla
;                    -----------------
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
Q20CURS     ENDP
            END     BEGIN
```

**Figura 11-1** (continuación)

- BEGIN llama a Q10CLR para limpiar la pantalla, llama a B10MENU para desplegar los elementos del menú y establecer el primer elemento en video inverso y llama a D10INPT 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, la flecha hacia arriba para subir por el menú. Enter para aceptar un elemento del menú y Esc para salir. Las demás entradas del teclado son ignoradas. La rutina da vuelta alrededor del cursor, de manera que tratar de mover el cursor por arriba de la primera línea del menú lo coloca en la última línea y viceversa. La rutina también llama a H10DISP para restaurar la línea anterior del menú a video normal y la nueva línea del menú (seleccionada) a video inverso.
- H10DISP muestra la línea actualmente seleccionada de acuerdo con un atributo (normal o 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 completo ejecutaría una rutina para cada elemento seleccionado. Entenderá mejor este programa tecleándolo 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 ... |

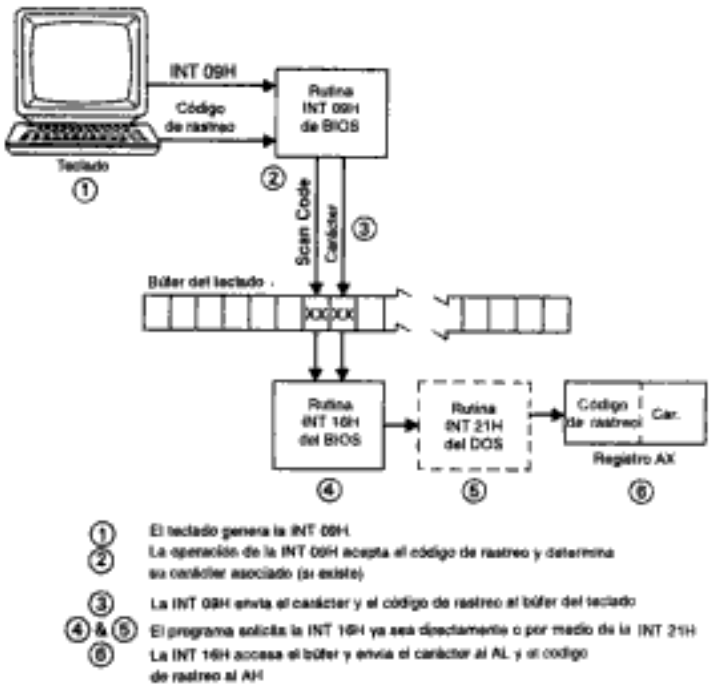



**Figura 11-2** Búfer del teclado

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>   . . .
      |     |     |     |       |       |
     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
  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
 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) 0AH.

**11-3.** Explique las diferencias entre las funciones 00H, 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 F1 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 basica | 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):

```
STRING1   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,STRING1  ;Inicializa la dirección emisora
          REP   MOVSB       ;Copia STRING1 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             ;O DEC DI
            INC   SI             ;O 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
NAME2   DB      10 DUP(' ')
NAME3   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,NAME2            ;  de NAME1 a NAME2
        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 NAME3
        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,0FH        ;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    D10PROMPT     ;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 video
;                 ----------------------------------------------
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
C10PROC   ENDP
;                 Indicación a usuario para presionar tecla
;                 -----------------------------------------
D10PROMPT 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 se combinan con un prefijo REP y una longitud en el CX, de manera sucesiva CMPS puede comparar cadenas de cualquier longitud.

Pero observe que CMPS proporciona una comparación *alfanumérica*, esto es, una comparación de acuerdo con los valores ASCII. La operación no es adecuada para comparaciones algebraicas, que consisten en números con signo. Considere la comparación de dos cadenas que contienen JEAN y JOAN. Una comparación de izquierda a derecha, tiene el resultado siguiente:

| | |
|---|---|
| J:J | Iguales |
| E:O | 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 comparación entre dos caracteres sea diferente. Para este propósito, REP tiene una variación, REPE (Repite cuando sea igual), que repite la operación mientras la comparación entre caracteres sea igual, o hasta que el registro CX sea igual a cero. El código para la comparación repetida de un byte es REPE CMPSB.

La figura 12-5 consta de dos ejemplos que utilizan CMPSB. El primero compara NAME1 con NAME2, que contienen los mismos valores. Por tanto, la operación CMPSB se realiza con 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 cero indica igual o cero.

El segundo ejemplo compara NAME2 con NAME3, que contienen valores diferentes. La operación CMPSB termina después de comparar el primer byte resultando una condición alta o diferente: El CX contiene 09, el DI contiene la dirección de NAME3+1, el SI contiene la 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 al registro BH. El segundo ejemplo resulta diferente y mueve 02 al registro BL. Si utiliza DEBUG 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. Inicialice 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
NAME2   DB      'Assemblers'
NAME3   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
        REPE CMPSB                  ;Compare NAME1: NAME2
        JNE     G20                 ;  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: NAME3
        JE      G30                 ;  igual, salir
        MOV     BL,02               ;  no es igual, fijar BL
G30:
        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, en 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  K20                ;¿Se encontró el carácter?
        DEC  DI                 ; sí, ajusta dirección
        MOV  BYTE PTR[DI],20H   ;Reemplace con un blanco
K20:    ...
```

## 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:

```
                    ***###***###
                      |     |
                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:

```
***###***###***###***###***###    ...    ***###
|           |           |                 |
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 DUP(' ')       ;Nombre

PROMPT    DB     'Name?', '$'
NAMEDSP   DB     31 DUP(' '), 13, 10, '$'
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,0FFH      ;¿No hay nombre? (indica fin)
          JZ     A90               ;  sí, salir
          CALL   D10SCAS           ;Escudriñar 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,0AH
          LEA    DX,NAMEPAR        ;Aceptar entrada
          INT    21H
          RET
B10INPT   ENDP
;                  Escudriñar asterisco en nombre
;                  ------------------------------
D10SCAS   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+30           ;Posición a la derecha de exhibición
name
                REP MOVSB                       ;Mover cadena derecha a
left
                MOV     DH,ROW
                MOV     DL,46
                CALL    Q20CURS                 ;Fijar cursor
                MOV     AH,09H
                LEA     DX,NAMEDSP              ;Exhibir nombre
                INT     21H

                CMP     ROW,20                  ;¿Parte inferior de la pantalla?
                JAE     E20                     ;  no,
                INC     ROW                     ;  incrementar hilera
                JMP     E90
E20:
                MOV     AX,0601H                ;  sí,
                CALL    Q10SCR                  ;  recorrer y
                MOV     DH,ROW                  ;  fijar cursor
                MOV     DL,00
                CALL    Q20CURS
E90:            RET
E10RGHT         ENDP
;                       Clear name:
;                       -----------
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
Q20CURS         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 uno (CLD) 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 NAME+8 ya que la operación de cadena de caracteres accesa NAME+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

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. Como hemos sumado dos números positivos, esperamos que 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      40H
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

;       Ejemplos de RESTA (SUB) de palabras:
;       ------------------------------------
C10SUB  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   60H
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 WORD1B) 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
; ------------------------------------------------------
WORD1A  DW      0BC62H              ;Datos
WORD1B  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,WORD1B           ;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,WORD1A           ;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:   62BC   2301

WORD2A y WORD2B:   3A55   1200
```

El primer procedimiento suma WORD2A a WORD1A 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:     BC62H
WORD2A:    +553AH
           -------
Total:    (1)119CH    (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 1B:          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 WORD1A:WORD1B 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 CON SIGNO | 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 CON SIGNO | 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 CON SIGNO | 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 CON SIGNO | OF | CF |
|---|---|---|---|---|
| 11110110 | 246 | - 10 | | |
| +10001001 | +137 | -119 | | |
| (1)01111111 | 127 | +127 | 1 | 1 |
| | (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

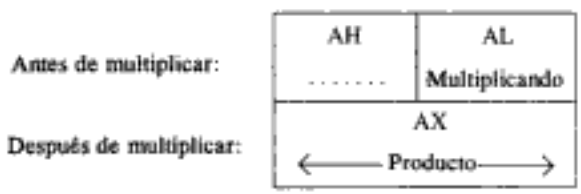
Para la multiplicación, la instrucción MUL maneja datos sin signo y la instrucción IMUL (multiplicación entera) maneja datos con signo. Ambas instrucciones afectan las banderas de acarreo y de desbordamiento. Como programador, usted tiene el control sobre el formato de los datos que procesa, y tiene 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:

```
BYTE1   DB  ?
WORD1   DW  ?
DWORD1  DD  ?
```

| OPERACIÓN | MULTIPLICADOR | MULTIPLICANDO | PRODUCTO |
|---|---|---|---|
| MUL BYTE1 | BYTE1 | AL | AX |
| MUL WORD1 | WORD1 | 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
; ------------------------------------------------------------
BYTE1   DB      80H
BYTE2   DB      40H
WORD1   DW      8000H
WORD2   DW      2000H
; ------------------------------------------------------------
MAIN    PROC    NEAR                ;Procedimiento principal
        CALL    C10MUL              ;Llama a la rutina MUL
        CALL    D10IMUL             ;Llama a la rutina IMUL
        MOV     AX,4C00H            ;Sale al DOS
        INT     21H
MAIN    ENDP
;               Ejemplos de MUL:
;               ----------------
C10MUL  PROC
        MOV     AL,BYTE1            ;Byte por byte
        MUL     BYTE2               ; el producto en AX

        MOV     AX,WORD1            ;Palabra por palabra
        MUL     WORD2               ;  el producto en DX:AX

        MOV     AL,BYTE1            ;Byte por palabra
        SUB     AH,AH               ;  extiende el multiplicando en AH
        MUL     WORD1               ;  el producto queda en DX:AX
        RET
C10MUL  ENDP
;               Ejemplos de IMUL:
;               -----------------
D10IMUL PROC
        MOV     AL,BYTE1            ;Byte por byte
        IMUL    BYTE2               ;  el producto en AX

        MOV     AX,WORD1            ;Palabra por palabra
        IMUL    WORD2               ;  el producto en DX:AX

        MOV     AL,BYTE1            ;Byte por palabra
        CBW                         ;  extiende el multiplicando en AH
        IMUL    WORD1               ;  el producto queda en DX:AX
        RET
D10IMUL 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:

$$\begin{array}{r} 13 \\ \times \ 12 \\ \hline 156 \end{array} \qquad \begin{array}{r} 65 \\ \times \ 12 \\ \hline 780 \end{array}$$

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 × 12 × 100)
+   780       (65 × 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      6400H
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
Z10ZERO     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 | × | palabra 2 |
| palabra 2 | × | palabra 1 |
| palabra 1 | × | palabra 2 |
| palabra 1 | × | 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

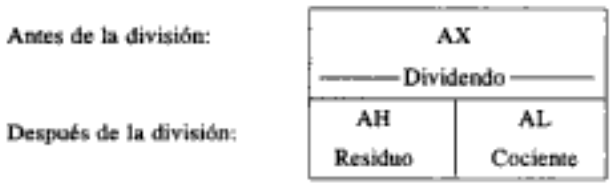
Para la división, la instrucción DIV (dividir) maneja datos sin signo y la IDIV (división entera) maneja datos con signo. Usted es responsable de seleccionar la instrucción apropiada. El formato general para DIV/IDIV es

| [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.

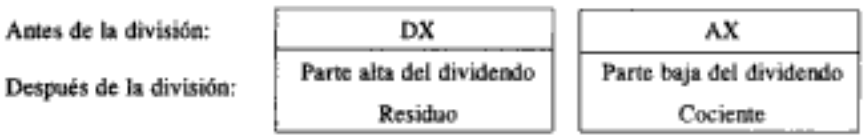
### Palabra entre byte

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.



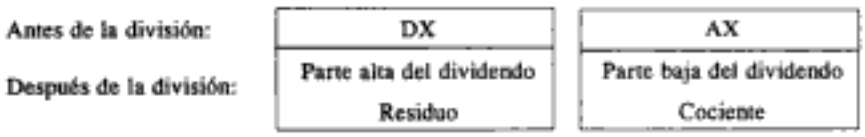

### Palabra doble entre palabra

Para esta operación, el dividendo está en el par DX:AX y el divisor es una palabra en memoria o 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 signo +16,383 (7FFFH). Tenemos:




### Palabra cuádruple entre palabra doble

Al dividir una palabra cuádruple entre una palabra doble, el dividendo está en el par EDX:EAX y el divisor está en una palabra doble en memoria o en otro registro. Después de la división, el residuo está en el EDX y el cociente en el EAX.



### 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:

```
BYTE1   DB      ?
WORD1   DW      ?
DWORD1  DD      ?
```

| ... | | DIVISOR | DIVIDENDO | COCIENTE | RESIDUO |
|---|---|---|---|---|---|
| DIV | BYTE1 | BYTE1 | AX | AL | AH |
| DIV | WORD1 | WORD1 | DX:AX | AX | DX |
| DIV | DWORD1 | DWORD1 | EDX:EAX | EAX | EDX |

**Residuo.** Si divide 13 entre 3, el resultado es $4\frac{1}{3}$, 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 $\frac{1}{3}$ 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 BYTE1 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 WORD1 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 C0H (–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,WORD1            ;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,WORD1            ;  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
                                    ;Palabra doble / palabra
        MOV     DX,WORD2            ;  dividendo en DX:AX
        MOV     AX,WORD3            ;  residuo:cociente en DX:AX
        IDIV    WORD1
        MOV     AX,WORD1            ;Palabra / palabra
        CWD                         ;  extiende el dividendo en DX
        IDIV    WORD3               ;  residuo:cociente en DX:AX
        RET
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) |
| 4 | 0000 | 0002 |

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:

```
        SUB   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 en AX
```

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,01
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        ;16 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 palabra DATAY; (b) la palabra doble que empieza en DATAX por la palabra doble en DATAY. Almacene el 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) la 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 de 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 más eficiente de corrimiento a la izquierda de cuatro bits. Corrija el ejemplo para un corrimiento a la 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):

$$\begin{array}{rl} 38 & \text{hex} \\ 34 & \text{hex} \\ \hline 6C & \text{hex} \end{array}$$

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
; ------------------------------------------
ASC1    DB      '578'               ;Datos
ASC2    DB      '694'
ASCSUM  DB      '0000'
; ------------------------------------------
MAIN    PROC    NEAR
        CLC                         ;Limpia bandera de acarreo
        LEA     SI,ASC1+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, ASC1 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 ASC1 contiene 38H y ASC2 contiene 34H. El primer ejemplo resta ASC2 (34H) de ASC1 (38H). AAS no necesita hacer un ajuste, ya que el dígito de la derecha es menor que A:

```
                        AX      AF
MOV   AL,ASC1   ;0038
SUB   AL,ASC2   ;0004   0
AAS             ;0004   0
OR    AL,30H    ;0034
```

El segundo ejemplo resta ASC1 (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,ASC1   ;00FC   1
AAS             ;FF06   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 (0AH) 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,0FH | ;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,0FH         ;Limpia el 3 ASCII
A20:
        MOV     AL,[SI]             ;Carga el carácter ASCII
        AND     AL,0FH              ;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],30H
        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 (0AH), 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 | ;Ajuste 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 ASC1 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 procesamiento de 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
;-------------------------------------------------
ASC1     DB      '057836'          ;Datos
ASC2     DB      '069427'
BCD1     DB      '000'
BCD2     DB      '000'
BCDSUM   DB      4 DUP(0)
;-------------------------------------------------
MAIN     PROC    NEAR
         LEA     SI,ASC1+4         ;Inicializa ASC1
         LEA     DI,BCD1+2
         CALL    B10CONV           ;Llama la rutina para convertir
         LEA     SI,ASC2+4         ;Inicializa ASC2
         LEA     DI,BCD2+2
         CALL    B10CONV           ;Llama la rutina para convertir
         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:
;                --------------------
C10ADD   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    C20                ;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 (0AH), 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 × 1 = | 4 | 4 × 01H = | 4H |
| | 3 × 10 = | 30 | 3 × 0AH = | 1EH |
| | 2 × 100 = | 200 | 2 × 64H = | C8H |
| | 1 × 1000 = | 1000 | 1 × 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 (0AH), dando los multiplicadores 1, 10 (0AH), 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 (0AH) 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,4C00H
        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 ⌐C | 1 | 2 |

Como el cociente (1) ahora es menor que el divisor (0AH) 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 (0AH) 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, despliega salario
            .MODEL SMALL
            .STACK 64
; -----------------------------------------------------
            .DATA
LEFCOL      EQU     28                  ;Equivalencia para la pantalla
RITCOL      EQU     52
TOPROW      EQU     10
BOTROW      EQU     14

HRSPAR      LABEL   BYTE                ;Lista de parámetros de horas
MAXHLEN     DB      6                   ;-----------------------------------
ACTHLEN     DB      ?
HRSFLD      DB      6 DUP(?)

RATEPAR     LABEL   BYTE                ;Lista de parámetros de sueldo por hora
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 o Esc para salir

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    E10RATE             ;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+1        ;Coloca el cursor
            MOV     COL,LEFCOL+3
            CALL    Q20CURS
            INC     ROW
            MOV     AH,09H
            LEA     DX,MESSG1           ;Indicación del número de horas
            INT     21H
            MOV     AH,0AH
            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,0AH
            LEA     DX,RATEPAR          ;Acepta el sueldo por hora
            INT     21H
            RET
 B10INPT    ENDP
;                   Procesa las horas:
;                   ------------------
 D10HOUR    PROC    NEAR
            MOV     NODEC,00
            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)

```
E10RATE   ENDP
;                 Multiplica, redondea y recorre:
;                 -------------------------------------------------
F10MULT   PROC    NEAR
          MOV     CX,05
          LEA     DI,ASCWAGE          ;Designa el salario ASCII
          MOV     AX,3030H            ;  a los 30
          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 inicio
G30:
          CMP     BYTE PTR[SI],'.'
          JNE     G40                 Si está en la posición dec, entonces saltar
          DEC     SI
G40:
          CMP     DX,00               ;Si DX:AX < 10,
          JNZ     G50
          CMP     AX,0010             ;  operación terminada
          JB      G60
G50:
          DIV     TENWD               ;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,LEPCOL+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:
;                   ----------------------
L10PAUS     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     16H
            RET
L10PAUS     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)

```
M100:     RET
M10ASBI   ENDP
;                    Recorre toda la pantalla:
;                    -------------------------------------
Q10SCR    PROC   NEAR
          MOV    AX,0600H
          MOV    BH,30H              ;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,16H              ;Atributo
          MOV    CH,TOPROW
          MOV    CL,LEFCOL
          MOV    DH,BOTROW
          MOV    DL,RITCOL
          INT    10H
          RET
Q15WIN    ENDP
;                    Coloca el cursor:
;                    -------------------------
Q20CURS   PROC   NEAR
          MOV    AH,02H
          SUB    BH,BH
          MOV    DH,ROW              ;Designa el renglón
          MOV    DL,COL              ;Designa la columna
          INT    10H
          RET
Q20CURS   ENDP
          END    BEGIN
```

**Figura 14-7** (continuación)

| | |
|---|---|
| E10RATE | 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 despliéguelo 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 los 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, 'Paper....',17, '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 DIRECTO 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): $2 \times 9 = 18$.
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 no es cero, | 0002 | |
| limpiar y sumar 0AH (10 decimal) | 000C | (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.

```
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 binario:
;               --------------------------
C10CONV PROC
        MOV     AH,MONIN            ;Configura el mes
        MOV     AL,MONIN+1
        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      ?
SAVEMON   DB      ?
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 la entrada
          LEA     DX,MONTAB       ;Dirección de la tabla
          ADD     DX,AX           ;  más desplazamiento
          MOV     AH,09H          ;Despliega
          INT     21H
          RET
C10MONTH  ENDP
.386
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 el teclado
                MOV     AH,10H          ;Petición de entrada
                INT     16H             ;Llama al BIOS
                RET
E10INPT         ENDP

Q10SCR          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

Q20CURS         PROC    NEAR
                MOV     AH,02H          ;Petición para colocar el cursor
                MOV     BH,00           ;Página
                MOV     DH,10           ;Renglón
                MOV     DL,24           ;Columna
                INT     10H
                RET
Q20CURS         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, ..., o aún para 5, 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 |
| 4,250.01-6,000.00 | .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]             ;#Existencia(1) : tabla
        JNE     A30                 ;No es igual, salir
        MOV     AL,STOKNIN+1        ;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,4C00H            ;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. El factor de ajuste compensa nuestro impuesto calculado en tasas altas, mientras que las tasas bajas se aplican a niveles menores de ingresos. Las entradas para los ingresos gravables contienen el 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 × 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 C0                      MOV   ES,AX         ;  segmentos
0007 FC                         CLD
0008 8D 3E 0003 R               LEA   DI,STOKTAB    ;Inicializa dirección
000C                    A20:                        ;  de la tabla
000C B9 0003                    MOV   CX,03         ;Compara 3 bytes
000F 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 1D                      JB   A40            ;  menor, no hay entrada
0019 03 F9                      ADD   DI,CX         ;Sumar CX al desplazamiento
001B 83 C7 0A                   ADD   DI,10         ;Siguiente elemento de la tabla
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: | + 0AH |
| 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  40 DUP(40H)                     ;espacios en blanco EBCDIC
        DB  0F0H,0F1H,0F2H,0F3H, ...,0F9H   ;EBCDIC 0-9
        DB  198 DUP(40H)                    ;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      0F0H,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 principal
        CALL  Q10CLR            ;Limpia la pantalla
        LEA   SI,DISPROW
A20LOOP:
        CALL  C10HEX            ;Traduce
        CALL  D10DISP           ;  y despliega
        CMP   HEXCTR,0FFH       ;¿Es el último número hex (FF)?
        JE    A50               ;  sí, termina
        INC   HEXCTR            ;  no, obtener el siguiente hex
        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 blanco
        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 F1 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 00H 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)

                 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
NAMEPAR 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, '$'
ENDADDR DW     ?
MESSG1  DB     'Name? ', '$'
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:
;               ----------------------------------------
B10READ PROC
        MOV    AH,09H
        LEA    DX,MESSG1        ;Despliega indicación
        INT    21H
        MOV    AH,0AH
        LEA    DX,NAMEPAR       ;Acepta nombre
        INT    21H
        MOV    AH,09H
        LEA    DX,CRLF          ;Retorno/avance de línea
        INT    21H

        MOV    BH,00            ;Limpia los caracteres después del nombre
        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
B10READ 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 × 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 siguiente
        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 × 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.)